Collusion Resistant Traitor Tracing from Learning with Errors

Rishab Goyal
UT Austin
rgoyal@cs.utexas.edu

Venkata Koppula
UT Austin
kvenkata@cs.utexas.edu

Brent Waters
UT Austin
bwaters@cs.utexas.edu[*]

Abstract

In this work we provide a traitor tracing construction with ciphertexts that grow polynomially in $\log(n)$ where n is the number of users and prove it secure under the Learning with Errors (LWE) assumption. This is the first traitor tracing scheme with such parameters provably secure from a standard assumption. In addition to achieving new traitor tracing results, we believe our techniques push forward the broader area of computing on encrypted data under standard assumptions. Notably, traitor tracing is substantially different problem from other cryptography primitives that have seen recent progress in LWE solutions.

We achieve our results by first conceiving a novel approach to building traitor tracing that starts with a new form of Functional Encryption that we call Mixed FE. In a Mixed FE system the encryption algorithm is bimodal and works with either a public key or master secret key. Ciphertexts encrypted using the public key can only encrypt one type of functionality. On the other hand the secret key encryption can be used to encode many different types of programs, but is only secure as long as the attacker sees a bounded number of such ciphertexts.

We first show how to combine Mixed FE with Attribute-Based Encryption to achieve traitor tracing. Second we build Mixed FE systems for polynomial sized branching programs (which corresponds to the complexity class LOGSPACE) by relying on the polynomial hardness of the LWE assumption with super-polynomial modulus-to-noise ratio.

1 Introduction

In a (traitor) tracing [CFN94] system an authority runs a setup algorithm that takes in a security parameter λ and the number, n, of users in the system. The setup outputs a public key pk, master secret key msk, and n secret keys $(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n)$. The system has an encryption algorithm that uses the public key pk to create a ciphertext for a message m that is decryptable by any of the n secret keys, but where the message will be hidden from any user who does not have access to the keys. Finally, suppose that some subset S of users collude to create a decoding box D which is capable of decrypting ciphertexts with some non-negligible probability. The tracing property of the system states that there exists an algorithm Trace, which given the master secret and oracle access to D, outputs a set of users T where T contains at least one user from the colluding set S and no users outside of S.

Existing approaches for achieving collusion resistant broadcast encryption can be fit in the framework of Private Linear Broadcast Encryption (PLBE) introduced by Boneh, Sahai, and Waters (BSW) [BSW06]. In a PLBE system the setup algorithm takes as input a security parameter λ and the number of users n. It outputs a public key pk, master secret key msk, and n private keys $\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n$ where a user with index j is given key sk_j. Any of the private keys is capable of decrypting a ciphertext ct created using pk. However, there is an additional $\mathsf{TrEncrypt}$ algorithm that takes in the master secret key, a message and an index i. This produces a ciphertext that only users with index $j > i$ can decrypt. Moreover, any adversary produced

[*]Supported by NSF CNS-1228599 and CNS-1414082. DARPA through the U.S. Office of Naval Research under Contract N00014-11-1-0382, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

decryption box D that was created with private keys in the set of S will not be able to distinguish between encryptions to index $i-1$ or i, where $i \notin S$. In addition, encryptions of two different messages m_0, m_1 to index n must be indistinguishable.

The tracing system is setup by simply running the PLBE setup and distributing each PLBE key to the corresponding user. To trace the set of colluding parties given a decoding box D, the tracing algorithm first measures (with several samples) the probability that D correctly decrypts a ciphertext encrypted to index i for all $i \in [0, n]$. If the box D originally decrypted with probability ϵ, then there must exists some index i where the probability the box decrypts on index $i-1$ is at least ϵ/n more than the probability it decrypts on ciphertexts encrypted to index i, as by PLBE security D can not decrypt encryptions to index n with non-negligible probability. At this point the tracing algorithm marks user i as a colluder.

Currently, there are three approaches to building PLBE. The most basic approach is to simply create n public/private key pairs under a standard IND-CPA secure public key encryption system. A PLBE ciphertext is formed by encrypting the message m to each user's public key individually and concatenating all of the sub-ciphertexts to form one long ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_n)$. A user with secret key sk_i in the system will decrypt by running decryption on ct_i and ignore the rest of the ciphertext components. To $\mathsf{TrEncrypt}$ to index i simply encrypt the all 0's string in first i ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_i$ in place of the message. The index hiding property follows directly from IND-CPA security of the underlying encryption scheme as without secret key i no attacker can distinguish whether ct_i is an encryption of the message or all 0's string.

The above approach works because there is a portion of the ciphertext ct_i dedicated to each user i in the system which is not touched during the decryption process of other users with keys sk_j for $j \neq i$. This dedicated ciphertext space strategy makes it easy to silently kill user i's ability to access the message in a way unnoticeable to other users, but also inherently requires a ciphertext size that grows linearly in n. In order to achieve PLBE with sublinear size ciphertexts one needs to implement some form of computing on encrypted data.

BSW [BSW06] provided the first construction that achieved PLBE with ciphertext growth that was sublinear in n. They leveraged composite order bilinear groups to achieve ciphertexts that grew proportionally to $\sqrt{n}$. While future variants [BW06, GKSW10, Fre10] used bilinear maps to obtain additional properties, the ciphertext size for all bilinear-map based constructions remained stuck at the $\sqrt{n}$ mark.

Several years later Boneh and Zhandry [BZ14] showed how to utilize indistinguishability obfuscation and apply punctured programming techniques to achieve the ideal case where ciphertexts grow polynomially in $\log(n)$ and λ. The downside of applying indistinguishability obfuscation is that all current obfuscation candidates are based on non-standard multilinear map group assumptions, and several such multilinear candidates have been attacked (see [CLT14, CHL+15, CGH+15, BGH+15, CLLT16, CLLT17, BWZ14, HJ16, Hal15, CFL+16, MSZ16, CJL16, ADGM16] and the references therein). (One could also achieve similar results from using the functional encryption scheme of Garg et al. [GGH+13], but this also relies on multilinear maps.) This leaves open the following question:

Can we build secure traitor tracing with $\textsc{poly}(\lambda, \log(n))$*-sized ciphertexts from standard assumptions?*

Our Results

In this work we resolve the above question by providing a traitor tracing construction with ciphertexts that grow polynomially in $\log(n)$ and λ and prove it secure under the Learning with Errors (LWE) assumption. This is the first traitor tracing scheme with such parameters that is provably secure from a standard assumption. In addition to achieving new traitor tracing results, we believe our techniques push forward the broader area of computing on encrypted data under standard assumptions. Notably, traitor tracing is substantially different problem from other cryptography primitives that have seen recent progress in LWE solutions.

We achieve our result by first conceiving a novel approach to building traitor tracing that starts with a new form of Functional Encryption that we call Mixed FE. In a Mixed FE system the encryption algorithm is bimodal and works with either a public key or master secret key. Ciphertexts encrypted using the public key can only encrypt one type of functionality. On the other hand the secret key encryption can be used to encode many different types of programs, but is only secure as long as the attacker sees a bounded number of such ciphertexts.

We first show how to combine Mixed FE with Attribute-Based Encryption to achieve traitor tracing. Second we show under the LWE assumption how to construct Mixed FE systems for polynomial sized branching programs (which corresponds to the complexity class LOGSPACE).

1.1 Technical Overview

We now give a technical overview of our work. This overview is broken into four parts. In the first part we review the BSW notion of Private Linear Broadcast Encryption and its transformation into a traitor tracing system. Along the way we discover that the PLBE definitions as presented in [BSW06] do *not* imply traitor tracing. We then show how to repair the argument by giving the attacker an additional oracle encryption query in the PLBE definitions. Second, we present the notion of Mixed FE and show how an ABE and Mixed FE system (for the right functionalities) can be used to construct a PLBE system. The third part of our overview describes a new LWE toolkit which includes "enhanced" versions of lattice trapdoor sampling algorithms with additional security properties. Finally, we outline our main ideas for constructing the Mixed FE system and proving it secure under the LWE assumption.

Part 1: Breaking and Repairing the PLBE to Tracing Argument. First, let us review the PLBE algorithms as defined in [BSW06]. A PLBE scheme consists of a setup, encryption, decryption and trace-encryption algorithm. The setup algorithm outputs a public key, a master secret key and n secret keys, one for each index in $[n]$. The encryption/decryption algorithms are self-explanatory; the trace-encryption algorithm is a special encryption algorithm that requires the master secret key, and can be used to encrypt a message to any index $i \in [0, n]$. The output ciphertext can be decrypted only by secret keys for indices $j > i$. BSW defined three security properties. The first security property (public to zero-index indistinguishability) requires that a standard encryption of message m is indistinguishable from a trace-encryption of m to the index 0, even when the adversary has all the n secret keys. The second security property (index hiding) states that a trace-encryption of m to index $i-1$ is indistinguishable from a trace-encryption of m to index i, even when the adversary has all the secret keys except the i^{th} one. Finally, the third security property states that trace-encryption of m_0 to index n is indistinguishable from trace-encryption of m_1 to index n, even when the adversary is given all n secret keys.

BSW argued that these three properties of PLBE are sufficient for constructing a traitor tracing (TT) scheme. In their transformation, the TT public key and n secret keys are set to be the PLBE public key and n secret keys, respectively. The TT encryption/decryption algorithms are identical to the PLBE encryption/decryption algorithms. Finally, the tracing algorithm uses the PLBE trace-encryption algorithm. Given a decoder box D, the tracing algorithm encrypts random messages to each index, and checks if D can decrypt it correctly. If the decoder box is ϵ-successful[1] in decrypting (standard) encryptions, then it is also ϵ successful in decrypting trace-encryptions to index 0 (via the first security property). Next, note that the decoder box cannot decrypt trace-encryptions to index n (via the message indistinguishability property). Therefore, there must exist an index $i^* \in [n]$ where the success of the decoder box in decrypting trace-encryptions to index $i^* - 1$ is at least ϵ/n more than its success in decrypting trace-encryptions to i^*. This index i^* must be one of the indices queried by the adversary (since if the adversary does not have a key for index i^*, then the decoder box must not be able to distinguish between trace-encryptions to $i^* - 1$ and i^*). For each index i, the tracing algorithm computes an estimate of the decoder box's success probability in decrypting random trace-encryptions for index i. For all indices i where the measured success probabilities for $i - 1$ and i are substantially different, user i is declared to be a traitor.

At an intuitive level, it seems like the BSW transformation should work. However, here we argue that it is indeed possible to have a PLBE scheme secure under the original BSW definition, but produce an insecure TT scheme in this regard. The problem lies in the fact that there is a "semantic gap" between the

[1] A decoder box is said to be ϵ-successful if its probability of correctly decrypting a ciphertext is at least ϵ, where the probability is taken over the choice of the ciphertext and D's random coins.

TT definition and the PLBE definition. The TT definition considers an attacker that produces a (stateless) decoder D whose success on decrypting multiple trace-encryptions is measured, whereas the PLBE definition considers indistinguishability on a single ciphertext (in particular, no ciphertext queries). Diving deeper, we show a separation by adding a feature to a PLBE scheme where the feature does not impact PLBE security, but results in an insecure TT scheme.

Given any secure PLBE scheme P, consider a scheme P' defined as follows. The setup algorithm of P' is similar to the setup of P, except it also samples an additional PRF key K as part of the master secret key (we will assume the PRF has single bit output). The (standard) encryption algorithm computes a ciphertext ct using the underlying scheme's encryption algorithm, chooses a uniformly random bit b and outputs (ct, b). The trace-encryption of message m is the ciphertext $\mathrm{ct}' = (\mathrm{ct}, y = \mathrm{PRF}_K(i))$ where ct is the ciphertext obtained from the trace-encryption algorithm of P. It is easy to see that the new scheme satisfies all three PLBE security definitions, since there are no encryption queries allowed in the PLBE scheme beyond the challenge ciphertext.

However, it is possible to construct a decoding box using only secret key for index n such that the trace algorithm falsely accuses some user $i < n$. The decoder D, on input of a ciphertext $\mathrm{ct}' = (\mathrm{ct}, y)$, tests if $y = 1$. If so, it decrypts the ciphertext using key sk_n; otherwise it outputs a random message. Using PRF security, we can argue that there exists an index $i < n$ such that $\mathrm{PRF}_K(i-1) = 1$ and $\mathrm{PRF}_K(i) = 0$ with high probability. In this case the probability that D decrypts ciphertexts for index $i-1$ will be measurably different than the case it decrypts ciphertext for index i. Thus user i will be flagged as a colluder.

We repair the BSW transformation from PLBE to TT by considering a modified set of PLBE security definitions and prove that these do imply TT. We do so in two steps. First, we consider a *decoder-based* version of the BSW PLBE definitions. For concreteness, let us consider the index hiding definition. The decoder-based version of index hiding version states that no adversary, given all secret keys except the i^{th} one, can produce a decoder box D and a message m such that D can distinguish between trace-encryptions of m to index $i-1$ and trace-encryptions of m to index i. Decoder-based versions of the other properties are defined similarly.

Now that we have decoder-based PLBE definitions that align with the decoder in the TT definitions, it is fairly straightforward to prove that the BSW transformation implies TT. The downside of introducing decoder-based PLBE definitions is that they are more difficult to work with as a target for a construction. We address this issue by circling back to the original (BSW) PLBE definitions, and augmenting them by allowing an attacker to make an apriori bounded number of queries to an encryption oracle. We show that 1-query PLBE implies decoder-based PLBE. This gives us an easier target (that is, 1-query PLBE).

Before describing the transformation from 1-query PLBE to decoder-based PLBE, we would like to point out that if the BSW definitions were augmented to allow an unbounded number of ciphertext queries, then decoder-based security follows immediately. For instance, let us consider the index hiding game. The reduction algorithm (that reduces unbounded-query PLBE to decoder-based PLBE) receives a decoder box D from the attacker. Given the unbounded queries, the reduction algorithm can measure (within reasonable accuracy) the success probabilities of D for indices $i-1$ and i, and therefore, whether it can use D to distinguish between an encryption to index $i-1$ and i. However, with only 1 encryption query no such precise measurement is possible. Therefore, showing an attacker on decoder-based PLBE security implies and attacker on 1-query PLBE is a bit tricky. The reduction algorithm, after receiving the decoder box and message m from the adversary, chooses a random index $i^* \in \{i-1, i\}$, and queries the challenger for encryption of m for index i^*. It receives a ciphertext ct. Next, it queries the challenger with challenge message m, and receives a challenge ciphertext ct^*. The reduction algorithm checks if $D(\mathsf{ct}) = D(\mathsf{ct}^*)$; if so, it guesses that m was encrypted for index i^*. We would like to point out that choosing query index i^* uniformly at random from $\{i-1, i\}$ (as opposed to just fixing one of the two) is important for our analysis. The complete details of our analysis can be found in Section 4.1.

Impact on prior TT works using PLBE framework. Traitor tracing schemes that had secret key tracing would need a new proof under the new PLBE definitions with 1-query allowed. We believe the bilinear map constructions [BSW06, GKSW10, Fre10] are likely secure under this definition, but showing this is outside scope of this paper. Note that same problem is not present in PLBE with public trace-encryption (e.g.

[BW06]), since the public key allows the reduction algorithm to generate ciphertexts.

Part 2: Constructing PLBE from Mixed FE. The hardness of constructing a PLBE scheme stems from the fact that it needs to satisfy the following three properties at the same time. First, a PLBE scheme needs to provide an Attribute-Based Encryption (ABE) like functionality where each secret key is associated with an "index", and each ciphertext is associated with an *index comparison* predicate. Second, the scheme must provide a Broadcast Encryption (BE) like compactness guarantee which is that the size of ciphertexts must be short. Third and most importantly, it must provide a Predicate Encryption (PE) like security, that is the ciphertexts must not reveal any more information about the associated index comparison predicate other than what can be learnt by running decryption.

In this work, instead of directly building a PLBE scheme, we further reduce the task to constructing a new form of FE scheme called Mixed FE. We show how Mixed FE can be combined with ABE for circuits to obtain PLBE. At a very high level, our approach is to decouple the functionality (delivering the message to users) and security requirements of a PLBE scheme, and deal with them separately.

We begin by informally introducing the notion of Mixed FE. A Mixed FE scheme consists of a setup, normal (or public key) encryption, *secret key encryption* , key generation, and decryption algorithm. The setup algorithm takes as input the security parameter λ and description of the function class $\mathcal{F}$, and outputs the public parameters pp and the master secret key msk. The normal encryption algorithm only takes as input the public parameters pp, and outputs a (normal) ciphertext ct. The secret key encryption algorithm takes as input the master secret key msk and a function $f \in \mathcal{F}$, and outputs a (secret key) ciphertext ct. The key generation algorithm takes as input the master secret key msk and a message m, and outputs a key sk_m. The decryption algorithm takes as input a ciphertext ct and a secret key sk_m, and outputs a single bit. Now for correctness we require that decrypting a secret key encryption of any function f using a secret key sk_m outputs the evaluation of function f on message m, i.e. $f(m)$. Whereas the decryption algorithm (almost) always outputs 1 when given a normal ciphertext as input, irrespective of the secret key used. Thus, one could visualize the normal encryption algorithm as always encrypting a "canonical" *always-accepting* function.

Intuitively, security states that no attacker should be able to distinguish between two ciphertexts that decrypt to same values under all the secret keys in attacker's possession. Now since there are two separate encryption algorithms, we have two different security properties. The first property says that secret key encryptions of two functions f_0 and f_1 should be indistinguishable if for every key in attacker's possession, the output of f_0, f_1 is identical. We call this function indistinguishability property. The second property says that it should be hard to distinguish between a normal (public key) encryption and secret key encryption of a function f, where $f(m)$ must be equal to 1 for all keys sk_m in attacker's possession. We call this accept indistinguishability property.

We show that we can construct a PLBE scheme from a (key-policy) ABE scheme and a Mixed FE scheme. The idea is to encrypt a message using the ABE system with attribute being set to be a Mixed FE ciphertext. And, each user's secret key will be an ABE private key for the policy circuit being the Mixed FE decryption circuit with a Mixed FE secret key corresponding to user's index hardwired. The high level intuition is that when the attribute is a normal FE ciphertext then all Mixed FE keys decrypt it to 1, thus any user with an appropriate ABE key could perform the decryption. Whereas if the attribute is set to be a secret key ciphertext, then we can control the users who can decrypt it.

Formally, the scheme works as follows. During setup, the algorithm samples both ABE and Mixed FE key pairs $(\mathsf{abe.pp}, \mathsf{abe.msk})$, $(\mathsf{mixed.pp}, \mathsf{mixed.msk})$. To compute the i^{th} user's private key, it samples a Mixed FE secret key $\mathsf{mixed.sk}_i$ for input i, and also computes an ABE key $\mathsf{abe.sk}_i$ for predicate $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$, i.e. Mixed FE decryption circuit with key $\mathsf{mixed.sk}_i$ hardwired. And the ABE key $\mathsf{abe.sk}_i$ is set to be the i^{th} user's private key. Now to encrypt a message m, the algorithm simply runs the ABE encryption algorithm with attributes being set to be a Mixed FE ciphertext $\mathsf{ct}_{\mathsf{attr}}$. For standard PLBE encryption, $\mathsf{ct}_{\mathsf{attr}}$ is computed as a Mixed FE normal ciphertext. And for PLBE index encryption to some index i, $\mathsf{ct}_{\mathsf{attr}}$ is computed as a Mixed FE secret key encryption of function *greater than* i. Lastly, the PLBE decryption is same as the ABE decryption algorithm.

Correctness can be observed directly. For standard PLBE ciphertext, $\mathsf{ct}_{\mathsf{attr}}$ is a normal FE ciphertext which decrypts to 1, thus the predicate $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$ is satisfied for all i. Therefore, by ABE correctness, the ABE decryption algorithm will output the message m. For PLBE index i ciphertext, $\mathsf{ct}_{\mathsf{attr}}$ is a Mixed FE secret key encryption of function '$> i$' which decrypts to 1 for all keys $\mathsf{mixed.sk}_j$ with $j > i$, thus the predicate is satisfied for all users with indices larger than i. Therefore, by ABE correctness, ABE decryption algorithm will output the message m whenever $j > i$. For proving security, we rely on the fact that Mixed FE ciphertexts are indistinguishable to any adversary that does not have distinguishing secret keys. For instance, suppose there exists an adversary that can distinguish between PLBE normal encryptions and index 0 encryptions, then such an adversary can also be used to distinguish between Mixed FE normal ciphertexts and secret key ciphertexts encrypting function '> 0' (note that this is an always-accepting function). Thus, such an attack can be used to break accept indistinguishability property of Mixed FE scheme. Similarly, we can argue index hiding and message hiding security of the construction by reducing to Mixed FE and ABE (selective) security, respectively. Now if the Mixed FE scheme is 1-query secure, then so will be the PLBE scheme.

Now the size of ciphertexts has only poly-log dependence on the number of users n as required. Because each user can be uniquely identified using a bit string of length $\log n$, so the length of attribute (Mixed FE ciphertext) will be polynomial in $\log n$, and thus the PLBE ciphertext which is in turn an ABE ciphertext will have length polynomial in $\log n$ as well. Also, note that to use the above transformation it is sufficient to construct a Mixed FE scheme that supports comparison operation on $\log n$ bit strings. In this work, we show how to construct a Mixed FE scheme for any class of polynomial sized branching program from the Learning with Errors assumption.[2] Our construction relies only on the polynomial hardness of LWE, although we require super-polynomial modulus-to-noise ratio. Since we already have circuit ABE schemes from the LWE assumption [GVW13, BGG+14], combining that with our Mixed FE construction we get collusion resistant traitor tracing from the LWE assumption as well.

Looking back, it is easy to observe that Mixed FE for branching programs that supports comparison functionality is sufficient for our application. However, as a design choice, here we instead chose to construct Mixed FE for general polynomial length branching programs as it is possible that this generalization leads to more applications in the future. Moreover, focusing on logarithmic length branching programs supporting comparisons, instead of general branching programs, did not lead to any significant simplification in the Mixed FE construction or its proof.

Part 3: An Enhanced LWE Toolkit. Before describing our LWE-based construction for Mixed FE, we define new "enhanced" properties for lattice trapdoors that will be useful in our work and we believe it will find more applications in the future. In many LWE-based works, in addition to the LWE assumption itself, a critical tool has been the notion of lattice trapdoors [Ajt99, GPV08]. Lattice trapdoor samplers consist of a pair of algorithms $\mathsf{TrapGen}$ and $\mathsf{SamplePre}$. The trapdoor generation algorithm $\mathsf{TrapGen}$ outputs a matrix $\mathbf{A}$ (that defines the lattice), and a trapdoor $T_{\mathbf{A}}$. The preimage sampling algorithm $\mathsf{SamplePre}$ takes as input a matrix $\mathbf{Z}$, a trapdoor for matrix $\mathbf{A}$, a Gaussian parameter σ and outputs a matrix $\mathbf{U}$ such that $\mathbf{U}$ maps $\mathbf{A}$ to $\mathbf{Z}$ (that is, $\mathbf{A} \cdot \mathbf{U} = \mathbf{Z}$).[3]

These algorithms satisfy the following properties. The matrix $\mathbf{A}$ output by the trapdoor generation algorithm 'looks like' a uniformly random matrix; we call this *well-sampledness of matrix* property. Secondly, the matrix output by $\mathsf{SamplePre}$ is indistinguishable from a matrix drawn from a discrete Gaussian distribution with parameter σ over the set of all matrices $\mathbf{V}$ such that $\mathbf{A} \cdot \mathbf{V} = \mathbf{Z}$. In particular, if $\mathbf{Z}$ is chosen uniformly at random, then the output of $\mathsf{SamplePre}$ 'looks like' a matrix $\mathbf{U}$ drawn from a discrete Gaussian distribution with parameter σ; we call this the *well-sampledness of preimage*. Lattice trapdoors with these properties have found a remarkable number of applications in building LWE-based cryptography.

In this work, we introduce two new *enhanced* properties for lattice trapdoors. The first property is the *row removal* property, which can be intuitively described as follows. Consider a setting where an adversary

[2] Note that this also gives us an alternate construction for selectively-secure private-key FE with bounded collusions [SS10, GVW12].

[3] Although the notion of preimage sampling is usually defined w.r.t. vectors instead of matrices, here we stick to using matrices for technical reasons discussed later in Section 7.

specifies some 'target vectors', and the challenger must output a matrix $\mathbf{A}$ and a matrix $\mathbf{U}$ such that $\mathbf{U}$ maps some of the rows of $\mathbf{A}$ to the target vectors, and maps the remaining rows to uniformly random vectors. Then, these rows targetting uniformly random vectors can be removed from the trapdoor sampling. In particular, the challenger can sample a shorter matrix $\mathbf{B}$ with trapdoor, extend $\mathbf{B}$ with uniformly random vectors to get $\mathbf{A}$, and set $\mathbf{U}$ to be a matrix that maps $\mathbf{B}$ to the target vectors. These two scenarios will be indistinguishable for the PPT adversary.

The second property is called the *target switching* property. In this setting, consider an adversary that specifies two matrices $\mathbf{Z}_0, \mathbf{Z}_1$ and a set of 'target' indices such that the rows of $\mathbf{Z}_0$ and $\mathbf{Z}_1$ agree on these target indices. The challenger is supposed to sample a matrix $\mathbf{A}$ with a trapdoor, compute a matrix $\mathbf{U}$ that maps $\mathbf{A}$ to $\mathbf{Z}_0$[4] and output $\mathbf{U}$ together with the rows of $\mathbf{A}$ corresponding to the target indices *and only those rows*. Then, the challenger can switch the $\mathbf{U}$ to map $\mathbf{A}$ to $\mathbf{Z}_1$, and the target switching property requires that this change is indistinguishable to the adversary (note that this would not be possible if the adversary receives any of the non-target rows of $\mathbf{A}$). Moreover, the adversary is allowed to adaptively query for different target vectors/indices in both these games.

Now that we have these enhanced properties, let us discuss how to construct lattice trapdoors with these enhanced properties (using standard lattice trapdoors). Our construction is similar to the $\mathsf{SampleLeft}/\mathsf{SampleRight}$ algorithms of [ABB10, CHKP10]. The enhanced trapdoor generation algorithm uses the standard trapdoor sampling algorithm to sample two matrices $\mathbf{A}_1, \mathbf{A}_2$ together with the respective trapdoors $T_{\mathbf{A}_1}, T_{\mathbf{A}_2}$. It outputs $\mathbf{A} = [\mathbf{A}_1|\mathbf{A}_2]$ as the matrix, and $T_{\mathbf{A}} = (T_{\mathbf{A}_1}, T_{\mathbf{A}_2})$ as the trapdoor. To sample a matrix $\mathbf{U}$ that maps $\mathbf{A}$ to $\mathbf{Z}$, the preimage sampling algorithm first chooses a uniformly random matrix $\mathbf{W}$ (of same dimensions as $\mathbf{Z}$). It then uses $T_{\mathbf{A}_1}$ to compute a matrix $\mathbf{U}_1$ that maps $\mathbf{A}_1$ to $\mathbf{W}$, and uses $T_{\mathbf{A}_2}$ to compute a matrix $\mathbf{U}_2$ that maps $\mathbf{A}_2$ to $\mathbf{Z} - \mathbf{W}$. The final preimage matrix is set to be $\begin{bmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{bmatrix}$. We use the matrix well-sampledness and preimage well-sampledness of the standard lattice trapdoors to prove these enhanced properties; the detailed proof can be found in Section 7.2.

Part 4: Constructing Mixed FE from LWE. Here we outline our Mixed FE construction for polynomial sized (leveled) branching program from the Learning with Errors assumption. The main ingredient of our construction is the "enhanced" lattice trapdoor sampling procedure $\mathsf{LT}_{\mathsf{en}} = (\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ discussed above.

First, let us recall the notion of leveled branching programs. A leveled branching program of length ℓ and width w can be represented using w states per level, 2ℓ state transition functions $\pi_{j,b}$ for each level $j \leq \ell$, an input-selector function $\mathsf{inp}(\cdot)$ which determines the input read at each level, and an accepting and rejecting state. The program execution starts at state $\mathsf{st} = 1$ of level 1. Suppose the branching program reads the first input bit (say b) at level 1 (i.e., $\mathsf{inp}(1) = 1$). Then, the state of the program changes from st to $\pi_{1,b}(\mathsf{st})$. Such a process is carried out (iteratively) until the program's final state at level ℓ is computed. Depending upon the final state, the program either accepts or rejects.

For ease of exposition we will start with a simpler goal of constructing a 0-query secure Mixed FE scheme for class of width-w read-once branching programs where each input bit is read once and in an ascending order. Below we first outline a construction for such a 0-query system as it contains most of the central ideas, but is easier to digest. Later we discuss the modifications with which we can improve it to be a secure 1-query scheme (and more generally, q-query secure for any polynomial q) as well as expand the function class to arbitrary polynomial sized branching programs.

Moving on to our 0-query Mixed FE construction, the master secret key consists of two sets of matrices and some trapdoor information. The first set, labeled as 'randomization' matrices, consists of 4ℓ matrices $\{\mathbf{B}_{i,b}, \mathbf{C}_{i,b}\}_{i,b}$ for $i \in [\ell], b \in \{0,1\}$. And, the second set, labeled as 'program' matrices, consists of $w\ell$ matrices $\{\mathbf{P}_{i,v}\}_{i,v}$ for $i \in [\ell], v \in [w]$. Here the $\mathbf{C}_{i,b}$ matrices are sampled uniformly at random from $\mathbb{Z}_q^{n \times m}$, whereas the remaining (randomization and program matrices) are sampled jointly with common trapdoors

[4]Strictly speaking, we require $\mathbf{U}$ to map the target vectors of $\mathbf{A}$ to the target vectors of $\mathbf{Z}_0$, but the remaining vectors of $\mathbf{A}$ *approximately* map to the corresponding vectors of $\mathbf{Z}_0$.

(per level). Basically, for each level $i \in [\ell]$, we sample a $(w+2)n \times m$ matrix $\mathbf{M}_i$ as

$$(\mathbf{M}_i, T_i) \leftarrow \mathsf{EnTrapGen}(1^{(w+2)n}, 1^m, q).$$

Now each $\mathbf{M}_i$ matrix is parsed as four $n \times m$ matrices stacked on top of each other, where first two matrices are the randomization matrices and the remaining w matrices are the program matrices for i^{th} level. That is, for each i,

$$\begin{bmatrix} \mathbf{B}_{i,0} \\ \mathbf{B}_{i,1} \\ \mathbf{P}_{i,1} \\ \vdots \\ \mathbf{P}_{i,w} \end{bmatrix} = \mathbf{M}_i.$$

All ℓ trapdoors $T_1, \ldots, T_\ell$ are stored as the trapdoor information in the master secret key. The public parameters, on the other hand, only include the matrix dimensions, LWE modulus and noise parameters, but none of these matrices or trapdoor information.

At a high level, the encryption and key generation algorithms will adhere to the following structure. To (secret key) encrypt a branching program, the trapdoors will be used to sample 2ℓ low norm matrices $\{\mathbf{U}_{i,b}\}_{i,b}$ (two per level) such that each matrix $\mathbf{U}_{i,b}$ encodes the corresponding state transition function by mapping/targetting level i 'program' matrices to level $i+1$ 'program' matrices as per the transition function $\pi_{i,b}$. Now the secret key for an input x will consist of $\ell+1$ key vectors $\{\mathbf{t}_i\}_i$. The first key component $\mathbf{t}_1$ will contain the program matrix $\mathbf{P}_{1,1}$ (which represents the starting state) plus some randomization component generated using the level 1 randomization matrix $\mathbf{B}_{1,b}$. The remaining ℓ key vectors will have two components — the first component will cancel the previous randomization component, and the second component will add new randomization terms.[5] The idea is that if decryption is performed honestly, then all the randomization terms will get cancelled and the final output will reflect the output of the branching program.

So this way the program matrices will be tied in such a manner that they encode the state transition information and they can be used to perform the branching program execution. And the randomization matrices are added to make sure that — (1) the computation is hidden at each step, and (2) if ciphertext matrices and key vectors are combined in any inadmissible way, then the randomization components do not get cancelled. Let us now look at how to execute the above ideas.

Key Generation. The key generation algorithm takes as input a string x and generates key vectors $\{\mathbf{t}_i\}_i$ as follows. It chooses ℓ uniform secret vectors $\mathbf{s}_i \in \mathbb{Z}_q^n$ for $i \in [\ell]$ and $\ell+1$ noise vectors $\mathbf{e}_i \in \mathbb{Z}_q^m$ for $i \in [\ell+1]$. It also chooses a *short* secret vector $\widetilde{\mathbf{s}} \in \mathbb{Z}_q^n$, and sets key vectors as:

$$\forall\ i \in [\ell+1], \quad \mathbf{t}_i = \begin{cases} \mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} + \mathbf{e}_1 & \text{if } i = 1 \\ -\mathbf{s}_{i-1} \cdot \mathbf{C}_{i-1,x_{i-1}} + \mathbf{s}_i \cdot \mathbf{B}_{i,x_i} + \mathbf{e}_i & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell \cdot \mathbf{C}_{\ell,x_\ell} + \mathbf{e}_{\ell+1} & \text{if } i = \ell+1 \end{cases}$$

In words, the randomization component (likewise, cancellation component) added in the i^{th} key vector ($(i+1)^{th}$ key vector) is an LWE sample where the public matrix used depends on the i^{th} bit of input x. Looking ahead, choosing the 'randomization' matrices depending on the string x would assert that the ciphertext matrices can not be arbitrarily combined to learn meaningful terms.

Normal Encryption. The normal (public key) encryption algorithm simply samples 2ℓ random short matrices $\{\mathbf{U}_{i,b}\}_{i,b}$ as $\mathbf{U}_{i,b} \leftarrow \chi^{m \times m}$, where χ is the noise distribution chosen during setup.

[5]Technically, the last key vector will only remove the previous randomization component. It doesn't add a new randomization term.

Secret Key Encryption. Moving on to the secret key encryption algorithm, on input the master secret key and a branching program $\mathsf{BP} = \left(\{\pi_{i,b}\}_{i,b}, \mathsf{acc}, \mathsf{rej}\right)$, it samples low norm matrices $\{\mathbf{U}_{i,b}\}$ as follows. It first chooses two 'program' matrices for last level $\ell + 1$ as $\mathbf{P}_{\ell+1,\mathsf{rej}} = \mathbf{0}^{n\times m}$ and $\mathbf{P}_{\ell+1,\mathsf{acc}} \leftarrow \mathbb{Z}_q^{n\times m}$. That is, for accepting state, it chooses a random program matrix, and for rejecting state it sets the matrix to be all zeros. Next, using the i^{th} trapdoor T_i (included in the master secret key) it runs the $\mathsf{EnSamplePre}$ algorithm to sample the ciphertext (transition) matrices $\mathbf{U}_{i,0}, \mathbf{U}_{i,1}$ such that they map/target matrix $\mathbf{M}_i$ as follows:

$$\begin{bmatrix} \mathbf{B}_{i,0} \\ \mathbf{B}_{i,1} \\ \mathbf{P}_{i,1} \\ \vdots \\ \mathbf{P}_{i,w} \end{bmatrix} \xrightarrow{\mathbf{U}_{i,0}} \begin{bmatrix} \mathbf{C}_{i,0} \\ \$ \\ \mathbf{P}_{i+1,\pi_{i,0}(1)} \\ \vdots \\ \mathbf{P}_{i+1,\pi_{i,0}(w)} \end{bmatrix}, \qquad \begin{bmatrix} \mathbf{B}_{i,0} \\ \mathbf{B}_{i,1} \\ \mathbf{P}_{i,1} \\ \vdots \\ \mathbf{P}_{i,w} \end{bmatrix} \xrightarrow{\mathbf{U}_{i,1}} \begin{bmatrix} \$ \\ \mathbf{C}_{i,1} \\ \mathbf{P}_{i+1,\pi_{i,1}(1)} \\ \vdots \\ \mathbf{P}_{i+1,\pi_{i,1}(w)} \end{bmatrix}.$$

Here we use '\$' to denote a uniformly random $n \times m$ matrix of appropriate dimension. In words, the structure we enforce here is that the matrix $\mathbf{U}_{i,b}$ targets the $\mathbf{B}_{i,b}$ randomization matrix to its $\mathbf{C}_{i,b}$ counterpart, and the $\mathbf{B}_{i,1-b}$ randomization matrix to a random matrix. Additionally, $\mathbf{U}_{i,b}$ encodes the information about transition function $\pi_{i,b}$ by targetting the level i program matrices to their level $i + 1$ counterparts as per $\pi_{i,b}$. Thus, from the perspective of both correctness and security, this guarantees that a key vector $\mathbf{t}_i$ for some input x must be combined with ciphertext component $\mathbf{U}_{i,x_i}$ as otherwise randomization matrix would be mapped to a random matrix, thereby destroying the underlying structure.

Decryption. First, let us focus on decrypting a secret key encryption of branching program BP using a secret key $\{\mathbf{t}_i\}_i$ corresponding to an input x. Intuitively, one could visualize the matrices $\{\mathbf{U}_{i,0}, \mathbf{U}_{i,1}\}_i$ in the ciphertext as "encodings" of the branching program state transition functions $\pi_{i,0}, \pi_{i,1}$ (respectively). Therefore, decrypting the ciphertext using secret key for some input x will be analogous to evaluating the branching programs BP on input x directly. Recall that we assumed (for ease of exposition) that the branching programs are read-once and input bits are read sequentially in ascending order. Thus, the first input bit x_1 is read at level 1. Then evaluation of BP at level 1 would map the state $\mathsf{st}_1 = 1$ at level 1 to state $\mathsf{st}_2 = \pi_{1,x_1}(1)$ at level 2. Analogously, the decryptor can compute

$$\begin{aligned} \mathbf{t}_1 \cdot \mathbf{U}_{1,x_1} + \mathbf{t}_2 &\approx (\mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}) \cdot \mathbf{U}_{1,x_1} + \mathbf{t}_2. \\ &\approx \mathbf{s}_1 \cdot \mathbf{C}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \mathbf{t}_2. \\ &\approx \cancel{\mathbf{s}_1 \cdot \mathbf{C}_{1,x_1}} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \left(\cancel{-\mathbf{s}_1 \cdot \mathbf{C}_{1,x_1}} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2}\right). \\ &\approx \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2}. \end{aligned}$$

In general, if the program state at level i during execution is st_i, then the decryptor will accumulate the term of the form $\mathbf{s}_i \cdot \mathbf{B}_{i,x_i} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i,\mathsf{st}_i}$ by successively summing and multiplying secret key and ciphertext components as

$$(\cdots((\mathbf{t}_1 \cdot \mathbf{U}_{1,x_1} + \mathbf{t}_2) \cdot \mathbf{U}_{2,x_2} + \mathbf{t}_3) \cdots + \mathbf{t}_i)$$

This can be verified as follows. We know that the bit read at level i is x_i, thus the new state at level $i + 1$ will be $\mathsf{st}_{i+1} = \pi_{i,x_i}(\mathsf{st}_i)$. Now the accumulated sum-product during decryption will be

$$\begin{aligned} (\mathbf{s}_i \cdot \mathbf{B}_{i,x_i} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i,\mathsf{st}_i}) \cdot \mathbf{U}_{i,x_i} + \mathbf{t}_{i+1} &\approx \cancel{\mathbf{s}_i \cdot \mathbf{C}_{i,x_i}} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i+1,\mathsf{st}_{i+1}} + \left(\cancel{-\mathbf{s}_i \cdot \mathbf{C}_{i,x_i}} + \mathbf{s}_{i+1} \cdot \mathbf{B}_{i+1,x_{i+1}}\right). \\ &\approx \mathbf{s}_{i+1} \cdot \mathbf{B}_{i+1,x_{i+1}} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i+1,\mathsf{st}_{i+1}}. \end{aligned}$$

Therefore, the invariant is maintained. Continuing this way, the decryptor can iteratively compute the sum-product combining all key and ciphertext components. Note that (by definition) adding in the $(\ell+1)^{th}$ key component t_ℓ does not introduce a term like $\mathbf{s}_{\ell+1} \cdot \mathbf{B}_{\ell+1,x_{\ell+1}}$ to the sum-product, thus the accumulated term at the top will be $\approx \widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{st}_{\ell+1}}$, where $\mathsf{st}_{\ell+1}$ is either acc or rej depending on $\mathsf{BP}(x)$. Finally, the decryptor simply checks whether the norm of the final sum-product term is small or not. Recall that the program matrix for last level corresponding to rejecting state is set to be all zeros, i.e. $\mathbf{P}_{\ell+1,\mathsf{rej}} = \mathbf{0}^{n \times m}$. Therefore, if $\mathsf{BP}(x) = 0$, then the norm of the final sum-product term will be small which the decryptor can test and output 0. Otherwise, with high probability the final sum-product term will be large and it outputs 1.

By the above analysis, correctness follows in the case where ciphertext is a secret key encryption. The correctness of decryption when the ciphertext is a normal (public key) ciphertext follows from the fact that the ciphertext matrices $\{\mathbf{U}_{i,0}, \mathbf{U}_{i,1}\}_i$ are independently sampled random short matrices.

0-*Query Security.* To prove 0-query security of our construction, we need to argue that it satisfies both function indistinguishability as well as accept indistinguishability properties. We start by proving function indistinguishability security. Recall that in 0-query function indistinguishability security game, an adversary submits two branching programs $\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)}$ and is allowed to make a polynomial number of key queries such that for each queried input x, $\mathsf{BP}^{(0)}(x) = \mathsf{BP}^{(1)}(x)$ (i.e., every secret key given out has same output on both the challenge programs). The adversary receives secret key encryption of either $\mathsf{BP}^{(0)}$ or $\mathsf{BP}^{(1)}$, and its goal is to distinguish between them.

Although the full security proof is technically involved, the main ideas behind our proof are very intuitive. Before diving into the proof structure, we point out that the construction described above has to be slightly modified for proving security. Below we describe our proof ideas as well as discuss the modifications required along the way.

At a high level, our idea is to "hardwire" the output of the challenge branching programs in every secret key given to the adversary. Note that the security definition states that both challenge programs must evaluate to the same value on all queried inputs, thus we only need to hardwire a single value in each key. For ease of exposition, assume that the adversary makes exactly one secret key query. (In the general case of polynomially many key queries, the proof proceeds by hardwiring the level 1 components in all secret keys, followed by level 2 hardwiring and so on.) Let $\{\mathbf{U}_{i,b}\}_{i,b}$ be the challenge ciphertext and $\{\mathbf{t}_i\}_i$ be the secret key computed by the challenger. Our hardwiring strategy works as follows. We start by re-writing the second secret key vector $\mathbf{t}_2$ in terms of $\mathbf{t}_1$ as follows:

$$\begin{aligned}
\mathbf{t}_2 &= -\mathbf{s}_1 \cdot \mathbf{C}_{1,x_1} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2 \\
&= -\mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} \cdot \mathbf{U}_{1,x_1} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2 \\
&= -\mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} \cdot \mathbf{U}_{1,x_1} - \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2 \\
&= -\left(\mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}\right) \cdot \mathbf{U}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2 \\
&= -(\mathbf{t}_1 - \mathbf{e}_1) \cdot \mathbf{U}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2
\end{aligned}$$

Here st_2 is the state of the challenge branching program encrypted (after one step is executed). Now in the above term, we can *smudge* the term $\mathbf{e}_1 \cdot \mathbf{U}_{1,x_1}$ by appropriately choosing the noise distributions, i.e. $\mathbf{e}_2 >> \mathbf{e}_1 \cdot \mathbf{U}_{1,x_1}$.[6] (Note that since we require smudging here, thus the LWE modulus q needs to be super-polynomial in the lattice dimension.) Thus, the second key component can be indistinguishably computed as follows without requiring any explicit knowledge of the $\mathbf{C}_{1,x_1}$ matrix.

$$\mathbf{t}_1 = \mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} + \mathbf{e}_1.$$

$$\mathbf{t}_2 = \begin{array}{c} -(\mathbf{t}_1 - \mathbf{e}_1) \cdot \mathbf{U}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} \\ + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2 \end{array} \xrightarrow{\mathsf{Smudging}} \mathbf{t}_2 = \begin{array}{c} -\mathbf{t}_1 \cdot \mathbf{U}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} \\ + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2 \end{array}.$$

Next, we use the row removal property of our enhanced trapdoor sampling algorithms to remove the $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ rows from the first matrix $\mathbf{M}_1$ and instead sample these randomly. To understand why this can be done recall that in the actual construction the encryptor needs the ability to create a ciphertext for *any* branching program that could be chosen even after all the keys have been distributed. That is, the encryptor must be able to sample matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$ such that they map level 1 program matrices $\{\mathbf{P}_{1,v}\}_v$ to level 2 program matrices $\{\mathbf{P}_{2,v}\}_v$ as per some transition functions $\{\pi_{1,b}\}_b$ as well as ensure that

[6]If we keep on smudging this way, then our noise distributions will have to grow by an exponential factor at each step. In the main body, we show how to avoid this by a better smudging argument.

$\mathbf{B}_{1,b} \cdot \mathbf{U}_{1,b} = \mathbf{C}_{1,b}$. Now since the keys contain the matrices $\mathbf{C}_{1,0}, \mathbf{C}_{1,1}$ and they could be given out even before matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$ are sampled, thus matrices $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ must be sampled together with $\{\mathbf{P}_{1,v}\}_v$ such that they share a common trapdoor.

However, at this stage in the proof the challenge branching program is (selectively) fixed ahead of any secret key queries. Therefore, in this context we can sample matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$ to only map level 1 program matrices to their level 2 counterparts, and simply set the matrices $\mathbf{C}_{1,b}$ as $\mathbf{C}_{1,b} = \mathbf{B}_{1,b} \cdot \mathbf{U}_{1,b}$ and use these to compute the secret keys. We would like to point out that in order to perform this row removal securely, it is important that $\mathbf{B}_{1,b} \cdot \mathbf{U}_{1,1-b} = \$$, that is matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$ map both matrices $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ to random and uncorrelated matrices.

Now once we have removed the $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ rows from the first matrix, we use the LWE assumption to switch the first key component $\mathbf{t}_1$ to a random vector. Note that at this point since matrices $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ are sampled uniformly (i.e., are no longer sampled with trapdoor information) and secret vector $\mathbf{s}_1$ is not used in computing the second key component $\mathbf{t}_2$, thus we can apply LWE to switch $\mathbf{t}_1$ to random, where the LWE secret is $\mathbf{s}_1$ and LWE public matrix will be $\mathbf{B}_{1,x_1}$.[7] Concretely, using LWE we can perform the following switch which essentially erases the information about the level 1 program matrix $\mathbf{P}_{1,1}$ from the secret keys, thereby rendering the program evaluation to start from level 2 and state st_2 instead.

$$\mathbf{t}_1 = \mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} + \mathbf{e}_1 \quad \xrightarrow{\mathsf{LWE}} \quad \mathbf{t}_1 = \$.$$
$$\mathbf{t}_2 = -\mathbf{t}_1 \cdot \mathbf{U}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2.$$

Now iteratively performing this *hardwiring* strategy (ℓ times) we end up switching all but last key components to be random vectors. Also, the last key component will contain the final program matrix which is either a random matrix or a zero matrix, depending on the program output. Thus, the key vectors contain no information about the 'program' matrices chosen during setup. At this point, the challenge matrices $\{\mathbf{U}_{i,b}\}_{i,b}$ still contain the information about the branching program encrypted in the form of mapping between level i and $i+1$ 'program' matrices, i.e. the state transition functions $\{\pi_{i,b}\}_{i,b}$. Finally, to argue indistinguishability here (i.e., between the challenge matrices) we use the target switching property of our enhanced trapdoors. We apply a bottom-up approach to execute this change. First, note that the level 1 program matrices do not explicitly appear anywhere, except that they are used to sample the level 1 ciphertext matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$. Thus, we can use the target switching property to switch the targets of matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$. Observe that this now removes the information about level 2 program matrices as well as the level 1 transition functions of challenge branching programs. Next, by the same principle, we can perform the same target switching step for $\mathbf{U}_{2,0}, \mathbf{U}_{2,1}$ and continue so on. If we keep on performing the *target switching* step this way until the top, then the challenge ciphertext will contain no information about the challenge programs (i.e., their state transition functions) thereby completing our claim of function indistinguishability.[8] This completes the first proof.

The proof of 0-query accept indistinguishability security of our construction is similar, but more technical due to the fact that we need to argue that the challenge ciphertext is indistinguishable from random short matrices.

However, for our PLBE construction, the Mixed FE scheme must handle one ciphertext query as well, and it is not clear how to prove the above construction to be 1-query secure directly. The bottleneck is the fact that in the above proof strategy we *hardwire* all secret key components to match the output of challenge program. Now if the adversary is allowed to make a secret key encryption query, then it is not clear that how would the challenger still program the secret key vectors. To get around this problem, we expand our system such that it consists of λ *pairs* of 0-query sub-systems. Very briefly, during encryption, the algorithm now also samples a λ-bit string tag randomly and depending on each bit of tag, it chooses one sub-system in each

[7]In the general case of multiple key queries the LWE public matrices will be both $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ and LWE secret will consist of all the secret key vectors $\mathbf{s}_i$ that are chosen independently and *per key*.

[8]Technically, we can not apply the target switching property here. Because the target switching property only guarantees that targets being switched are approximately mapped, whereas here we target exactly. Therefore, we also need to add some noise in the targetted 'program' matrices before running $\mathsf{EnSamplePre}$ algorithm. For simplicity, we avoid this modification.

pair and runs the 0-query encryption for that sub-system. Now during key generation, it (linearly) secret shares the starting program across these λ pairs of sub-systems such that same secret share is used for both sub-systems in each pair. Then, it runs the 0-query key generation algorithm for all these 2λ sub-systems with their corresponding secret shares as the starting program matrices. For decryption, the sub-systems chosen during encryption are combined with their counterparts in the secret keys and 0-query decryption is performed in these sub-systems along with a (linear) reconstruction on top of the output. More details are provided in the main body.[9]

This completes the technical overview of our construction.

Relation to recent LWE-based schemes. There have been several recent works that have advanced the state of the art in computing branching programs on encrypted data with the goal of reducing security to LWE or LWE-like assumptions [GGH15, BVWW16, BV15, GKW17b, CC17, GKW17a, WZ17]. While our construction above benefits from that lineage we wish to briefly call out a few important distinctions.

First from a purely mechanical perspective, the construction of our Mixed FE scheme is structurally very different from the constructions of the aforementioned primitives. Very briefly, in all previous constructions the evaluator multiplies a set of matrices, and sums them up to get the final output. Whereas in our construction, we do not use this 'one-shot' approach for evaluation. Instead, we multiply a component from the ciphertext with a secret key component, then add in another secret key component, multiply this sum with another ciphertext component and so on. Thus, our mechanism of combining the secret key and ciphertext components is much different that what was used in prior works.[10]

Second we structure our proof of security to hardwire the outputs for keys one level at a time until we hit the final output level in which we have the final outputs hardwired, but lost information about the program that got us there. In this sense at a high level this leveled programming proof structure much more closely resembles that of garbled circuit proofs. Thus, we need to develop new lower LWE specific techniques to match these goals. In contrast works such as [BVWW16, GKW17b, GKW17a, WZ17] have a different aim of loosing all meaningful information when a secret is not known.

1.2 Some Future Directions

Our construction of Mixed FE relied on the LWE assumption and leveraged certain algebraic properties in that setting. An intriguing question is whether there are other avenues for achieving Mixed FE. A natural path is to build Mixed FE with a garbled circuits [Yao86] backbone. If one starts with the bounded key FE scheme of Gorbunov, Vaikuntanathan and Wee (GVW) [GVW12] and flips [AGVW13, BS15, KMUW17] the semantics of message and function one can get a secret key FE scheme that is secure for an unbounded number of private keys and bounded number of ciphertexts. To make it a Mixed FE system we would somehow connect a public key mode of encryption to the scheme. One possible path is to use a "blinded" [BLSV17] form of garbled circuits as the underlying 1-bounded scheme in the GVW transformation. (Building on [DG17a, DG17b] blinded garbled circuits were recently used to give anonymous IBE from new assumptions). It seems possible that this approach could lead to a scheme with the accept indistinguishability property if no encryption oracle queries are allowed. However, there appears to be technical difficulties in making a public key generated ciphertext indistinguishable from a master secret key generated ciphertext when the attacker gets oracle queries. That being said, we believe that a garbled circuit approach remains a plausible future direction.

We remark that even if a garbled circuit approach becomes possible, the requirement for an ABE scheme supporting circuits will still indirectly require the LWE assumption given the state of the art. In addition,

[9]We would like to point out that the above idea could also be used to improve the Mixed FE construction to be q-query secure for any polynomial q. The idea will be to sample tag strings $\mathtt{tag}$ from a larger alphabet instead of $\{0,1\}$. However, we only focus on 1-query security as it is sufficient for our result.

[10]Although one can always express such a nested matrix multiplication and addition mechanism using only a sequence of matrix multiplications with much larger (and repetitive) matrices, we point out that the underlying structure of such matrices as well as the modified evaluation algorithm will still be much different from those used in the previous works.

we expect that our LWE toolkit and underlying construction ideas will have future value in any case.

A second interesting direction is whether there are other applications that can leverage an FE system that has a bimodal encryption where the public key and master secret key support different spaces of messages or functions. In our Mixed FE system the public key only supported the always accept function, but there could conceivably be other variants of interest.

Finally, a natural open question is to construct traitor tracing schemes with public traceability from LWE. Currently, it is unclear if achieving public tracing is an easier task than building general public key functional encryption.

1.3 Additional Related Work

Connections to differential privacy. Dwork et al. [DNR$^+$09] showed that existence of collusion resistant traitor tracing schemes implies hardness results for efficient differentially private [DMNS06] data sanitization. In particular, they show that if there exists a traitor tracing scheme with ciphertexts of size $s(\lambda, n)$, then there exists a database of size n and a query class $\mathcal{Q}$ of size $2^{s(\lambda,n)}$ such that it is hard to sanitize the database D for query class $\mathcal{Q}$ in a differentially private manner. Combining our LWE based construction with the result of Dwork et al. , we get an LWE-based hardness result for differentially private sanitization with query space of size $2^{\mathsf{poly}(\lambda,\log n)}$. We note that Goyal et al. [GKRW17] and Kowalczyk et al. [KMUW17] recently achieved similar differential privacy impossibility results from the the security of bilinear map assumptions and one way functions respectively.

Weaker notions of traitor tracing. Since the notion of traitor tracing was first proposed [CFN94], several relaxed variants have been studied in order to achieve short ciphertexts. The first natural relaxation is the bounded collusion setting, where we have an apriori bound k that is fixed during setup, and security is guaranteed only if the adversary gets at most k secret keys. Collusion bounded systems can either be constructed via combinatorial tools [CFN94, SW98, CFNP00, SSW01, PST06, BP08], or can be algebraic and constructed under different cryptographic assumptions such as DDH [KD98, BF99, KY02a, KY02b], bilinear DDH [CPP05, ADM$^+$07, FNP07] and LWE [LPSS14]. Recently, Agrawal et al. [ABP$^+$17] showed a transformation from inner product FE to collusion-bounded TT, resulting in algebraic constructions based on various assumptions such as DCR, DDH and LWE. In all the above works, the size of the ciphertext grows with the collusion bound.

The second relaxation is called threshold TT, introduced by [NP98, CFNP00]. In a threshold TT scheme, a threshold $\delta \in [0, 1]$ is chosen during setup, and the traceability guarantee only holds if the decoder box works with probability at least δ. Boneh and Naor [BN08] showed a threshold TT scheme where the ciphertexts have size $O(\lambda)$ and the secret keys have size $O(n^2\lambda/\delta^2)$. While this scheme achieves collusion resistance, the system must be configured with a specific δ value, and once it is set one will not necessarily be able to identify a traitor from a box D that works with smaller probability. In practice, it can be tricky to ascertain what threshold will actually be okay. This is because the encrypted messages could have redundancy, so even a decoder box with a small fraction of success might allow attacker to learn the underlying message.

Finally, in a recent work, Goyal et al. [GKRW17] introduced a new relaxation called *risky traitor tracing*. In this notion, the scheme is fully collusion resistant (and does not have the threshold restriction as above). Instead, the probability of tracing a traitor, given a successful decoding box, can be substantially smaller than 1. For instance, [GKRW17] showed a bilinear maps based construction where the ciphertext size grows as $\lambda \cdot k$, but the trace algorithm has only a k/n chance of catching a traitor. The authors show that this weaker notion is actually enough to achieve strong hardness results for differential privacy [DMNS06, DNR$^+$09], and also argue that in a certain 'continuous use' setting, the probability of tracing can be amplified back up to one. However, in general settings, the Goyal et al. tradeoff between the probability of catching a traitor and the size of ciphertexts might be undesirable.

1.4 Organization

In Section 2, we present the preliminaries required for this work. Next, in Section 3, we have the traitor tracing and PLBE definitions. This includes the new decoder-based and q-query PLBE definitions. In Section 4, we show how 1-query PLBE implies decoder based PLBE, and how decoder-based PLBE suffices for constructing traitor tracing schemes. Therefore, the problem of constructing a traitor tracing scheme reduces to the problem of constructing a 1-query PLBE scheme. For this, we introduce a new primitive called mixed FE in Section 5. In Section 6, we show how to construct 1-query PLBE using mixed FE and ABE (the syntax and security definitions of ABE can be found in Section A). Finally, in Section 8, we present our mixed FE construction (before presenting the mixed FE construction, in Section 7, we present new lattice tools which are required for our construction).

2 Preliminaries

Notations. Let PPT denote probabilistic polynomial-time. We will use lowercase bold letters for vectors (e.g. $\mathbf{v}$), uppercase bold letters for matrices (e.g. $\mathbf{A}$) and assume all vectors are row vectors. The j^{th} row of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}[j]$. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q. We represent $\mathbb{Z}_q$ as integers in the range $(-q/2, q/2]$. For a vector $\mathbf{v}$, we let $\|\mathbf{v}\|$ denote its ℓ_2 norm and $\|\mathbf{v}\|_\infty$ denote its infinity norm. Similarly, for matrices $\|\cdot\|$ and $\|\cdot\|_\infty$ denote their ℓ_2 and infinity norms (respectively).

We denote the set of all positive integers upto n as $[n] := \{1, \ldots, n\}$. Throughout this paper, unless specified, all polynomials we consider are positive polynomials. Also, we represent each a finite set on integers $S \subset \mathbb{N}$ as an *ordered* set $S = \{i_1, i_2, \ldots, i_n\}$, i.e. $i_j < i_k$ for every $1 \leq j < k \leq n$. For any finite set S, $x \leftarrow S$ denotes a uniformly random element x from the set S. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element x drawn from distribution $\mathcal{D}$. The distribution $\mathcal{D}^n$ is used to represent a distribution over vectors of n components, where each component is drawn independently from the distribution $\mathcal{D}$.

For two distributions X, Y, over a finite domain Ω, the statistical distance between X and Y is defined as $\mathsf{SD}(X, Y) \overset{\text{def}}{=} \frac{1}{2}\sum_{\omega \in \Omega} |X(\omega) - Y(\omega)|$. A family of distributions $\mathcal{D}_1 = \{\mathcal{D}_1(\lambda)\}_\lambda$ and $\mathcal{D}_2 = \{\mathcal{D}_2(\lambda)\}_\lambda$, parameterized by security parameter λ, are said to be statistically indistinguishable, represented by $\mathcal{D}_1 \approx_s \mathcal{D}_2$, if there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{SD}(\mathcal{D}_1(\lambda), \mathcal{D}_2(\lambda)) \leq \text{negl}(\lambda)$. For a family of distributions $\mathcal{D} = \{\mathcal{D}(\lambda)\}_\lambda$ over the integers, and integers bounds $B = \{B(\lambda)\}_\lambda$, we say that $\mathcal{D}$ is B-bounded if $\Pr[|x| \leq B(\lambda) : x \leftarrow \mathcal{D}(\lambda)] = 1$. In words, a B-bounded distribution is supported only on the range $[-B, B]$. Below we state the "smudging" lemma as it appears in prior works.

Lemma 2.1 (Smudging Lemma [AJW11, Lemma 2.1, paraphrased])**.** Let B_1, B_2 be two polynomials over the integers and let $\mathcal{D} = \{\mathcal{D}(\lambda)\}_\lambda$ be any B_1-bounded distribution family. Let $U = \{U(\lambda)\}_\lambda$ and $U(\lambda)$ denote the uniform distribution over integers $[-B_2(\lambda), B_2(\lambda)]$. The family of distributions $\mathcal{D}$ and U is statistically indistinguishable, $\mathcal{D} + U \approx_s U$, if there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $B_1(\lambda)/B_2(\lambda) \leq \text{negl}(\lambda)$.

2.1 Lattice Preliminaries

An m-dimensional lattice $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^m$. Given positive integers n, m, q and a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we let $\Lambda_q^{\perp}(\mathbf{A})$ denote the lattice $\{\mathbf{x} \in \mathbb{Z}^m \ : \ \mathbf{A} \cdot \mathbf{x}^T = \mathbf{0}^T \mod q\}$. For $\mathbf{u} \in \mathbb{Z}_q^n$, we let $\Lambda_q^{\mathbf{u}}(\mathbf{A})$ denote the coset $\{\mathbf{x} \in \mathbb{Z}^m \ : \ \mathbf{A} \cdot \mathbf{x}^T = \mathbf{u}^T \mod q\}$.

Discrete Gaussians. Let σ be any positive real number. The Gaussian distribution $\mathcal{D}_\sigma$ with parameter σ is defined by the probability distribution function $\rho_\sigma(\mathbf{x}) = \exp(-\pi \|\mathbf{x}\|^2 / \sigma^2)$. For any set $\mathcal{L} \subset \mathcal{R}^m$, define $\rho_\sigma(\mathcal{L}) = \sum_{\mathbf{x} \in \mathcal{L}} \rho_\sigma(\mathbf{x})$. The discrete Gaussian distribution $\mathcal{D}_{\mathcal{L},\sigma}$ over $\mathcal{L}$ with parameter σ is defined by the probability distribution function $\rho_{\mathcal{L},\sigma}(\mathbf{x}) = \rho_\sigma(\mathbf{x})/\rho_\sigma(\mathcal{L})$ for all $\mathbf{x} \in \mathcal{L}$.

The following lemma (Lemma 4.4 of [MR07], [GPV08]) shows that if the parameter σ of a discrete Gaussian distribution is small, then any vector drawn from this distribution will be short (with high probability).

Lemma 2.2. Let m, n, q be positive integers with $m > n$, $q \geq 2$. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be a matrix of dimensions $n \times m$, $\sigma = \tilde{\Omega}(n)$ and $\mathcal{L} = \Lambda_q^{\perp}(\mathbf{A})$. Then

$$\Pr[\|\mathbf{x}\| > \sqrt{m} \cdot \sigma : \mathbf{x} \leftarrow \mathcal{D}_{\mathcal{L},\sigma}] \leq \text{negl}(n).$$

Truncated Discrete Gaussians. The truncated discrete Gaussian distribution over $\mathbb{Z}^m$ with parameter σ, denoted by $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma}$, is same as the discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}^m,\sigma}$ except it outputs 0 whenever the ℓ_∞ norm exceeds $\sqrt{m} \cdot \sigma$. Note that, by definition, $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma}$ is $\sqrt{m} \cdot \sigma$-bounded. Also, by the above lemma we get that $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma} \approx_s \mathcal{D}_{\mathbb{Z}^m,\sigma}$.

2.1.1 Learning with Errors

The Learning with Errors (LWE) problem was introduced by Regev [Reg05], who showed that solving LWE on *average* is as hard as quantumly solving several standard lattice based problems in the worst case. The LWE assumption states that no polynomial time adversary can distinguish between the following oracles. In one case, the oracle chooses a uniformly random secret $\mathbf{s}$, and for each query, it chooses a vector $\mathbf{a}$ uniformly at random, scalar e from a noise distribution and outputs $(\mathbf{a}, \mathbf{s} \cdot \mathbf{a}^T + e)$. In the second case, the oracle simply outputs a uniformly random vector $\mathbf{a}$ together with a uniformly random scalar u. Regev showed that if there exists a polynomial time adversary that can break the LWE assumption, then there exists a polynomial time quantum algorithm that can solve some hard lattice problems in the worst case.

Several works also explored different variants of the LWE assumption, where the secret vector $\mathbf{s}$, public vectors $\mathbf{a}$ and noise are drawn from different distributions. In this work, we will be using two of these variants. First, we will be using the LWE version with *short secrets* (also known as the *normal form*), introduced by Applebaum et al. [ACPS09]. In this variant, the secret vector $\mathbf{s}$ is also drawn from the noise distribution. Applebaum et al. showed that this version is as hard as the LWE problem if the modulus is p^e for some prime p and integer e. This was later generalized to all moduli by Brakerski et al. [BLP$^+$13]. The second variant, which was proposed by Boneh et al. [BLMR13], allows the public vectors $\mathbf{a}$ to be chosen from the noise distribution as well. Boneh et al. showed that this version of LWE is as hard as standard LWE.

We will first present the LWE assumption in a general framework,[11] which captures the standard LWE, LWE with short secrets and LWE with short public vectors. In this framework, we will have an explicit security parameter λ, and the other parameters are allowed to grow as a function of the security parameter.

Definition 2.1 (Generalized Learning with Errors)**.** Fix any polynomial $n(\cdot)$, function $q(\cdot)$, secret distribution $\eta(\cdot)$, public vector distribution $\phi(\cdot)$ and noise distribution $\chi(\cdot)$, where $n : \mathbb{N} \to \mathbb{N}$, $q : \mathbb{N} \to \mathbb{N}$ and for each $\lambda \in \mathbb{N}$, $\eta(\lambda)$ and $\phi(\lambda)$ are distributions over $\mathbb{Z}_{q(\lambda)}^{n(\lambda)}$ and $\chi(\lambda)$ is a distribution over $\mathbb{Z}$. We say that the generalized LWE assumption $\mathsf{GLWE}_{n,q,\eta,\phi,\chi}$ holds if for any PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q = q(\lambda)$, $n = n(\lambda)$, $\eta = \eta(\lambda)$, $\phi = \phi(\lambda)$ and $\chi = \chi(\lambda)$, $\mathsf{Adv}_{\mathsf{GLWE},\mathcal{A}}^{n,q,\eta,\phi,\chi}(\lambda) \leq \text{negl}(\lambda)$, where

$$\mathsf{Adv}_{\mathsf{GLWE},\mathcal{A}}^{n,q,\eta,\phi,\chi}(\lambda) = \Pr\left[\mathcal{A}^{O_1^{\mathbf{s}}()}(1^\lambda) = 1 : \mathbf{s} \leftarrow \eta\right] - \Pr\left[\mathcal{A}^{O_2()}(1^\lambda = 1)\right],$$

and oracles $O_1^{\mathbf{s}}(), O_2()$ are defined as follows: oracle $O_1^{\mathbf{s}}()$ has $\mathbf{s} \in \mathbb{Z}_q^n$ hardwired, and on each query it chooses $\mathbf{a} \leftarrow \phi$, $e \leftarrow \chi$ and outputs $\left(\mathbf{a}, \mathbf{s} \cdot \mathbf{a}^T + e \bmod q\right)$, and oracle $O_2()$ (on each query) chooses $\mathbf{a} \leftarrow \phi$, $u \leftarrow \mathbb{Z}_q$ and outputs $(\mathbf{a}, u)$.

We now present different variants of LWE assumption, and discuss the parameters for which they are believed to be secure.

[11] Canetti and Chen [CC17] proposed the General LWE problem. However, their version requires the public vectors be sampled from a uniform distribution, whereas we require the public vectors to be sampled from non-uniform distributions. Also, it is possible to generalize our version further. Here, we present the minimal generalization that suffices for our work.

Assumption 1 (Learning with Errors)**.** Let $n : \mathbb{N} \to \mathbb{N}$ be a polynomial, and $q : \mathbb{N} \to \mathbb{N}$, $\sigma : \mathbb{N} \to \mathbb{R}^+$ be functions. The $\mathsf{LWE}_{n,q,\sigma}$ assumption states that $\mathsf{GLWE}_{n,q,\eta,\phi,\chi}$ holds, where $\eta(\lambda), \phi(\lambda)$ are uniform distributions over $\mathbb{Z}_{q(\lambda)}^{n(\lambda)}$, and $\chi(\lambda) \equiv \mathcal{D}_{\mathbb{Z},\sigma(\lambda)}$.

The following theorem shows that breaking LWE is as hard as solving hard lattice problems. In particular, given the current state of the art of lattice problems, the LWE assumption is believed to be true for any polynomial $n(\cdot)$ and functions $q(\cdot), \sigma(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$, the following constraints are satisfied: $0 < \sigma < q < 2^n$, $n \cdot q/\sigma < 2^{n^\epsilon}$ (for any constant $\epsilon < 1/2$) and $\sigma > 2\sqrt{n}$.

Theorem 2.1 (LWE to worst-case lattice problem [Reg05, Pei09, BLP+13])**.** Fix any polynomial $n(\cdot)$ and functions $q(\cdot), \sigma(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$, $0 < \sigma < q < 2^n$ and $\sigma > 2\sqrt{n}$. For every $\lambda \in \mathbb{N}$, let $\eta = \eta(\lambda)$ and $\phi = \phi(\lambda)$ denote the uniform distributions over $\mathbb{Z}_q^n$ and $\chi = \chi(\lambda) \equiv \mathcal{D}_{\mathbb{Z},\sigma}$. If there exists a PPT algorithm $\mathcal{A}$ and a non-negligible function $\epsilon_{\mathcal{A}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathsf{GLWE},\mathcal{A}}^{n,q,\eta,\phi,\chi}(\lambda) \geq \epsilon_{\mathcal{A}}(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ and a non-negligible function $\epsilon_{\mathcal{B}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all instances X of $\mathsf{GapSVP}_{n,n\cdot q/\sigma}$, $\mathcal{B}$ can solve X with probability at least $\epsilon_{\mathcal{B}}(\lambda)$.

Assumption 2 (LWE with Short Secrets)**.** Let $n : \mathbb{N} \to \mathbb{N}$ be a polynomial, and $q : \mathbb{N} \to \mathbb{N}$, $\sigma : \mathbb{N} \to \mathbb{R}^+$ be functions. The $\mathsf{LWE\text{-}ss}_{n,q,\sigma}$ assumption states that $\mathsf{GLWE}_{n,q,\eta,\phi,\chi}$ holds, where $\phi(\lambda)$ is the uniform distributions over $\mathbb{Z}_{q(\lambda)}^{n(\lambda)}$, $\eta(\lambda) \equiv \mathcal{D}_{\mathbb{Z}^{n(\lambda)},\sqrt{2}\sigma(\lambda)}$ and $\chi(\lambda) \equiv \mathcal{D}_{\mathbb{Z},\sigma(\lambda)}$.

The next theorem shows that breaking LWE with short secrets is as hard as breaking (standard) LWE, provided $0 < \sigma(\lambda) < q(\lambda) < 2^{n(\lambda)}$ and $\sigma(\lambda) > \lambda$.

Theorem 2.2 (LWE with Short Secrets [ACPS09, Lemma 2],[BLP+13, Lemma 2.12])**.** Fix any polynomial $n(\cdot)$ and functions $q(\cdot), \sigma(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$, $0 < \sigma < q < 2^n$ and $\sigma > \lambda$.[12] For every $\lambda \in \mathbb{N}$, let $\eta(\lambda) \equiv \mathcal{D}_{\mathbb{Z}_q^n,\sqrt{2}\sigma}$, $\phi(\lambda)$ the uniform distribution over $\mathbb{Z}_q^n$ and $\chi(\lambda) \equiv \mathcal{D}_{\mathbb{Z},\sigma}$. If there exists a PPT algorithm $\mathcal{A}$ and a non-negligible function $\epsilon_{\mathcal{A}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathsf{GLWE},\mathcal{A}}^{n,q,\eta,\phi,\chi}(\lambda) \geq \epsilon_{\mathcal{A}}(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ and a non-negligible function $\epsilon_{\mathcal{B}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathsf{GLWE},\mathcal{B}}^{n,q,\phi,\phi,\chi}(\lambda) \geq \epsilon_{\mathcal{B}}(\lambda)$.

Assumption 3 (LWE with Short Public Vectors)**.** Let $n : \mathbb{N} \to \mathbb{N}$ be a polynomial, $q : \mathbb{N} \to \mathbb{N}$, $\sigma : \mathbb{N} \to \mathbb{R}^+$ be functions, and $\{\chi(\lambda)\}_{\lambda\in\mathbb{N}}$ family of distributions over $\mathbb{Z}$. The $\mathsf{LWE\text{-}sp}_{n,q,\sigma,\chi}$ assumption states that $\mathsf{GLWE}_{n,q,\eta,\phi,\chi}$ holds, where $\eta(\lambda)$ is the uniform distributions over $\mathbb{Z}_{q(\lambda)}^{n(\lambda)}$, $\phi(\lambda) \equiv \mathcal{D}_{\mathbb{Z}^{n(\lambda)},\sigma(\lambda)}$.

The last theorem in this sequence shows a reduction from LWE with short public vectors to (standard) LWE with a lower dimension.

Theorem 2.3 (LWE with Short Public Vectors [BLMR13, Corollary 4.6])**.** Fix any polynomials $n(\cdot), k(\cdot)$ and functions $q(\cdot), \sigma(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $q = q(\lambda)$, $k = k(\lambda)$, $\sigma = \sigma(\lambda)$, $0 < \sigma < q < 2^n$, $k \geq 6n \log q$, and $\sigma \geq \sqrt{n \log q}$. For every $\lambda \in \mathbb{N}$, let $\phi(\lambda) \equiv \mathcal{D}_{\mathbb{Z}_q^k,\sigma}$, and let $\eta(\lambda), \phi'(\lambda)$ denote the uniform distributions over $\mathbb{Z}_q^k$ and $\mathbb{Z}_q^n$, respectively. Now for any distribution $\chi(\lambda)$ over $\mathbb{Z}$, if there exists a PPT algorithm $\mathcal{A}$ and a non-negligible function $\epsilon_{\mathcal{A}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathsf{GLWE},\mathcal{A}}^{k,q,\eta,\phi,\chi}(\lambda) \geq \epsilon_{\mathcal{A}}(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ and a non-negligible function $\epsilon_{\mathcal{B}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathsf{GLWE},\mathcal{B}}^{n,q,\phi',\phi',\chi}(\lambda) \geq \epsilon_{\mathcal{B}}(\lambda)$.

2.1.2 Lattice Trapdoors

Lattices with trapdoors are lattices that are indistinguishable from randomly chosen lattices, but have certain 'trapdoors' that allow efficient solutions to hard lattice problems.

A trapdoor lattice sampler [Ajt99, GPV08] consists of algorithms $\mathsf{TrapGen}$ and $\mathsf{SamplePre}$ with the following syntax and properties:

[12]Strictly speaking, it is only required that $\sigma > \sqrt{\ln n + \omega(1) \ln \lambda}$.

- $\mathsf{TrapGen}(1^n, 1^m, q) \to (\mathbf{A}, T_{\mathbf{A}})$: The lattice generation algorithm is a randomized algorithm that takes as input the matrix dimensions n, m, modulus q, and outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with a trapdoor $T_{\mathbf{A}}$.
- $\mathsf{SamplePre}(\mathbf{A}, T_{\mathbf{A}}, \sigma, \mathbf{u}) \to \mathbf{s}$: The presampling algorithm takes as input a matrix $\mathbf{A}$, trapdoor $T_{\mathbf{A}}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$ and a parameter $\sigma \in \mathbb{R}$ (which determines the length of the output vectors).[13] It outputs a vector $\mathbf{s} \in \mathbb{Z}_q^m$ such that $\mathbf{A} \cdot \mathbf{s}^T = \mathbf{u}^T$ and $\|\mathbf{s}\| \leq \sqrt{m} \cdot \sigma$.

We require these algorithms to satisfy the following well-sampledness properties. While these properties are similar 'in spirit' to the ones in previous works on lattice trapdoors [Ajt99, GPV08, MP12], there are a couple of differences. First, we present these properties as a security game between a challenger and a computationally bounded adversary.[14] Second, we separate out the dimensions of the matrix and the security parameter.

The first property (well-sampledness of matrix) states that the matrix output by $\mathsf{TrapGen}$ should look like a uniformly random matrix.

Definition 2.2 (Well-sampledness of Matrix)**.** Fix any function $q : \mathbb{N} \to \mathbb{N}$. A pair of trapdoor generation algorithms $\mathcal{T} = (\mathsf{TrapGen}, \mathsf{SamplePre})$ is said to satisfy the *q-well-sampledness of matrix* property if for any stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q = q(\lambda)$, $\mathsf{pr}_{\mathcal{T},\mathcal{A}}^{\mathrm{matrix},q}(\lambda) = \Pr[1 \leftarrow \mathsf{Expt}_{\mathcal{T},\mathcal{A}}^{\mathrm{matrix},q}(\lambda)] \leq 1/2 + \mathrm{negl}(\lambda)$, where $\mathsf{Expt}_{\mathcal{T},\mathcal{A}}^{\mathrm{matrix},q}(\lambda)$ is defined in Figure 1.

$\mathsf{Expt}_{\mathcal{T},\mathcal{A}}^{\mathrm{matrix},q}(\lambda)$

1. Adversary $\mathcal{A}$ receives input 1^λ and sends $1^n, 1^m$ such that $m > n \log q(\lambda) + \lambda$.
2. Challenger chooses $b \leftarrow \{0,1\}$ and $(\mathbf{A}_0, T_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$ and $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times m}$. It sends $\mathbf{A}_b$ to the adversary.
3. $\mathcal{A}$ outputs its guess b'. The experiment outputs 1 iff $b = b'$.

Figure 1: Experiment $\mathsf{Expt}_{\mathcal{T},\mathcal{A}}^{\mathrm{matrix},q}$

The next property states that the preimage of a uniformly random vector/matrix is indistinguishable from a matrix with entries drawn from Gaussian distribution.

Definition 2.3 (Well-sampledness of Preimage)**.** Fix any functions $q : \mathbb{N} \to \mathbb{N}$ and $\sigma : \mathbb{N} \to \mathbb{N}$. A pair of trapdoor generation algorithms $\mathcal{T} = (\mathsf{TrapGen}, \mathsf{SamplePre})$ is said to satisfy the (q, σ)-*well-sampledness of preimage* property if for any stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$, $\mathsf{pr}_{\mathcal{T},\mathcal{A}}^{\mathrm{preimg},q,\sigma}(\lambda) = \Pr[1 \leftarrow \mathsf{Expt}_{\mathcal{T},\mathcal{A}}^{\mathrm{preimg},q,\sigma}(\lambda)] \leq 1/2 + \mathrm{negl}(\lambda)$, where $\mathsf{Expt}_{\mathcal{T},\mathcal{A}}^{\mathrm{preimg},q,\sigma}(\lambda)$ is defined in Figure 2.

$\mathsf{Expt}_{\mathcal{T},\mathcal{A}}^{\mathrm{preimg},q,\sigma}(\lambda)$

1. Adversary $\mathcal{A}$ receives input 1^λ and sends $1^n, 1^m, 1^k$ such that $\sigma(\lambda) > \sqrt{n \cdot \log q \cdot \log m} + \lambda$ and $m > n \log q(\lambda) + \lambda$.
2. Challenger chooses $b \leftarrow \{0,1\}$ and $(\mathbf{A}, T_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$; $\mathbf{Z} \leftarrow \mathbb{Z}_q^{n \times k}$, $\mathbf{U}_0 \leftarrow \mathsf{SamplePre}(\mathbf{A}, T_{\mathbf{A}}, \sigma, \mathbf{Z})$ and $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times k}$. It sends $(\mathbf{A}, \mathbf{U}_b)$ to the adversary.
3. $\mathcal{A}$ outputs its guess b'. The experiment outputs 1 iff $b = b'$.

Figure 2: Experiment $\mathsf{Expt}_{\mathcal{T},\mathcal{A}}^{\mathrm{preimg},q,\sigma}$

These properties are satisfied by the gadget-based trapdoor lattice sampler of [MP12].

[13]Note that the pre-image sampling algorithm could be easily generalized to generate pre-images of matrices in $\mathbb{Z}_q^{n \times k}$ (for any k) by independently running $\mathsf{SamplePre}$ algorithm on each column of the matrix. Throughout this work, we overload the notation by directly giving matrices $\mathbf{U} \in \mathbb{Z}_q^{n \times k}$ as inputs to the $\mathsf{SamplePre}$ algorithm.

[14]In some cases, we can consider computationally unbounded adversaries if the inputs of the adversary are polynomially bounded.

2.2 Branching Programs

Branching programs are a model of computation used to capture space-bounded computations [BDFP86, Bar86]. In this work, we will be working with *leveled* branching programs.

Definition 2.4 (Leveled Branching Program)**.** A leveled branching program of length L, width w and input space $\{0,1\}^n$ consists of a sequence of $2L$ functions $\pi_{i,b} : [w] \to [w]$ for $1 \le i \le L, b \in \{0,1\}$, an input selection function $\mathsf{inp} : [L] \to [n]$, an accepting state $\mathsf{acc} \in [w]$ and a rejection state $\mathsf{rej} \in [w]$.The starting state st_0 is set to be 1 without loss of generality. The branching program evaluation on input $x \in \{0,1\}^n$ proceeds as follows:

- For $i = 1$ to L,
  - Let $\mathsf{pos} = \mathsf{inp}(i)$ and $b = x_{\mathsf{pos}}$. Compute $\mathsf{st}_i = \pi_{i,b}(\mathsf{st}_{i-1})$.
- If $\mathsf{st}_L = \mathsf{acc}$, output 1. If $\mathsf{st}_L = \mathsf{rej}$, output 0, else output $\perp$.

Additionally, we also define a notion of 'input-circling' (leveled) branching programs. In an input-circling branching program, the input bits are read sequentially in an ascending order (i.e., $1, \ldots, n, 1, \ldots$). Thus, the input-selector function inp is fixed. Additionally, each bit must be read the same number of times. Formally, we describe as follows.

Definition 2.5. A branching program $\mathsf{BP} = \left(\{\pi_{i,b} : [w] \to [w]\}_{i \in [L], b \in \{0,1\}}, \mathsf{acc} \in [w], \mathsf{rej} \in [w]\right)$ with input space $\{0,1\}^n$ is said to be a input-circling branching program if for all $i \le L$, $\mathsf{inp}(i) = ((i-1) \bmod n) + 1$, and $L \bmod n = 0$.

Any leveled branching program of length L and input space $\{0,1\}^n$ can be easily transformed to an input-circling branching program of length $n \cdot L$. In this work, we work with classes of branching programs that all share the same input selector function $\mathsf{inp}(\cdot)$ which is known during setup. The input selector as described above is just one possibility, and we stick with it for simplicity. Note that we do not require the transition functions $\pi_{i,b}$ to be permutations.

3 Traitor Tracing

In this section, we will first present the syntax and security definitions for traitor tracing schemes. Next, we will introduce the notion of private linear broadcast encryption (PLBE), and finally show that PLBE implies traitor tracing.

The notion of traitor tracing was introduced by Chor, Fiat and Naor [CFN94]. In a traitor tracing scheme for n parties, the setup algorithm chooses a master secret key, a public key and n secret keys for the users. Encryption can be performed using the public key, and each user can decrypt the ciphertext using his/her secret key. There is also a trace algorithm that, given black box access to a successful pirate decoding box, can catch the traitors who colluded to create the pirate decoding box. Traditional definitions of traitor tracing [CFN94, BSW06] required that the trace algorithm must catch a traitor if a pirate decoding box can decrypt an encryption of a random ciphertext. In this work, we will be using the *indistinguishability* based definition introduced by Goyal et al. [GKRW17], which is itself based on the definition introduced by Nishimaki, Wichs, and Zhandry [NWZ16]. In this definition, the trace algorithm must catch a traitor even if the pirate decoder box can only distinguish between encryptions of two adversarially chosen messages.

3.1 Public Key Traitor Tracing

A traitor tracing scheme $\mathcal{T}$ with message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ consists of four PPT algorithms $\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}$ and Trace with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^n) \to (\mathsf{msk}, \mathsf{pk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$. The setup algorithm takes as input the security parameter λ, number of users n, and outputs a master secret key msk, a public key pk and n secret keys $\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n$.

- $\mathsf{Enc}(\mathsf{pk}, m \in \mathcal{M}_\lambda) \to \mathsf{ct}$. The encryption algorithm takes as input a public key pk, message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to y$. The decryption algorithm takes as input a secret key sk, ciphertext ct and outputs $y \in \mathcal{M}_\lambda \cup \{\perp\}$.
- $\mathsf{Trace}^D(\mathsf{msk}, 1^y, m_0, m_1) \to T$. The trace algorithm has oracle access to a program D, it takes as input a master secret key msk, parameter y (in unary) and two messages m_0, m_1. It outputs a set $T \subset \{1, 2, \ldots, n\}$.

Correctness. Informally, correctness requirement states decrypting an encryption of message m using any one of the valid secret keys must output m. Formally, a traitor tracing scheme is said to be correct if there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$, and $i \in \{1, 2, \ldots, n\}$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{msk}, \mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

3.1.1 Security

There are two security requirements for a traitor tracing scheme. First, it is required that it satisfies IND-CPA security. Second, it is required that the tracing algorithm must (almost always) correctly trace at least one key used to create a pirate decoding box (whenever the pirate box successfully decrypts with noticeable probability) as well as it should not falsely accuse any user of cheating. The formal definitions are provided below.

Definition 3.1 (IND-CPA security)**.** A traitor tracing scheme $\mathcal{T} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ is IND-CPA secure if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}(\mathsf{ct}) = b \ : \ \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda); (\mathsf{msk}, \mathsf{pk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \\ b \leftarrow \{0,1\}; (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

Definition 3.2 (Ind-secure traitor tracing)**.** Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a traitor tracing scheme. For any non-negligible function $\epsilon(\cdot)$ and PPT adversary $\mathcal{A}$, consider the experiment $\mathsf{Expt\text{-}TT}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$ defined as follows.

Experiment $\mathsf{Expt\text{-}TT}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$

- $1^n \leftarrow \mathcal{A}(1^\lambda)$
- $(\mathsf{msk}, \mathsf{pk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$.
- $(D, m_0, m_1) \leftarrow \mathcal{A}^{O(\cdot)}(\mathsf{pk})$
- $T \leftarrow \mathsf{Trace}^D(\mathsf{msk}, 1^{1/\epsilon(\lambda)}, m_0, m_1)$.

Here, $O(\cdot)$ is an oracle that has $\{\mathsf{sk}_i\}_{i\in[n]}$ hardwired, takes as input an index $i \in [n]$ and outputs sk_i. Let S be the set of indices queried by $\mathcal{A}$.

Figure 3: Experiment $\mathsf{Expt\text{-}TT}$

Based on the above experiment, we now define the following (probabilistic) events and the corresponding probabilities (which are a functions of λ, parameterized by $\mathcal{A}, \epsilon$):

- $\mathsf{Good\text{-}Decoder}$: $\Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$
  $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Good\text{-}Decoder}]$.

- $\mathsf{Cor\text{-}Tr}$: $T \neq \emptyset \wedge T \subseteq S$
  $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$.
- $\mathsf{Fal\text{-}Tr}$: $T \not\subseteq S$
  $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Fal\text{-}Tr}]$.

A traitor tracing scheme $\mathcal{T}$ is said to be ind-secure if for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists negligible functions $\mathrm{negl}_1(\cdot)$, $\mathrm{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, the following holds

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \mathrm{negl}_1(\lambda), \quad \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}_2(\lambda).$$

3.2 Private Linear Broadcast Encryption

Next, we present the notion of private linear broadcast encryption (PLBE). PLBE was introduced by Boneh, Sahai and Waters [BSW06] as a framework for constructing traitor tracing schemes. There are four algorithms in a PLBE scheme — $\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec}$. The setup algorithm outputs a master secret key, public parameters and n secret keys, one for each user in the system. The public key encryption algorithm can be used to encrypt messages, and ciphertexts can be decrypted using one of the n secret keys via the decryption algorithm. In addition to these algorithms, there is also a special *trace-encryption* algorithm. This algorithm, which uses the master secret key, can be used to encrypt messages to any index $i \in \{0, 1, \ldots, n\}$. A secret key for user j can decrypt a ciphertext for index i only if $j > i$.

Boneh, Sahai and Waters [BSW06] proposed three security definitions for PLBE schemes. The first one requires that special-encryptions to index 0 must be indistinguishable from public key encryptions, even if the adversary has all the secret keys. The next security requirement is that special encryptions to index $i - 1$ must be indistinguishable from special encryptions to index i if the adversary does not have secret key for user i. Finally, the third security property is that special encryption of message m_0 to index n must be indistinguishable from special encryption of message m_1 to index n, even if the adversary has all secret keys. However, as discussed in Section 1.1, the BSW definitions of PLBE do not suffice for constructing traitor tracing schemes. Here, we first provide the PLBE syntax, and then present the decoder-based and query-based security definitions of PLBE.

Syntax. A PLBE scheme $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ has the following syntax.

- $\mathsf{Setup}(1^\lambda, 1^n) \rightarrow (\mathsf{msk}, \mathsf{pp}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$. The setup algorithm takes as input the security parameter λ, number of users n and outputs public parameters pp, master secret key msk and n secret keys $(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n)$.
- $\mathsf{Enc}(\mathsf{pp}, m) \rightarrow \mathsf{ct}$. The encryption algorithm takes as input public parameters pp, message $m \in \mathcal{M}_\lambda$, and outputs a ciphertext ct.
- $\mathsf{Enc\text{-}index}(\mathsf{msk}, m, i \in \{0, 1, 2, \ldots, n\}) \rightarrow \mathsf{ct}$. The index-encryption algorithm takes as input the master secret key msk, message $m \in \mathcal{M}_\lambda$, index $i \in \{0, 1, 2, \ldots, n\}$ and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \rightarrow y$. The decryption algorithm takes as input a secret key sk, ciphertext ct and outputs $y \in \mathcal{M}_\lambda \cup \{\perp\}$.

Correctness. A PLBE scheme is said to be correct if there exists a negligible functions $\mathrm{negl}_1(\cdot)$, $\mathrm{negl}_2(\cdot)$, $\mathrm{negl}_3(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$, and $i \in \{0, 1, \ldots, n\}$, $j \in \{1, 2, \ldots, n\}$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_j, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{msk}, \mathsf{pk}, \{\mathsf{sk}_k\}_{k\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \end{array}\right] \geq 1 - \mathrm{negl}_1(\lambda),$$

$$i < j \Rightarrow \Pr\left[\mathsf{Dec}(\mathsf{sk}_j, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{msk}, \mathsf{pk}, \{\mathsf{sk}_k\}_{k\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, i) \end{array}\right] \geq 1 - \mathrm{negl}_2(\lambda),$$

$$i \geq j \Rightarrow \Pr\left[\mathsf{Dec}(\mathsf{sk}_j, \mathsf{ct}) = \perp : \begin{array}{c} (\mathsf{msk}, \mathsf{pk}, \{\mathsf{sk}_k\}_{k\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, i) \end{array}\right] \geq 1 - \mathrm{negl}_3(\lambda).$$

3.2.1 q-Bounded PLBE Security

In this section we extend the existing PLBE security definitions by allowing the adversary to make a bounded number of index-encryption queries. Below we describe them in detail.

Definition 3.3 (q-bounded Normal Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A PLBE scheme is said to satisfy q-bounded normal hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$p_{\mathcal{A}}^{q,\mathsf{nrml}}(\lambda) = \Pr\left[\mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,0)}(\mathsf{ct}_b) = b \ : \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda);\ (\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ m \leftarrow \mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,0)}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[n]}\right) \\ b \leftarrow \{0,1\};\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m) \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where $\mathcal{A}$ can make at most $q(\lambda)$ queries to $\mathsf{Enc\text{-}index}(\mathsf{msk}, \cdot, 0)$ oracle. Note that here $\mathcal{A}$ is only allowed to query for ciphertexts corresponding to index 0.

Definition 3.4 (q-bounded Index Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A PLBE scheme is said to satisfy q-bounded index hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, every index $i^* \in \{0, \ldots, n-1\}$, the following holds:

$$p_{\mathcal{A}}^{q,\mathsf{ind}}(\lambda, i^*) = \Pr\left[\mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,\cdot)}(\mathsf{ct}) = b \ : \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda);\ (\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ m \leftarrow \mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,\cdot)}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in\{1,\ldots,n\}\setminus\{i^*+1\}}\right) \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, i^* + b) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where $\mathcal{A}$ can make at most $q(\lambda)$ queries to $\mathsf{Enc\text{-}index}(\mathsf{msk}, \cdot, \cdot)$ oracle. Note that here $\mathcal{A}$ can query the encryption oracle on arbitrary message-index pairs.

Definition 3.5 (q-bounded Message Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A PLBE scheme is said to satisfy q-bounded message hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$p_{\mathcal{A}}^{q,\mathsf{msg}}(\lambda) = \Pr\left[\mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,\cdot)}(\mathsf{ct}) = b \ : \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda);\ (\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,\cdot)}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[n]}\right) \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m_b, n) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where $\mathcal{A}$ can make at most $q(\lambda)$ queries to $\mathsf{Enc\text{-}index}(\mathsf{msk}, \cdot, \cdot)$ oracle. Note that here $\mathcal{A}$ can query the encryption oracle on arbitrary message-index pairs.

3.2.2 Decoder-based PLBE Security

In this section we introduce new decoder-based security definitions for PLBE schemes. We start by formally defining the notion of good distinguishers for PLBE schemes w.r.t. different encryption modes.

PLBE Distinguishers. For any $\gamma \in [-1/2, 1/2]$, PPT algorithm D, $\lambda, n \in \mathbb{N}$, $\mathsf{params} = (\mathsf{pp}, \mathsf{msk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$ and message $m \in \mathcal{M}_\lambda$, we say D is γ-$\mathsf{Dist}^{\mathsf{nrml},0}_{\mathsf{params}}$ for m if

$$\Pr\left[D(\mathsf{ct}_b) = b \ : \ \begin{array}{c} b \leftarrow \{0,1\};\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0); \end{array}\right] \geq \frac{1}{2} + \gamma,$$

where the probability is taken over the random coins used during encryption, the random coins of D and the choice of b.
Similarly, for any $i \in \{0, \ldots, n-1\}$ we can define D to be γ-$\mathsf{Dist}^{i,i+1}_{\mathsf{params}}$ for m if

$$\Pr\left[D(\mathsf{ct}_b) = b \ : \ \begin{array}{c} b \leftarrow \{0,1\} \\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, i+b) \end{array}\right] \geq \frac{1}{2} + \gamma.$$

Finally, we also define D to be γ-$\mathsf{Dist}^{n}_{\mathsf{params}}$ for messages m_0, m_1 if

$$\Pr\left[D(\mathsf{ct}_b) = b \ : \ \begin{array}{c} b \leftarrow \{0,1\} \\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m_b, n) \end{array}\right] \geq \frac{1}{2} + \gamma.$$

Definition 3.6 (Decoder-based Normal Hiding Security)**.** A PLBE scheme is said to satisfy decoder-based normal hiding security if for any PPT adversary $\mathcal{A}$, non-negligible function $\gamma(\cdot)$ and polynomial $q(\cdot)$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\gamma(\lambda) > 1/q(\lambda)$,

$$p^{\mathsf{dec,nrml}}_{\mathcal{A},\gamma,q}(\lambda) = \Pr\left[D \text{ is } \gamma(\lambda)\text{-}\mathsf{Dist}^{\mathsf{nrml},0}_{\mathsf{params}} \text{ for } m \ : \ \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda); \\ \mathsf{params} = \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i \in [n]}\right) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ (D, m) \leftarrow \mathcal{A}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i \in [n]}\right) \end{array}\right] \leq \mathrm{negl}(\lambda).$$

Definition 3.7 (Decoder-based Index Hiding Security)**.** A PLBE scheme is said to satisfy decoder-based index hiding security if for any PPT adversary $\mathcal{A}$, non-negligible function $\gamma(\cdot)$ and polynomial $q(\cdot)$ there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\gamma(\lambda) > 1/q(\lambda)$ and all $i^* \in \{0, \ldots, n-1\}$,

$$p^{\mathsf{dec,ind}}_{\mathcal{A},\gamma,q}(\lambda, i^*) = \Pr\left[D \text{ is } \gamma(\lambda)\text{-}\mathsf{Dist}^{i^*,i^*+1}_{\mathsf{params}} \text{ for } m \ : \ \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda); \\ \mathsf{params} = \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i \in [n]}\right) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ (D, m) \leftarrow \mathcal{A}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i \in \{1,\ldots,n\} \setminus \{i^*+1\}}\right) \end{array}\right] \leq \mathrm{negl}(\lambda).$$

Definition 3.8 (Decoder-based Message Hiding Security)**.** A PLBE scheme is said to satisfy decoder-based message hiding security if for any PPT adversary $\mathcal{A}$, non-negligible function $\gamma(\cdot)$ and polynomial $q(\cdot)$ there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\gamma(\lambda) > 1/q(\lambda)$,

$$p^{\mathsf{dec,msg}}_{\mathcal{A},\gamma,q}(\lambda) = \Pr\left[D \text{ is } \gamma(\lambda)\text{-}\mathsf{Dist}^{n}_{\mathsf{params}} \text{ for } m_0, m_1 \ : \ \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda); \\ \mathsf{params} = \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i \in [n]}\right) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ (D, m_0, m_1) \leftarrow \mathcal{A}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i \in [n]}\right) \end{array}\right] \leq \mathrm{negl}(\lambda).$$

4 Traitor Tracing from 1-bounded secure PLBE

In this section, we show how to construct traitor tracing schemes from PLBE schemes that achieve 1-bounded security. Our construction is divided in two components. First, we first show that PLBE schemes that achieve 1-bounded security also satisfy decoder-based security. Later, we show how to construct a traitor tracing scheme from PLBE schemes that achieves decoder-based security.

4.1 Decoder-based PLBE from 1-bounded secure PLBE

Let $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ be a PLBE scheme that satisfies 1-bounded security. We will show that the same scheme also satisfies decoder-based security.

Lemma 4.1. If PLBE satisfies 1-bounded normal hiding security (Definition 3.3), then it also satisfies decoder-based normal hiding security (Definition 3.6).

Proof. Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$, non-negligible function $\gamma(\cdot)$, polynomials $q(\cdot), r(\cdot)$ and an infinite sequence of security parameters $\Lambda = \{\lambda_i\}_{i\in\mathbb{N}}$ such that for all $\lambda \in \Lambda$, $\gamma(\lambda) > 1/q(\lambda)$ and $p^{\mathsf{dec,nrml}}_{\mathcal{A},\gamma,q}(\lambda) \geq 1/r(\lambda)$. We will use $\mathcal{A}$ that plays the 1-bounded normal hiding security game to build a PPT reduction algorithm $\mathcal{B}$ that plays the decoder-based normal hiding security game as follows.

For any $\lambda \in \mathbb{N}$, the reduction algorithm first receives 1^n from $\mathcal{A}$, which it forwards to the challenger. It then receives pp and n secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ from the challenger, which it forwards to $\mathcal{A}$. The adversary $\mathcal{A}$ outputs D and m. The reduction algorithm then queries the challenger for an encryption of m for index 0 (recall that the reduction algorithm is allowed one query). Let the challenger's response be ct_1. It then computes $\mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m)$. Next, it sends challenge message m, and receives ct^*, which is either a normal encryption of m, or an encryption of m for index 0. The reduction algorithm chooses a random bit $\beta \leftarrow \{0,1\}$, checks if $D(\mathsf{ct}_\beta) = D(\mathsf{ct}^*)$. If so, it outputs $b' = \beta$, else it outputs $b' = 1-\beta$ as its guess.

Let us now analyse $\mathcal{B}'s$ advantage. We need to show that there exists polynomials $q_{\mathcal{B}}(\cdot)$ and an infinite sequence $\Lambda_{\mathcal{B}} = \{\lambda_i\}_i$ such that for all $\lambda \in \Lambda_{\mathcal{B}}$, $p^{1,\mathsf{nrml}}_{\mathcal{B}} \geq 1/2 + 1/q_{\mathcal{B}}(\lambda)$. Let $\Lambda_B = \Lambda$ and $q_{\mathcal{B}}(\cdot) = q^2(\cdot)\cdot r(\cdot)/2$. Fix any $\lambda \in \Lambda$, and let $\gamma = \gamma(\lambda)$, $q = q(\lambda)$, $r = r(\lambda)$. Let b denote the 1-bounded challenger's choice (recall the challenger chooses $b \leftarrow \{0,1\}$, if $b = 0$, it sends a normal encryption, else it sends an encryption to index 0). First, let us fix $\mathsf{params} = (\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\leq n}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$ and $(D, m) \leftarrow \mathcal{A}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\leq n}\right)$ such that

$$\Pr\left[D(\mathsf{ct}_b) = b \;:\; \begin{array}{c} b \leftarrow \{0,1\};\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0); \end{array}\right] = \frac{1}{2} + \alpha_{\mathsf{params},D,m},$$

for some $\alpha_{\mathsf{params},D,m} \in [-1/2, 1/2]$. Next, consider the following probability:

$$\rho_{\mathsf{params},D,m} = \Pr\left[b = b' \;:\; \begin{array}{c} b \leftarrow \{0,1\}; \mathsf{ct}^*_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \mathsf{ct}^*_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0); \\ \beta \leftarrow \{0,1\}; \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0); \\ b' = \beta \text{ if } D(\mathsf{ct}^*_b) = D(\mathsf{ct}_\beta),\ \text{else } b' = 1-\beta \end{array}\right].$$

Since the decoder D is run on ciphertexts $\mathsf{ct}^*_b, \mathsf{ct}_\beta$ independently, we could rewrite the above probability as follows:

$$\begin{aligned}
\rho_{\mathsf{params},D,m} =& \Pr\left[D(\mathsf{ct}^*_b) = b : \begin{array}{l} b \leftarrow \{0,1\}; \mathsf{ct}^*_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}^*_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right] \cdot \Pr\left[D(\mathsf{ct}_\beta) = \beta : \begin{array}{l} \beta \leftarrow \{0,1\}; \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right] \\
&+ \Pr\left[D(\mathsf{ct}^*_b) \neq b : \begin{array}{l} b \leftarrow \{0,1\}; \mathsf{ct}^*_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}^*_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right] \cdot \Pr\left[D(\mathsf{ct}_\beta) \neq \beta : \begin{array}{l} \beta \leftarrow \{0,1\}; \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right] \\
=& \Pr\left[D(\mathsf{ct}_b) = b : \begin{array}{l} b \leftarrow \{0,1\}; \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right]^2 \\
&+ \Pr\left[D(\mathsf{ct}_b) \neq b : \begin{array}{l} b \leftarrow \{0,1\}; \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right]^2 \\
=& \left(\frac{1}{2} + \alpha_{\mathsf{params},D,m}\right)^2 + \left(\frac{1}{2} - \alpha_{\mathsf{params},D,m}\right)^2 \\
=& \frac{1}{2} + 2\cdot\alpha^2_{\mathsf{params},D,m}
\end{aligned}$$

Thus, we get that for any decoder D that is δ-$\mathsf{Dist}^{\mathsf{nrml},0}_{\mathsf{params}}$ for message m,

$$\rho_{\mathsf{params},D,m} = 1/2 + 2\cdot\alpha^2_{\mathsf{params},D,m} \geq 1/2 + 2\cdot\delta^2.$$

Also, since $\alpha^2_{\mathsf{params},D,m} \geq 0$, we get that for every decoder D, $\rho_{\mathsf{params},D,m} \geq 1/2$. Therefore, since adversary $\mathcal{A}$ outputs a $1/q$-$\mathsf{Dist}^{\mathsf{nrml},0}_{\mathsf{params}}$ box with probability at least $1/r$, we get that the reduction algorithm $\mathcal{B}$'s winning probability $p^{1,\mathsf{nrml}}_{\mathcal{B},n}$ (as defined in Definition 3.3) is

$$\begin{aligned} p^{1,\mathsf{nrml}}_{\mathcal{B},n} &\geq \frac{1}{r} \cdot \left(\frac{1}{2} + \frac{2}{q^2}\right) + \left(1 - \frac{1}{r}\right) \cdot \left(\frac{1}{2}\right). \\ &\geq \frac{1}{2} + \frac{2}{r \cdot q^2}. \end{aligned}$$

This concludes our proof.

■

Lemma 4.2. If PLBE satisfies 1-bounded index hiding security (Definition 3.4), then it also satisfies decoder-based index hiding security (Definition 3.7).

The proof of this lemma is identical to the proof of Lemma 4.1, except that the reduction algorithm queries for either a special encryption of m for index i or a special encryption of m for index $i+1$ (depending on $\beta \leftarrow \{0,1\}$).

Lemma 4.3. If PLBE satisfies 1-bounded message hiding security (Definition 3.5), then it also satisfies decoder-based message hiding security (Definition 3.8).

The proof of this lemma is identical to the proof of Lemma 4.1, except that the reduction algorithm queries for either a special encryption of m_0 for index n or a special encryption of m_1 for index n (depending on $\beta \leftarrow \{0,1\}$).

4.2 Traitor Tracing from Decoder-based PLBE

Let $\mathsf{PLBE} = (\mathsf{PLBE.Setup}, \mathsf{PLBE.Enc}, \mathsf{PLBE.Enc\text{-}index}, \mathsf{PLBE.Dec})$ be a PLBE scheme with decoder-based security. We will use PLBE to construct a traitor tracing scheme $\mathcal{T} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ as follows. The construction is identical to the transformation in [BSW06], however the security proof provided in [BSW06] was not correct. Concretely, to argue correctness of tracing they incorrectly leveraged the indistinguishability-based security of underlying PLBE scheme. We show that the same transformation could be proven to satisfy correct tracing if one starts with a PLBE scheme that achieves decoder-based security.

$\mathsf{Setup}(1^\lambda, 1^n)$: The setup algorithm computes $(\mathsf{pp}, \mathsf{msk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)) \leftarrow \mathsf{PLBE.Setup}(1^\lambda, 1^n)$. The public parameters are pp, master secret key is msk and the n secret keys are $\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}$.

$\mathsf{Enc}(\mathsf{pp}, m)$: The encryption algorithm outputs $\mathsf{ct} \leftarrow \mathsf{PLBE.Enc}(\mathsf{pp}, m)$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: The decryption algorithm outputs $\mathsf{PLBE.Dec}(\mathsf{sk}, \mathsf{ct})$.

$\mathsf{Trace}^D(\mathsf{msk}, 1^y, m_0, m_1)$: Let $\epsilon = 1/y$ and $W = \lambda \cdot (n \cdot y)^2$. For $i = 0$ to n, the trace algorithm does the following:

1. It first sets $\mathsf{count}_i = 0$. For $j = 1$ to W, it does the following:
   (a) It chooses $b_{i,j} \leftarrow \{0,1\}$, sets $\mathsf{ct}_{i,j} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m_b, i)$. If $D(\mathsf{ct}_{i,j}) = b_{i,j}$, it sets $\mathsf{count}_i = \mathsf{count}_i + 1$.
2. It sets $\hat{p}_i = \mathsf{count}_i / W$.

The trace algorithm outputs every index $i \in \{1, 2, \ldots, n\}$ such that $\hat{p}_{i-1} - \hat{p}_i \geq \epsilon/4n$.

Correctness. This follows directly from the first correctness property of PLBE scheme.

4.2.1 IND-CPA Security

We would like to point out that the scheme PLBE is IND-CPA secure even if it only satisfies 0-bounded security. In other words, we do not need the scheme to achieve stronger decoder-based security. Thus, the proof of IND-CPA security is identical to that provided in [BSW06]. Below we provide a high level sketch.

Theorem 4.1. Assuming the PLBE scheme $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ satisfies the security properties in Definition 3.6, Definition 3.7 and Definition 3.8, the traitor tracing scheme described above is IND-CPA secure (Definition 3.1).

Proof. We will construct a sequence of $2n + 2$ hybrid experiments to prove INDCPA security. The first experiment, that is Hybrid H_0, is exactly the IND-CPA game.

Hybrid H_0 : In this experiment, the challenger sends public parameters pp, receives m_0, m_1 from $\mathcal{A}$ and sends $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m_0)$ to $\mathcal{A}$.

Hybrid $H_{i,b}$ (for $i \le n, b \in \{0,1\}$) : This experiment is identical to the INDCPA experiment, except that the adversary, after sending challenge messages m_0, m_1, receives $\mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, i, m_b)$.

Hybrid H_1 : In this experiment, the challenger sends public parameters pp, receives m_0, m_1 from $\mathcal{A}$ and sends $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m_1)$ to $\mathcal{A}$.

For any PPT adversary $\mathcal{A}$, let $p_{\mathcal{A},x}(\cdot)$ be a function of λ that denotes the probability of $\mathcal{A}$ outputting 0 in H_x. Note that $p_{\mathcal{A},0} - p_{\mathcal{A},1}$ is the advantage of $\mathcal{A}$ in the INDCPA security game.

Claim 4.1. Assuming PLBE satisfies Definition 3.6, for any PPT adversary $\mathcal{A}$, there exists a negligible function such that for all $\lambda \in \mathbb{N}$ and $b \in \{0,1\}$, $|p_{\mathcal{A},b} - p_{\mathcal{A},0,b}| \le \mathrm{negl}(\lambda)$.

This follows from decoder-based indistinguishability of normal and 0-index encryptions (Definition 3.6) of PLBE.

Claim 4.2. Assuming PLBE satisfies Definition 3.7, for any PPT adversary $\mathcal{A}$, there exists a negligible function such that for all $\lambda \in \mathbb{N}$, $b \in \{0,1\}$ and $i \in \{1, 2, \dots, n\}$, $p_{\mathcal{A},i-1,b} - p_{\mathcal{A},i,b} \le \mathrm{negl}(\lambda)$.

This follows from the decoder-based index hiding security notion (Definition 3.7) of PLBE.

Claim 4.3. Assuming PLBE satisfies Definition 3.8, for any PPT adversary $\mathcal{A}$, there exists a negligible function such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},n,0} - p_{\mathcal{A},n,1} \le \mathrm{negl}(\lambda)$.

This follows from the decoder-based message hiding security notion (Definition 3.8) of PLBE.
From the above claims, it follows that $p_{\mathcal{A},0} - p_{\mathcal{A},1}$ is bounded by a negligible function.

■

4.2.2 Correctness of Tracing

Next, we will show that the false trace probability is bounded by a negligible function, and the correct trace probability is close to the probability of $\mathcal{A}$ outputting an ϵ-successful decoding box.

First, we will introduce some notations. Given any pirate decoder box D and messages m_0, m_1, for any $i \in \{0, 1, \dots, n\}$, let

$$p_i^D = \Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, i, m_b)]$$

where the probability is taken over random coins of decoder D as well as the randomness used during encryption. Similarly, let $p_{\mathsf{nrml}}^D = \Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, m_b)]$.

False Trace Probability. First, we show that the tracing algorithm never falsely accuses any user. Formally, we prove the following.

Theorem 4.2. For every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$,

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \text{negl}(\lambda),$$

where $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\cdot)$ is as defined in Definition 3.2.

Proof. We will skip the dependence of $\epsilon(\cdot)$ on λ for simplicity of notation. Let S be the set of keys queried and D the decoder output by $\mathcal{A}$. For $i \in \{1, 2, \ldots, n\}$, we define events $\mathsf{Diff\text{-}Adv}_i^D \; : \; p_{i-1}^D - p_i^D > \epsilon/8n$, and $\mathsf{Diff\text{-}Adv}^D \; : \; \bigvee_{k \in \{1,\ldots,n\} \setminus S} \mathsf{Diff\text{-}Adv}_k^D$.

First, note that the probability of the event *false trace* can be rewritten as follows by conditioning on the events defined above

$$\Pr[\mathsf{Fal\text{-}Tr}] \leq \Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}^D}] + \sum_{i \in \{1,\ldots,n\}} \Pr[i \notin S \wedge \mathsf{Diff\text{-}Adv}_i^D].$$

We will show that each of these terms is bounded by a negligible function.

Lemma 4.4. For every PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_1(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}^D}] \leq \text{negl}_1(\lambda).$$

Proof. The proof of this lemma follows from Chernoff bounds. Let n be the number of users chosen by the adversary $\mathcal{A}$. Fix any $i \in \{1, \ldots, n\} \setminus S$ and decoding box D. Let us consider the probability that the output of Trace algorithm includes i, given that $\mathsf{Diff\text{-}Adv}_i^D$ does not occur. Note that the tracing algorithm includes i in the traitor set if the estimates $\hat{p}_{i-1}$ and $\hat{p}_i$ differ by at least $\epsilon/4n$.

Let $X_{k,j}$ denote the random variable that is 1 if $D(\mathsf{ct}_{k,j}) = b_{k,j}$ for $k \in \{i-1, i\}$ and $j \in \{1, 2, \ldots, W\}$ (here the randomness is over the choice of $b_{k,j}$ and the randomness used by $\mathsf{Enc\text{-}index}$ and D) and $Z_{i,j} = X_{i-1,j} - X_{i,j}$. Then $(\sum_{j=1}^{W} X_{k,j})/W = \hat{p}_k$ and $\mu_i = \mathbb{E}[Z_{i,j}] = p_{i-1}^D - p_i^D$.

Since the Z_is are independent samples, using Chernoff bounds, we get that $\Pr[\sum_j Z_j/W > 2\mu_i] \leq 2^{-O(\lambda)}$. Using this we can write that for every $i \in \{1, \ldots, n\} \setminus S$, $\Pr[\mathsf{Fal\text{-}Tr} \wedge i \in T \mid \overline{\mathsf{Diff\text{-}Adv}^D}] \leq 2^{-O(\lambda)}$, where T denotes the set of indices output by Trace. Finally, using a union bound, we get that

$$\Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}^D}] \leq n \cdot 2^{-O(\lambda)} = \text{negl}_1(\lambda).$$

■

Lemma 4.5. Assuming PLBE is a secure PLBE scheme satisfying the decoder based index hiding security property (Definition 3.7), for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists a negligible function $\text{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$ and $i \in \{1, 2, \ldots, n\}$,

$$\Pr[i \notin S \wedge \mathsf{Diff\text{-}Adv}_i^D] \leq \text{negl}_2(\lambda),$$

where n is the number of users, S is the set of key queries, and D is the decoder box sent by $\mathcal{A}$.

Proof. Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible functions $\epsilon(\cdot), \delta(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, there exists an $i^* \in \{1, 2, \ldots, n\}$ s.t. $\Pr[i^* \notin S \wedge \mathsf{Diff\text{-}Adv}_{i^*}^D] \geq \delta(\lambda)$. Then we can use $\mathcal{A}$ to build a PPT reduction algorithm $\mathcal{B}$ that breaks the index hiding security property of PLBE.

The reduction algorithm $\mathcal{B}$ first receives 1^n from the adversary, which it forwards to the challenger. It then receives the PLBE public parameters pp from the challenger, which it sends to $\mathcal{A}$. Next, it chooses an index

$i \leftarrow \{0, 1, \ldots, n\}$ and sends it to the PLBE challenger.[15] It receives secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{i-1}, \mathsf{sk}_{i+1}, \ldots, \mathsf{sk}_n$. The adversary $\mathcal{A}$ queries for secret keys. If $\mathcal{A}$ queries for i, $\mathcal{B}$ sends an empty decoding box to the PLBE challenger. Else, on receiving query $j \neq i$ from $\mathcal{A}$, it sends sk_j to $\mathcal{A}$. After all queries, the adversary sends a decoding box D and messages m_0, m_1 to $\mathcal{B}$. The reduction algorithm chooses a uniformly random bit $b' \leftarrow \{0, 1\}$ and sends $D, m_{b'}$ to the PLBE challenger.

Let $p_{j,b}^D = \Pr[D(\mathsf{ct}) = b \ : \ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, j, m_b)]$, where the probability is taken over the coins of decoder D and encryption algorithm. Recall we have that $\Pr[i^* \notin S \wedge \mathsf{Diff\text{-}Adv}_{i^*}] \geq \delta(\lambda)$. Therefore, we can write that

$$\Pr\left[i^* \notin S \wedge \left((p_{i^*-1,0}^D + p_{i^*-1,1}^D)/2 - (p_{i^*,0}^D + p_{i^*,1}^D)/2\right) \geq \epsilon/8n\right] \geq \delta(\lambda)$$
$$\Rightarrow \Pr\left[i = i^* \wedge i^* \notin S \wedge \left((p_{i-1,0}^D + p_{i-1,1}^D)/2 - (p_{i,0}^D + p_{i,1}^D)/2\right) \geq \epsilon/8n\right] \geq \delta(\lambda)/n.$$

Thus, we can also write that there exists a bit b such that

$$\Pr\left[i = i^* \wedge i^* \notin S \wedge \left(p_{i-1,b}^D - p_{i,b}^D\right) \geq \epsilon/8n\right] \geq \delta(\lambda)/n.$$

Now since the reduction algorithm $\mathcal{B}$ simply randomly guesses this bit b, thus we have that with probability at least $\delta/2n$, $\mathcal{B}$ outputs a decoding box D and a message m_b such that D can distinguish between encryptions of m_b to indices $i-1$ and i with advantage at least $\epsilon/8n$. Thus, the lemma follows. ∎

From the above lemmas, it follows that the probability of false trace is at most $\mathrm{negl}_1(\lambda) + n \cdot \mathrm{negl}_2(\lambda)$, thus theorem follows. ∎

Correct Trace Probability. Now we show that whenever the adversary outputs a good decoder, then with all but negligible probability the tracing algorithm outputs a non-empty set T. Combining this with Theorem 4.2, we get that the tracing algorithm correctly traces. Formally, we show the following.

Theorem 4.3. For every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$,

$$\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}(\lambda)$$

where $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\cdot)$ and $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\cdot)$ are as defined in Definition 3.2.

Proof. Let us start by analyzing the probability that tracing algorithm outputs a non-empty set T. First, we know that if event $\mathsf{Good\text{-}Decoder}$ occurs, then $p_{\mathsf{nrml}}^D \geq 1/2 + \epsilon$. Next, let S be the set of indices $i \in \{1, \ldots, n\}$ such that $p_{i-1}^D - p_i^D > \epsilon/2n$. Using Chernoff bounds, we get that

$$\forall\ i \in S, \quad \Pr\left[\hat{p}_{i-1}^D - \hat{p}_i^D < \epsilon/4n\right] \leq 2^{-O(\lambda)} = \mathrm{negl}_1(\lambda). \tag{1}$$

Note that by message hiding security of the underlying PLBE scheme, we have that $p_n^D \leq 1/2 + \mathrm{negl}_2(\lambda)$ for some negligible function $\mathrm{negl}_2(\cdot)$. Also, by indistinguishability of *normal* and index 0 ciphertexts, we have that $p_{\mathsf{nrml}}^D - p_0^D \leq \mathrm{negl}_3(\lambda)$ for some negligible function $\mathrm{negl}_3(\cdot)$. Thus, $p_0^D - p_n^D \geq \epsilon - \mathrm{negl}_2(\lambda) - \mathrm{negl}_3(\lambda) > \epsilon/2$. Given this we can conclude that the set S as defined above (i.e., for $i \in S$, $p_{i-1}^D - p_i^D > \epsilon/2n$) must be non-empty whenever event $\mathsf{Good\text{-}Decoder}$ occurs. Combining this with Equation (1), we get that

$$\Pr[T \neq \emptyset] \geq (1 - \mathrm{negl}_1(\lambda)) \cdot \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}(\lambda).$$

Finally, combining with Theorem 4.2, we get that

$$\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}(\lambda).$$

This concludes the proof. ∎

[15] In other words, the reduction algorithm randomly guesses the index hiding challenger with which it interacts.

5 Mixed Functional Encryption

A functional encryption scheme consists of a setup, encryption, key generation, and decryption algorithm. The setup algorithm takes the security parameter and functionality index as inputs, and outputs public parameters and a master secret key. The encryption algorithm uses the public parameters to encrypt a message, while the key generation algorithm uses the master secret key to compute a secret key corresponding to a function. The decryption algorithm takes as input a ciphertext and a secret key, and outputs the function evaluation on the message.

In this work, we introduce the notion of mixed functional encryption (Mixed FE). A Mixed FE scheme is defined as a dual of the standard FE (i.e., ciphertext-policy) in which the secrets keys are associated with a message, and ciphertexts are associated with (boolean) functions. Additionally, in a Mixed FE system, there are two encryption algorithms — Enc and $\mathsf{SK\text{-}Enc}$. The normal encryption algorithm Enc takes as input only the public parameters and outputs a encryption of a "canonical" *always-accepting* function. The "secret key" encryption algorithm, on the other hand, takes as input the master secret key and a function f, and encrypts f. The decryption algorithm in a Mixed FE system works similar to that in standard FE, that is it outputs the evaluation of encrypted function f on the message m associated with the secret key. Below we provide a formal definition.

Consider function classes $\mathcal{F} = \{\mathcal{F}_\kappa\}_\kappa$ and message spaces $\mathcal{M} = \{\mathcal{M}_\kappa\}_\kappa$, where $f : \mathcal{M}_\kappa \to \{0,1\}$ for each $f \in \mathcal{F}_\kappa$.[16] A mixed functional encryption scheme $\mathsf{Mixed\text{-}FE}$, for function classes $\mathcal{F}$ and message spaces $\mathcal{M}$, consists of five polytime algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{SK\text{-}Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{pp}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ and functionality index κ, and outputs the public parameters pp and the master secret key msk.
- $\mathsf{Enc}(\mathsf{pp}) \to \mathsf{ct}$. The normal encryption algorithm takes as input public parameters pp, and outputs a ciphertext ct.
- $\mathsf{SK\text{-}Enc}(\mathsf{msk}, f) \to \mathsf{ct}$. The secret key encryption algorithm takes as input master secret key msk and a function $f \in \mathcal{F}_\kappa$. It outputs a ciphertext ct.
- $\mathsf{KeyGen}(\mathsf{msk}, m) \to \mathsf{sk}_m$. The key generation algorithm takes as input master secret key msk and a input/message $m \in \mathcal{M}_\kappa$. It outputs a secret key sk_m.
- $\mathsf{Dec}(\mathsf{sk}_m, \mathsf{ct}) \to \{0,1\}$. The decryption algorithm takes as input a secret key sk_m and a ciphertext ct, and it outputs a single bit.

Correctness. A mixed functional encryption scheme is said to be correct if there exists negligible functions $\mathrm{negl}_1(\cdot), \mathrm{negl}_2(\cdot)$ such that for all $\lambda, \kappa \in \mathbb{N}$, for every $f \in \mathcal{F}_\kappa$, $m \in \mathcal{M}_\kappa$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_m, \mathsf{ct}) = 1 : \begin{array}{c} (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa); \\ \mathsf{sk}_m \leftarrow \mathsf{KeyGen}(\mathsf{msk}, m); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}) \end{array}\right] \geq 1 - \mathrm{negl}_1(\lambda),$$

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_m, \mathsf{ct}) = f(m) : \begin{array}{c} (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa); \\ \mathsf{sk}_m \leftarrow \mathsf{KeyGen}(\mathsf{msk}, m); \mathsf{ct} \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f) \end{array}\right] \geq 1 - \mathrm{negl}_2(\lambda).$$

Security. Informally, for security we require that no PPT adversary should be able to distinguish between secret key encryptions of two functions f_0 and f_1 if for every key in its possession, the output of f_0, f_1 is identical. Additionally, we also require that it should be hard to distinguish between normal encryptions and secret key encryptions of the special always-accepting function. In this work, we are only interested in mixed FE schemes that guarantees security against adversaries which make a bounded number of secret key encryption queries. Below we formally define it.

[16]The following definition could be easily generalized for multi-bit function classes, but for simplicity we stick to boolean functions.

Definition 5.1 (q-bounded Function Indistinguishability)**.** Let $q(\cdot)$ be any fixed polynomial. A mixed functional encryption scheme $\mathsf{Mixed\text{-}FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{SK\text{-}Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to satisfy q-bounded function indistinguishability security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b : \begin{array}{c} 1^{\kappa} \leftarrow \mathcal{A}(1^{\lambda}); (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{\kappa}) \\ (f^{(0)}, f^{(1)}) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{pp}) \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f^{(b)}) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where

- $\mathcal{A}$ can make at most $q(\lambda)$ queries to $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \cdot)$ oracle, and
- every secret key query m made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle must satisfy the condition that $f^{(0)}(m) = f^{(1)}(m)$.

We also define a restricted version of the function indistinguishability game in which the adversary must declare its challenge functions $(f^{(0)}, f^{(1)})$ at the beginning, and it must make all its q encryption queries before any of its key generation queries.

Definition 5.2 (q-bounded Restricted Function Indistinguishability)**.** Let $q(\cdot)$ be any fixed polynomial. A mixed functional encryption scheme $\mathsf{Mixed\text{-}FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{SK\text{-}Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to satisfy q-bounded selective function indistinguishability security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{pp}, \mathsf{ct}) = b : \begin{array}{c} (1^{\kappa}, f^{(0)}, f^{(1)}) \leftarrow \mathcal{A}(1^{\lambda}); \\ (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{\kappa}) \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f^{(b)}) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where

- $\mathcal{A}$ can make at most $q(\lambda)$ queries to $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \cdot)$ oracle, and
- every secret key query m made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle must satisfy the condition that $f^{(0)}(m) = f^{(1)}(m)$, and
- $\mathcal{A}$ must make all (at most $q(\lambda)$) $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \cdot)$ oracle queries before making any query to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle.

Definition 5.3 (q-bounded Accept Indistinguishability)**.** Let $q(\cdot)$ be any fixed polynomial. A mixed functional encryption scheme $\mathsf{Mixed\text{-}FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{SK\text{-}Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to satisfy q-bounded accept indistinguishability security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{ct}_b) = b : \begin{array}{c} 1^{\kappa} \leftarrow \mathcal{A}(1^{\lambda}); (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{\kappa}) \\ f^* \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{pp}) \\ b \leftarrow \{0,1\};\ \mathsf{ct}_1 \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f^*) \\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where

- $\mathcal{A}$ can make at most q queries to $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \cdot)$ oracle,
- every secret key query m made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle must satisfy the condition that $f^*(m) = 1$.

Additionally, we also define a restricted notion of the accept indistinguishability property for mixed functional encryption schemes, in which the adversary must declare its challenge function f^* at the beginning, and it must make all its q encryption queries before any of its key generation queries, and it is restricted to only make ciphertext queries for functions f such that all queried f evaluate to 1 on all (secret key) queried messages m. Below we formally describe it.

Definition 5.4 (q-bounded Restricted Accept Indistinguishability)**.** Let $q(\cdot)$ be any fixed polynomial. A mixed functional encryption scheme $\mathsf{Mixed\text{-}FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{SK\text{-}Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to satisfy q-bounded *restricted* accept indistinguishability security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{pp}, \mathsf{ct}_b) = b : \begin{array}{c} (1^\kappa, f^*) \leftarrow \mathcal{A}(1^\lambda); (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa) \\ b \leftarrow \{0,1\};\ \mathsf{ct}_1 \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f^*) \\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}) \end{array}\right] \leq \frac{1}{2} + \text{negl}(\lambda)$$

where

- $\mathcal{A}$ can make at most $q(\lambda)$ queries to $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \cdot)$ oracle,
- every secret key query m made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle must satisfy the condition that $f^*(m) = 1$ as well as $f(m) = 1$ for every ciphertext query f made by $\mathcal{A}$ to the $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \cdot)$ oracle, and
- $\mathcal{A}$ must make all (at most $q(\lambda)$) $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \cdot)$ oracle queries before making any query to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle.

6 Construction of PLBE from Mixed FE and ABE

In this section, we construct a private linear broadcast encryption (PLBE) scheme from any key-policy attribute based encryption (KP-ABE) scheme and a mixed functional encryption (Mixed FE) scheme. Our construction inherits the message space of the underlying KP-ABE scheme. Also, we show that if the underlying ABE scheme is selectively-secure, and mixed FE scheme satisfies 1-bounded restricted function and accept indistinguishability properties, then our PLBE scheme satisfies 1-bounded normal, index, and message hiding security properties.

Outline. The idea is to use the ABE system to encrypt a message with attributes being set to be either the "normal" ciphertext (i.e., encryption of the canonical always-accepting function), or a special (secret key) ciphertext which encrypts the comparison function depending on the type of PLBE encryption operation being performed. Now each user's secret key will be an ABE private key for the policy circuit being the mixed FE decryption circuit with a mixed FE secret key corresponding to user's index hardwired. The high level intuition is that when the attribute is a normal FE ciphertext then all keys decrypt it to 1, thus any user with an appropriate ABE key could perform the decryption. And, when the attribute is set to be a special ciphertext (that encrypts comparison with some index i), then only those users whose indices are larger than the threshold i set can perform the decryption. For proving security, we rely on the fact that special ciphertexts are indistinguishable to any adversary that does not have distinguishing secret keys. Below we provide a detailed overview.

The PLBE setup algorithm starts by sampling an ABE key pair $(\mathsf{abe.pp}, \mathsf{abe.msk})$ and a mixed FE key pair $(\mathsf{mixed.pp}, \mathsf{mixed.msk})$. To generate the private key for i^{th} user, it first generates a mixed FE secret key $\mathsf{mixed.sk}_i$ for message i, and later computes an ABE key $\mathsf{abe.sk}_i$ for predicate $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$, i.e. $\mathsf{Mixed\text{-}FE}$ decryption circuit with key $\mathsf{mixed.sk}_i$ hardwired. For PLBE normal encryption, one simply computes ciphertext ct as an encryption of message m under attributes $\mathsf{ct}_{\mathsf{attr}}$, where $\mathsf{ct}_{\mathsf{attr}}$ is a mixed FE normal-ciphertext. For encrypting a message to index i, the encryption algorithm works identically except now the attribute is set to be a special ciphertext corresponding to function *greater than i*. Finally, the PLBE decryption is simply the ABE decryption algorithm.

Now correctness follows directly from the correctness of ABE and FE schemes. For proving security, the main idea is that suppose there exists an adversary that can distinguish between PLBE normal encryptions and index 0 encryptions, then it can be used to distinguish between mixed FE normal ciphertexts and ciphertexts encrypting function *greater than* 0 (note that this is an always-accepting function). In other words, such an attack can be used to break restricted accept indistinguishability property of mixed FE scheme. Similarly, we can also reduce a successful attack on the index hiding, or message hiding security to an attacker on restricted function indistinguishability of mixed FE, or ABE security, respectively. Below we describe our construction $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ for messages spaces $\{\mathcal{M}_\kappa\}_\kappa$ in detail.

6.1 Construction

Let $\mathcal{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.Enc}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Dec})$ be a key-policy attribute based encryption scheme for set of attribute spaces $\{\mathcal{X}_\kappa\}_\kappa$, predicate classes $\{\mathcal{C}_\kappa\}_\kappa$ and message spaces $\{\mathcal{M}_\kappa\}_\kappa$, and $\mathsf{Mixed\text{-}FE} = (\mathsf{Mixed.Setup}, \mathsf{Mixed.Enc}, \mathsf{Mixed.SK\text{-}Enc}, \mathsf{Mixed.KeyGen}, \mathsf{Mixed.Dec})$ be a mixed functional encryption scheme, for function classes $\{\mathcal{F}_\kappa\}_\kappa$ and message space $\{\mathcal{I}_\kappa\}_\kappa$, with ciphertexts of length $\ell(\lambda, \kappa)$. For every n, let $\kappa = \kappa(n)$ be the lexicographically smallest functionality index such that every string of length $\log(n)$ can be uniquely represented in message space $\mathcal{I}_\kappa$ (i.e., $\{0,1\}^{\log(n)} \subseteq \mathcal{I}_\kappa$), and function class $\mathcal{F}_\kappa$ contains the "comparison" ($>$) operator. Also, let $\widetilde{\kappa} = \widetilde{\kappa}(\lambda, \kappa)$ be the lexicographically smallest functionality index such that every string of length $\ell(\lambda, \kappa)$ can be uniquely represented in attribute class $\mathcal{X}_{\widetilde{\kappa}}$ (i.e., $\{0,1\}^{\ell(\lambda,\kappa)} \subseteq \mathcal{X}_{\widetilde{\kappa}}$), and $\mathcal{C}_{\widetilde{\kappa}}$ contains mixed FE decryption circuit corresponding to functionality index κ. Below we describe our construction.

- $\mathsf{Setup}(1^\lambda, 1^n) \to \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i \le n}\right)$. The setup algorithm runs $\mathsf{ABE.Setup}$ and $\mathsf{Mixed.Setup}$ to generate ABE and mixed FE public parameters and master secret key as $(\mathsf{abe.pp}, \mathsf{abe.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\widetilde{\kappa}})$ and $(\mathsf{mixed.pp}, \mathsf{mixed.msk}) \leftarrow \mathsf{Mixed.Setup}(1^\lambda, 1^\kappa)$. Next, it runs $\mathsf{Mixed.KeyGen}$ to generate n mixed secret keys $\mathsf{mixed.sk}_i$ as

  $$\forall\, i \le n, \quad \mathsf{mixed.sk}_i \leftarrow \mathsf{Mixed.KeyGen}(\mathsf{mixed.msk}, i).$$

  Let $C_{\mathsf{mixed.sk}_i}$ denote the circuit $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$, i.e. $\mathsf{Mixed\text{-}FE}$ decryption circuit with key $\mathsf{mixed.sk}_i$ hardwired. Next, it computes n ABE secret keys $\mathsf{abe.sk}_i$ as

  $$\forall\, i \le n, \quad \mathsf{abe.sk}_i \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}, C_{\mathsf{mixed.sk}_i}).$$

  Finally, it sets $\mathsf{pp} = (\mathsf{abe.pp}, \mathsf{mixed.pp})$, $\mathsf{msk} = (\mathsf{abe.msk}, \mathsf{mixed.msk})$ and $\mathsf{sk}_i = \mathsf{abe.sk}_i$ for $i \le n$.

- $\mathsf{Enc}(\mathsf{pp}, m) \to \mathsf{ct}$. Let $\mathsf{pp} = (\mathsf{abe.pp}, \mathsf{mixed.pp})$. The encryption algorithm first computes $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.Enc}(\mathsf{mixed.pp})$. Next, it encrypts message m as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$, and outputs ciphertext ct.

- $\mathsf{Enc\text{-}index}(\mathsf{msk}, m, i) \to \mathsf{ct}$. Let $\mathsf{msk} = (\mathsf{abe.msk}, \mathsf{mixed.msk})$ and comp_i denote the comparison function $\overset{?}{>} i$, i.e. $\mathsf{comp}_i(x) = 1$ iff $x > i$. The encryption algorithm first computes $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.SK\text{-}Enc}(\mathsf{mixed.msk}, \mathsf{comp}_i)$. Next, it encrypts message m as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$, and outputs ciphertext ct.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$ or $\perp$. The decryption algorithm runs $\mathsf{ABE.Dec}$ on ct using key sk as $y = \mathsf{ABE.Dec}(\mathsf{sk}, \mathsf{ct})$, and sets y as the output of decryption.

6.2 Correctness

For all $\lambda, n \in \mathbb{N}$, message $m \in \mathcal{M}_\lambda$, public parameters and master secret keys $(\mathsf{abe.pp}, \mathsf{abe.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\widetilde{\kappa}})$, $(\mathsf{mixed.pp}, \mathsf{mixed.msk}) \leftarrow \mathsf{Mixed.Setup}(1^\lambda, 1^\kappa)$, the secret keys sk_i for $i \le n$ are simply the ABE keys $\mathsf{abe.sk}_i \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}, C_{\mathsf{mixed.sk}_i})$. For any index $i \le n$, consider the following two cases:

1. **Normal encryption.** For any ciphertext ct computed as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct_{attr}}, m)$, where $\mathsf{ct_{attr}} \leftarrow \mathsf{Mixed.Enc}(\mathsf{mixed.pp})$, we know that with all but negligible probability $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \mathsf{ct_{attr}}) = 1$ by correctness of mixed scheme. In other words, $C_{\mathsf{mixed.sk}_i}(\mathsf{ct_{attr}}) = 1$. Therefore, by correctness of ABE scheme, we have that with all but negligible probability $\mathsf{ABE.Dec}(\mathsf{abe.sk}_i, \mathsf{ct}) = m$.

2. **Index encryption.** For any index $j \in \{0, 1, \ldots, n\}$ and ciphertext ct computed as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct_{attr}}, m)$, where $\mathsf{ct_{attr}} \leftarrow \mathsf{Mixed.SK\text{-}Enc}(\mathsf{mixed.msk}, \mathsf{comp}_j)$, we know that with all but negligible probability
$$\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \mathsf{ct_{attr}}) = \begin{cases} 1 \text{ if } i > j \\ 0 \text{ otherwise} \end{cases}$$
by correctness of mixed scheme. In other words, $C_{\mathsf{mixed.sk}_i}(\mathsf{ct_{attr}}) = \mathsf{comp}_j(i) = (i > j)$. Therefore, by correctness of ABE scheme, we have that with all but negligible probability $\mathsf{ABE.Dec}(\mathsf{abe.sk}_i, \mathsf{ct}) = m$ for $i > j$ and $\perp$ otherwise.

Therefore, PLBE satisfies the PLBE correctness condition.

6.3 Security

We will now show that the scheme described above is 1-bounded secure as per Definitions 3.3, 3.4 and 3.5. In other words, it satisfies normal hiding, index hiding, and message hiding security properties. Formally, we prove the following.

Theorem 6.1. If $\mathcal{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.Enc}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Dec})$ is a selectively-secure attribute based encryption for set of attribute spaces $\{\mathcal{X}_\kappa\}_\kappa$, predicate classes $\{\mathcal{C}_\kappa\}_\kappa$ and message spaces $\{\mathcal{M}_\kappa\}_\kappa$ satisfying Definition A.2, and $\mathsf{Mixed\text{-}FE} = (\mathsf{Mixed.Setup}, \mathsf{Mixed.Enc}, \mathsf{Mixed.SK\text{-}Enc}, \mathsf{Mixed.KeyGen}, \mathsf{Mixed.Dec})$ is a mixed functional encryption scheme, for function classes $\{\mathcal{F}_\kappa\}_\kappa$ and message spaces $\{\mathcal{I}_\kappa\}_\kappa$, satisfying 1-bounded restricted function indistinguishability (Definition 5.2) and 1-bounded restricted accept indistinguishability (Definition 5.4) properties, then $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ is a secure private linear broadcast encryption scheme, for messages spaces $\{\mathcal{M}_\kappa\}_\kappa$, satisfying 1-bounded normal, index and message hiding security properties as per Definitions 3.3, 3.4 and 3.5 (resp.).

Our proof is divided in three components/lemmas, one for each PLBE security property. Let $\mathcal{A}$ be any PPT adversary that wins the normal/index/message hiding game with non-negligible advantage. We argue that such an adversary must break security of at least one underlying primitive.

6.3.1 Normal Hiding Security

Lemma 6.1. If $\mathsf{Mixed\text{-}FE} = (\mathsf{Mixed.Setup}, \mathsf{Mixed.Enc}, \mathsf{Mixed.SK\text{-}Enc}, \mathsf{Mixed.KeyGen}, \mathsf{Mixed.Dec})$ is a mixed functional encryption scheme satisfying 1-bounded restricted accept indistinguishability (Definition 5.4) property, then $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ is a private linear broadcast encryption scheme satisfying 1-bounded normal hiding security property as per Definition 3.3.

Proof. Suppose there exists an adversary $\mathcal{A}$ such that $\mathcal{A}$'s advantage in 1-bounded normal hiding security game is non-negligible. We construct an algorithm $\mathcal{B}$ that can distinguish normal encryptions from secret key encryptions , therefore break 1-bounded restricted accept indistinguishability security of the mixed FE scheme.

The reduction algorithm $\mathcal{B}$ receives 1^n from $\mathcal{A}$. It sets $\kappa, \widetilde{\kappa}$ as in the construction, and sends κ as the functionality index and comp_0 (i.e., comparison with 0) as its challenge function to the mixed FE challenger. The challenger generates a key pair $(\mathsf{mixed.pp}, \mathsf{mixed.msk})$ and sends $\mathsf{mixed.pp}$ as the public parameters and challenge ciphertext $\mathsf{ct}^*_{\mathsf{attr}}$ to $\mathcal{B}$. Next, $\mathcal{B}$ makes an encryption query for function comp_0. Let the challenger's response be ciphertext $\mathsf{ct_{attr}}$. $\mathcal{B}$ then queries the challenger on n messages $i (\leq n)$ for corresponding mixed secret keys, and receives back keys $\mathsf{mixed.sk}_i$ for $i \leq n$. It then chooses an ABE key pair $(\mathsf{abe.pp}, \mathsf{abe.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\widetilde{\kappa}})$, and computes n ABE keys as $\mathsf{abe.sk}_i \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}, C_{\mathsf{mixed.sk}_i})$. Next, it sends

$(\mathsf{abe.pp}, \mathsf{mixed.pp})$ and $\{\mathsf{abe.sk}_i\}_{i \leq n}$ as the PLBE public parameters and secret keys to $\mathcal{A}$. After receiving all the keys, $\mathcal{A}$ sends its challenge message m^* to $\mathcal{B}$, and it can also make a single encryption query for message m on index 0. Here $\mathcal{A}$ is allowed to make the encryption query either before or after challenge query. The reduction algorithm $\mathcal{B}$ responds to each query as follows. $\mathcal{B}$ encrypts message m as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$, and sends ct to $\mathcal{A}$ as its response to encryption query. Also, it computes ciphertext ct^* as $\mathsf{ct}^* \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}^*_{\mathsf{attr}}, m^*)$, and sends ct^* as the challenge ciphertext to $\mathcal{A}$. Note that $\mathcal{A}$ could instead have sent its challenge query before sending the index encryption query. Also, $\mathcal{B}$ does not need to query mixed FE challenger for answering any query at this point at it already has ciphertexts $\mathsf{ct}_{\mathsf{attr}}, \mathsf{ct}^*_{\mathsf{attr}}$. Finally, $\mathcal{A}$ sends its guess b to $\mathcal{B}$, and $\mathcal{B}$ forwards b as its own guess.

First, note that both $\mathcal{A}$ and $\mathcal{B}$ are allowed to make at most 1 index encryption and secret key encryption queries, respectively. Also, note that $\mathcal{B}$ sends its secret key encryption query as well as challenge query before making making key generation queries, thus $\mathcal{B}$ is an admissible adversary in the 1-bounded restricted accept indistinguishability game. Since $\mathcal{A}$ is only allowed to make encryption queries to index 0 (in the 1-bounded normal hiding security game), thus $\mathcal{B}$ queries mixed FE challenger on functions comp_0 which are always accepting functions and therefore admissible queries as per restricted accept indistinguishability game. Next, for each query made by $\mathcal{A}$, $\mathcal{B}$ queries mixed FE challenger exactly once, thus the all the queries are honestly and exactly answered. Finally, note that if mixed FE challenger computed $\mathsf{ct}^*_{\mathsf{attr}}$ as a normal FE ciphertext, then $\mathcal{B}$ computes ct^* as a normal PLBE ciphertext, otherwise it computes ct^* as a PLBE ciphertext for index 0. Thus, $\mathcal{B}$ perfectly simulates the 1-bounded normal hiding security game for $\mathcal{A}$. As a result, if $\mathcal{A}$'s advantage is non-negligible, then $\mathcal{B}$ breaks 1-bounded restricted accept indistinguishability security with non-negligible advantage. This completes the proof. ■

6.3.2 Index Hiding Security

Lemma 6.2. If $\mathsf{Mixed\text{-}FE} = (\mathsf{Mixed.Setup}, \mathsf{Mixed.Enc}, \mathsf{Mixed.SK\text{-}Enc}, \mathsf{Mixed.KeyGen}, \mathsf{Mixed.Dec})$ is a mixed functional encryption scheme satisfying 1-bounded restricted function indistinguishability (Definition 5.2) property, then $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ is a private linear broadcast encryption scheme satisfying 1-bounded index hiding security property as per Definition 3.4.

Proof. The proof of this lemma is similar to that of Lemma 6.1, with the additional modification that the reduction algorithm now guesses the indices i, i^* on which the PLBE adversary makes its encryption query and challenge query, respectively. And, the reduction algorithm aborts if its guess is incorrect. This leads to a polynomial loss ($\approx 1/n^2$) in the reduction algorithm's advantage.[17]

Suppose there exists an adversary $\mathcal{A}$ such that $\mathcal{A}$'s advantage in 1-bounded index hiding security game is non-negligible. We construct an algorithm $\mathcal{B}$ that can distinguish between two secret key encryptions , therefore break 1-bounded restricted function indistinguishability security of the mixed FE scheme.

The reduction algorithm $\mathcal{B}$ receives 1^n from $\mathcal{A}$. It sets $\kappa, \widetilde{\kappa}$ as in the construction. Next, it guesses the challenge index $i^* \in \{0, \ldots, n-1\}$ and query index $i \in \{0, \ldots, n\}$.[18] It sends κ as the functionality index and $(\mathsf{comp}_{i^*}, \mathsf{comp}_{i^*+1})$ (i.e., comparison with i^* and i^*+1) as its challenge functions to the mixed FE challenger. The challenger generates a key pair $(\mathsf{mixed.pp}, \mathsf{mixed.msk})$ and sends $\mathsf{mixed.pp}$ as the public parameters and challenge ciphertext $\mathsf{ct}^*_{\mathsf{attr}}$ to $\mathcal{B}$. Next, $\mathcal{B}$ makes an encryption query for function comp_i. Letthe challenger's response be ciphertext $\mathsf{ct}_{\mathsf{attr}}$. $\mathcal{B}$ then queries the challenger on $n-1$ messages $j (\in [n] \setminus \{i^*+1\})$ for corresponding mixed secret keys, and receives back keys $\mathsf{mixed.sk}_j$ for each j. It then chooses an ABE key pair $(\mathsf{abe.pp}, \mathsf{abe.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\widetilde{\kappa}})$, and computes $n-1$ ABE keys as $\mathsf{abe.sk}_j \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}, C_{\mathsf{mixed.sk}_j})$. Next, it sends $(\mathsf{abe.pp}, \mathsf{mixed.pp})$ and $\{\mathsf{abe.sk}_j\}_{j \in [n] \setminus \{i^*+1\}}$ as the PLBE public parameters and secret keys to $\mathcal{A}$. After receiving all the keys, $\mathcal{A}$ sends its challenge message m^* to

[17]Due to the fact that the reduction algorithm has to guess the index, we can only extend the current analysis to prove q-bounded PLBE (adaptive) security assuming q-bounded mixed FE (restricted) security for constant q. However, we would like to point out that one could prove q-bounded PLBE *selective* security directly from q-bounded mixed FE (restricted) security without any security loss.

[18]Basically, the reduction algorithm guesses two things — first, it guesses the index hiding challenger with which $\mathcal{A}$ interacts and wins with non-negligible probability; second, it guesses the index on which adversary $\mathcal{A}$ queries the PLBE challenger for index encryption.

$\mathcal{B}$, and it can also make an encryption query for message m on index $\tilde{i}$. Here $\mathcal{A}$ is allowed to make the encryption query either before or after challenge query. The reduction algorithm $\mathcal{B}$ proceeds as follows. If $i \neq \tilde{i}$, $\mathcal{B}$ aborts and sends a random bit as its guess to the mixed FE challenger. Otherwise, it responds to each query as follows. $\mathcal{B}$ encrypts message m as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$, and sends ct to $\mathcal{A}$ as its response to encryption query. Also, it computes ciphertext ct^* as $\mathsf{ct}^* \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}^*_{\mathsf{attr}}, m^*)$, and sends ct^* as the challenge ciphertext to $\mathcal{A}$. Note that $\mathcal{A}$ could instead have sent its challenge query before sending the index encryption query. Also, $\mathcal{B}$ does not need to query mixed FE challenger for answering any query at this point at it already has ciphertexts $\mathsf{ct}_{\mathsf{attr}}, \mathsf{ct}^*_{\mathsf{attr}}$. Finally, $\mathcal{A}$ sends its guess b to $\mathcal{B}$, and $\mathcal{B}$ forwards b as its own guess.

First, note that both $\mathcal{A}$ and $\mathcal{B}$ are allowed to make at most 1 index encryption and secret key encryption queries, respectively. Also, note that $\mathcal{B}$ sends its secret key encryption query as well as challenge query before making making key generation queries, thus $\mathcal{B}$ is an admissible adversary in the 1-bounded restricted function indistinguishability game. Next, for each query made by $\mathcal{A}$, $\mathcal{B}$ queries mixed FE challenger exactly once, thus the all the queries are honestly and exactly answered. Finally, note that if mixed FE challenger computed $\mathsf{ct}^*_{\mathsf{attr}}$ as an secret key FE ciphertext for function comp_{i^*}, then $\mathcal{B}$ computes ct^* as a PLBE ciphertext for index i^*, otherwise it computes ct^* as a PLBE ciphertext for index i^*+1. Thus, $\mathcal{B}$ perfectly simulates the 1-bounded index hiding security game for $\mathcal{A}$. Also, since $\mathcal{B}$ randomly guesses the challenge index i^* as well as query index i, therefore with at least $1/n(n+1)$ probability $\mathcal{B}$'s guess will be correct, thus if $\mathcal{A}$'s advantage is (non-negligible) ϵ, then $\mathcal{B}$ breaks 1-bounded restricted function indistinguishability security with (non-negligible) advantage $\epsilon/n(n+1)$. This completes the proof. ∎

6.3.3 Message Hiding Security

Lemma 6.3. If $\mathcal{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.Enc}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Dec})$ is a selectively-secure attribute based encryption as per Definition A.2, then $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ is a private linear broadcast encryption scheme satisfying 1-bounded message hiding security property as per Definition 3.5.

Proof. Suppose there exists an adversary $\mathcal{A}$ such that $\mathcal{A}$'s advantage in 1-bounded message hiding security game is non-negligible. We construct an algorithm $\mathcal{B}$ that can distinguish between ABE encryptions of two different messages, therefore break security of the ABE scheme.

The reduction algorithm receives 1^n from $\mathcal{A}$. It sets $\kappa, \widetilde{\kappa}$ as in the construction, and starts by choosing mixed FE parameters as $(\mathsf{mixed.pp}, \mathsf{mixed.msk}) \leftarrow \mathsf{Mixed.Setup}(1^\lambda, 1^\kappa)$. It then computes $\mathsf{ct}^*_{\mathsf{attr}} \leftarrow \mathsf{Mixed.SK\text{-}Enc}(\mathsf{mixed.msk}, \mathsf{comp}_n)$, and sends to the ABE challenger $1^{\widetilde{\kappa}}$ and $\mathsf{ct}^*_{\mathsf{attr}}$ as its challenge attribute. The ABE challenger generates a key pair $(\mathsf{abe.pp}, \mathsf{abe.sk})$ and sends $\mathsf{abe.pp}$ to $\mathcal{B}$. For $i \leq n$, $\mathcal{B}$ generates mixed FE secret keys as $\mathsf{mixed.sk}_i \leftarrow \mathsf{Mixed.KeyGen}(\mathsf{mixed.msk}, i)$, and sends $C_{\mathsf{mixed.sk}_i}$ as a predicate query to the ABE challenger and receives back secret key $\mathsf{abe.sk}_i$. Next, it sends $(\mathsf{abe.pp}, \mathsf{mixed.pp})$ and $\{\mathsf{abe.sk}_i\}_{i \leq n}$ as the PLBE public parameters and secret keys to $\mathcal{A}$. After receiving all the keys, $\mathcal{A}$ makes a single index encryption query (m, j) to $\mathcal{B}$. $\mathcal{B}$ answers it by computing ciphertexts $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.SK\text{-}Enc}(\mathsf{mixed.msk}, \mathsf{comp}_j)$ and $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$, and sends ct to $\mathcal{A}$ as its response. $\mathcal{A}$ also sends two challenge message (m_0^*, m_1^*) to $\mathcal{B}$. $\mathcal{B}$ then forwards (m_0^*, m_1^*) as its challenge messages to ABE challenger. Next, $\mathcal{B}$ forwards the challenge ciphertext ct^* it receives from ABE challenger to $\mathcal{A}$. Note that $\mathcal{A}$ could instead have sent its challenge query before sending the index encryption query. In that case, the reduction algorithm simply answers that first. Finally, $\mathcal{A}$ sends its guess b to $\mathcal{B}$, and $\mathcal{B}$ forwards b as its own guess.

First, note that the challenge attribute $\mathsf{ct}^*_{\mathsf{attr}}$ on each predicate $(C_{\mathsf{mixed.sk}_i})$ queried made by $\mathcal{B}$ evaluates to 0, with all but negligible probability. This follows from the correctness condition of mixed FE system as $\mathsf{ct}^*_{\mathsf{attr}}$ encrypts function comp_n and for all $i \leq n$, $\mathsf{comp}_n(i) = 0$, thus decrypting $\mathsf{ct}^*_{\mathsf{attr}}$ using $\mathsf{mixed.sk}_i$ outputs 0. Thus, with all-but-negligible probability, reduction algorithm $\mathcal{B}$ is an admissible adversary in the ABE security game. Thus, $\mathcal{B}$ perfectly simulates the 1-bounded[19] message hiding security game for $\mathcal{A}$. As a

[19]We would like to point out that the current construction actually gives a PLBE scheme that satisfies q-bounded message hiding security property for arbitrary q, i.e. the number of queries need not be bounded, as long as the ABE scheme is not q-bounded selectively-secure.

result, if $\mathcal{A}$'s advantage is non-negligible, then $\mathcal{B}$ breaks ABE security with non-negligible advantage. This completes the proof. ■

7 A New LWE Toolkit

In Section 2, we defined the notion of lattice trapdoors along with certain well-sampledness properties. Recall that using lattice trapdoors, it is easy to compute pre-image $\mathbf{U}$ of any matrix $\mathbf{Z}$ with respect to a matrix $\mathbf{A}$ given the trapdoor information generated while sampling $\mathbf{A}$. For any matrix $\mathbf{U}$ sampled as above, we say $\mathbf{U}$ targets $\mathbf{A}$ to matrix $\mathbf{Z}$. Now the well-sampledness properties (at a high level) state that a matrix sampled using $\mathsf{TrapGen}$ algorithm looks uniformly random when not given the trapdoor information, and a pre-image matrix that targets to a random matrix looks like a matrix with entries drawn from a Gaussian distribution.

In this section, we introduce certain enhanced security properties for lattice trapdoors that will be useful later in proving security of our Mixed FE system. We also provide a generic construction of lattice trapdoors that achieves these enhanced properties from any lattice trapdoor scheme that satisfies the well-sampledness properties described above. At a very high level, the enhanced security properties state the following — (1) For any matrix $\mathbf{A}$ that is sampled using $\mathsf{TrapGen}$ algorithm, all those rows of $\mathbf{A}$ which are only used to target random rows look like random rows themselves (when not given the trapdoor information), (2) Two pre-image matrices $\mathbf{U}_0, \mathbf{U}_1$ that target a matrix $\mathbf{A}$ to different matrices $\mathbf{Z}_0, \mathbf{Z}_1$ should look indistinguishable to any adversary even when the adversary is given those rows of $\mathbf{A}$ where $\mathbf{Z}_0, \mathbf{Z}_1$ are identical. We point out that due to technical constrainst in the proof of property (2) we chose to define the enhanced properties w.r.t. matrices instead of vectors.

7.1 Enhanced Lattice Trapdoors

Let $(\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ be a pair of randomized algorithms with the following syntax:

$\mathsf{EnTrapGen}(1^n, 1^m, q) \to (\mathbf{A}, T_{\mathbf{A}})$. The trapdoor generation algorithm takes as input n, m, q and outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ together with a trapdoor $T_{\mathbf{A}}$.

$\mathsf{EnSamplePre}(\mathbf{A}, T_{\mathbf{A}}, \sigma, \mathbf{z}) \to \mathbf{u}$. The pre-image sampling algorithm takes as input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ together with its trapdoor $T_{\mathbf{A}}$, a target vector $\mathbf{z} \in \mathbb{Z}_q^n$ and a parameter σ.[20] It outputs $\mathbf{u} \in \mathbb{Z}_q^m$ such that $\mathbf{A} \cdot \mathbf{u}^T = \mathbf{z}^T$ and $\|\mathbf{u}\| \le \sqrt{m} \cdot \sigma$.

We require these algorithms to satisfy the following properties. These properties are captured via security games between a challenger and a computationally bounded adversary.

Notation. First, we introduce some more notation. We start by defining a matrix re-arrangement procedure $\mathsf{Arrange}$ which takes as input dimensions n_1, n_2, m, and two matrices $\mathbf{A} \in \mathbb{Z}_q^{n_1\times m}, \mathbf{B} \in \mathbb{Z}_q^{n_2\times m}$, and an *ordered* set $S \subseteq [n_1 + n_2]$ of size n_1, and it outputs a larger combined matrix $\mathbf{C} \in \mathbb{Z}_q^{(n_1+n_2)\times m}$. Concretely, $\mathbf{C} = \mathsf{Arrange}(1^{n_1}, 1^{n_2}, 1^m, \mathbf{A}, \mathbf{B}, S)$. The re-arrangement procedure is defined as follows.

Let $S = \{i_1, i_2, \ldots, i_{n_1}\}$, where $i_j < i_k$ for every $1 \le j < k \le n_1$. Similarly, let $\overline{S} = ([n_1 + n_2] \setminus S) = \{i'_1, i'_2, \ldots, i'_{n_2}\}$ denote the (ordered) complement of set S. Now matrix $\mathbf{C}$ is obtained by appending rows of matrices $\mathbf{A}$ and $\mathbf{B}$ as follows — for $j \in [n_1]$, $\mathbf{C}[i_j] = \mathbf{A}[j]$, and for $j \in [n_2]$, $\mathbf{C}[i'_j] = \mathbf{B}[j]$. For simplicity of notation, we drop the dimensions n_1, n_2, m as explicit inputs to $\mathsf{Arrange}$ procedure throughout this section whenever clear from context.

Additionally, we define a matrix restriction procedure $\mathsf{Restrict}$ which takes as input dimensions n, m, and matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, and an *ordered* set $S \subseteq [n]$ of size ℓ, and it outputs a smaller matrix $\mathbf{C} \in \mathbb{Z}_q^{\ell\times m}$. Concretely, $\mathbf{C} = \mathsf{Restrict}(1^n, 1^m, \mathbf{A}, S)$. The restriction procedure is defined as follows.

[20]As before, the pre-image sampling algorithm could be easily generalized to generate pre-images of matrices instead of vectors by independently running $\mathsf{EnSamplePre}$ algorithm on each column of the matrix. Throughout this work, we overload the notation by directly giving matrices $\mathbf{U} \in \mathbb{Z}_q^{n\times k}$ (for any k) as inputs to the $\mathsf{SamplePre}$ algorithm.

Let $S = \{i_1, i_2, \ldots, i_n\}$, where $i_j < i_k$ for every $1 \leq j < k \leq \ell$. Now matrix $\mathbf{C}$ is obtained by removing rows of matrix $\mathbf{A}$ which do not lie in set S. Formally, for $j \in [\ell]$, $\mathbf{C}[j] = \mathbf{A}[i_j]$. As before, for simplicity of notation, we drop the dimensions n, m as inputs whenever clear from context.

7.1.1 Row Removal Property

The first property we introduce is called the *row removal* property. It is defined via as an interactive security game between the challenger and an adversary. In the game, the adversary specifies matrix dimensions n, m, a set of k ($\leq n$) indices, which represent the 'target' set, and the adversary must distinguish between the following scenarios.

In the first scenario, the challenger chooses an $n \times m$ matrix $\mathbf{A}$ with a trapdoor, and sends $\mathbf{A}$ to the adversary. The adversary then participates in a query phase. For each query, the adversary sends a set of k target vectors. The challenger responds by outputting a matrix $\mathbf{U}$ such that for each index i in the target set, $\mathbf{U}$ maps the i^{th} row of $\mathbf{A}$ to one of the target vectors. The matrix $\mathbf{U}$ maps the remaining rows of $\mathcal{A}$ to uniformly random vectors.

In the second scenario, the challenger chooses a $k \times m$ matrix $\mathbf{A}$ with a trapdoor, extends $\mathbf{A}$ to dimension $n \times m$ by attaching uniformly random rows, and sends this extended matrix to the adversary. Next, the adversary sends queries, each query consisting of k target vectors. The challenger outputs a matrix $\mathbf{U}$ such that $\mathbf{U}$ maps the i^{th} row of $\mathcal{A}$ to the i^{th} target vector.[21]

Definition 7.1 (Row Removal Property)**.** Fix any function $q : \mathbb{N} \to \mathbb{N}$ and parameter $\sigma : \mathbb{N} \to \mathbb{R}^+$. A pair of trapdoor generation algorithms $\mathsf{LT} = (\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ is said to satisfy the (q, σ)-*row removal* property if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$

$$\mathsf{pr}^{\text{row-rem},q,\sigma}_{\mathsf{LT},\mathcal{A}}(\lambda) = \Pr\left[1 \leftarrow \mathsf{Expt}^{\text{row-rem},q,\sigma}_{\mathsf{LT},\mathcal{A}}(\lambda)\right] \leq 1/2 + \text{negl}(\lambda),$$

where $\mathsf{Expt}^{\text{row-rem},q,\sigma}_{\mathsf{LT},\mathcal{A}}(\cdot)$ is defined in Figure 4.

$\mathsf{Expt}^{\text{row-rem},q,\sigma}_{\mathsf{LT},\mathcal{A}}(\lambda)$

1. **Setup Phase.** The adversary $\mathcal{A}$ after receiving as input the security parameter λ, sends dimensions 1^n, 1^m, and a set $S \subseteq [n]$ of size k, such that $m > 2n \log q + 2\lambda$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda$, to the challenger. The challenger chooses a random bit $b \leftarrow \{0, 1\}$, and proceeds as follows:
   (a) If $b = 0$, it samples two matrices $\mathbf{B}, \mathbf{P}$ as $(\mathbf{B}, T_{\mathbf{B}}) \leftarrow \mathsf{EnTrapGen}(1^k, 1^m, q)$, $\mathbf{P} \leftarrow \mathbb{Z}_q^{(n-k) \times m}$. Next, it sets $\mathbf{A} = \mathsf{Arrange}(\mathbf{B}, \mathbf{P}, S)$, and sends $\mathbf{A}$ to the adversary.
   (b) Otherwise, if $b = 1$, it chooses matrix $\mathbf{A}$ as $(\mathbf{A}, T_{\mathbf{A}}) \leftarrow \mathsf{EnTrapGen}(1^n, 1^m, q)$. And, sends $\mathbf{A}$ to the adversary.
2. **Query Phase.** The adversary makes a polynomial number of pre-image queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{k \times t}$. The challenger responds to each query as follows.
   (a) If $b = 0$, it samples matrix $\mathbf{U}$ as $\mathbf{U} \leftarrow \mathsf{EnSamplePre}(\mathbf{B}, T_{\mathbf{B}}, \sigma, \mathbf{C})$. And, sends $\mathbf{U}$ to the adversary.
   (b) Otherwise if $b = 1$, it chooses a random matrix $\mathbf{Q}$ as $\mathbf{Q} \leftarrow \mathbb{Z}_q^{(n-k) \times t}$, and sets matrix $\mathbf{D}$ as $\mathbf{D} = \mathsf{Arrange}(\mathbf{C}, \mathbf{Q}, S)$. Next, it samples matrix $\mathbf{U}$ as $\mathbf{U} \leftarrow \mathsf{EnSamplePre}(\mathbf{A}, T_{\mathbf{A}}, \sigma, \mathbf{D})$, and sends $\mathbf{U}$ to the adversary.
3. $\mathcal{A}$ sends its guess b'. The experiment outputs 1 iff $b = b'$.

Figure 4: Experiment $\mathsf{Expt}^{\text{row-rem},q,\sigma}_{\mathsf{LT},\mathcal{A}}$

[21]Although one might observe some weak resemblance between our row removal property and lattice trapdoor properties used in [ALS16, BF11], we would like to point out that after a closer inspection we observe that our row removal property is different.

A weaker row removal property is one where, in each query, the adversary is restricted to choose a set S of size $n-1$ during setup phase, and now during query phase it must make pre-image queries of the form $(1^t, \mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t})$. A different way to represent the set S in this case is by a single index $i \in [n]$ such that $\{i\} = [n] \setminus S$. We call this property, the single row removal property. In our construction, we will first show a scheme that satisfies the single row removal property, and then show that single row removal property implies row removal property via a simple hybrid argument.

7.1.2 Target Switching Property

Next, we introduce the *target switching* property. For any target $\mathbf{Z}$, if we choose a matrix $\mathbf{B}$ with a trapdoor, and output *only* the pre-image of $\mathbf{Z}$ with respect to $\mathbf{B}$, then this pre-image looks like a random low-norm matrix. The target switching property is an extension of this property, and is captured via a security game between a challenger and an adversary. In this game, the challenger specifies the matrix dimensions n, m and a set $S \subseteq [n]$ of size k. The challenger chooses an $n \times m$ matrix $\mathbf{B}$ and sends the rows of $\mathbf{B}$ corresponding to the set S. It also chooses a challenge bit b which is used in the query phase.

Next, the adversary is allowed polynomially many queries. In each query, the adversary specifies two matrices $\mathbf{Z}_0, \mathbf{Z}_1$ such that for every index $i \in S$, the i^{th} rows of $\mathbf{Z}_0$ and $\mathbf{Z}_1$ are identical. The challenger outputs a matrix $\mathbf{U}$ such that for every $i \in S$, $\mathbf{U}$ maps the i^{th} row of $\mathbf{B}$ to the i^{th} row of $\mathbf{Z}_0$ (which is equal to the i^{th} row of $\mathbf{Z}_1$). For the remaining indices $i \notin S$, $\mathbf{U}$ *approximately* maps the i^{th} row of $\mathbf{A}$ to the i^{th} row of $\mathbf{Z}_b$. Intuitively, since the adversary does not have the rows indexed by $\overline{S}$, the challenger can switch the targets from rows of $\mathbf{Z}_0$ to $\mathbf{Z}_1$.

Definition 7.2 (Target Switching Property)**.** Fix any function $q : \mathbb{N} \to \mathbb{N}$, noise distribution family $\{\chi(\lambda)\}_{\lambda\in\mathbb{N}}$ and parameter $\sigma : \mathbb{N} \to \mathbb{R}^+$. A pair of trapdoor generation algorithms $\mathsf{LT} = (\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ is said to satisfy the (q, χ, σ)-*target switching* property if for any PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q = q(\lambda)$, $\chi = \chi(\lambda)$, $\sigma = \sigma(\lambda)$,

$$\mathsf{pr}^{\text{switch},q,\chi,\sigma}_{\mathsf{LT},\mathcal{A}}(\lambda) = \Pr\left[1 \leftarrow \mathsf{Expt}^{\text{switch},q,\chi,\sigma}_{\mathsf{LT},\mathcal{A}}(\lambda)\right] \leq 1/2 + \text{negl}(\lambda),$$

where $\mathsf{Expt}^{\text{switch},q,\chi,\sigma}_{\mathsf{LT},\mathcal{A}}(\cdot)$ is defined in Figure 5.

$\mathsf{Expt}^{\text{switch},q,\chi,\sigma}_{\mathsf{LT},\mathcal{A}}(\cdot)$

1. **Setup Phase.** The adversary $\mathcal{A}$ after receiving as input the security parameter λ, sends dimensions 1^n, 1^m, set $S \subseteq [n]$ of size k, such that $m > 2n \log q + 12\lambda \cdot \log q$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda$, to the challenger. The challenger chooses a random bit $b \in \{0,1\}$, and proceeds as follows:
   (a) It samples matrix $\mathbf{A}$ as $(\mathbf{A}, T_{\mathbf{A}}) \leftarrow \mathsf{EnTrapGen}(1^n, 1^m, q)$.
   (b) Next, it sets $\mathbf{B} = \mathsf{Restrict}(\mathbf{A}, S)$, and sends $\mathbf{B}$ to the adversary.
2. **Query Phase.** The adversary makes polynomially many queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$, where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n\times t}$ and $\mathsf{Restrict}(\mathbf{Z}_0, S) = \mathsf{Restrict}(\mathbf{Z}_1, S)$. The challenger responds to each query as follows.
   (a) It chooses matrix $\mathbf{E}$ as $\mathbf{E} \leftarrow \chi^{(n-k)\times t}$, and sets $\mathbf{F} = \mathsf{Arrange}(\mathbf{0}^{k\times t}, \mathbf{E}, S)$.
   (b) Next, it samples matrix $\mathbf{U}$ as $\mathbf{U} \leftarrow \mathsf{EnSamplePre}(\mathbf{A}, T_{\mathbf{A}}, \sigma, \mathbf{Z}_b + \mathbf{E})$, and sends $\mathbf{U}$ to the adversary.
3. $\mathcal{A}$ sends its guess b', and the experiment outputs 1 iff $b = b'$.

Figure 5: $\mathsf{Expt}^{\text{switch},q,\chi,\sigma}_{\mathsf{LT},\mathcal{A}}(\cdot)$

As before, we will introduce a weaker notion called single target switching property, where in each query, the adversary is restricted to output only a single index $i \in [n]$, and $\mathbf{Z}_0$ and $\mathbf{Z}_1$ must agree on all indices $j \neq i$. We will first show that our construction satisfies the single target switching property, and then show that single target switching implies target switching property via a hybrid argument.

7.2 Our Construction of Enhanced Lattice Trapdoors

Let $q : \mathbb{N} \to \mathbb{N}, \sigma : \mathbb{N} \to \mathbb{R}^+$ be functions, and $\mathsf{LT} = (\mathsf{TrapGen}, \mathsf{SamplePre})$ a pair of algorithms that satisfy q-well sampledness of matrix (Definition 2.2), (q, σ)-well sampledness of preimage (Definition 2.3). We will construct enhanced lattice trapdoors $\mathsf{LT}_{\mathsf{en}} = (\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ using LT as follows. The construction is reminiscient of the trapdoor extension algorithms of [ABB10, CHKP10].

$\mathsf{EnTrapGen}(1^n, 1^m, q) \to (\mathbf{A}, T_{\mathbf{A}})$. The trapdoor generation algorithm samples two matrices $\mathbf{A}_1, \mathbf{A}_2$ of dimensions $\lceil m/2 \rceil \times n$ and $\lfloor m/2 \rfloor \times n$ as follows

$$(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lceil m/2 \rceil}, q),$$
$$(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q).$$

It appends both these matrices column-wise to obtain a larger matrix as $\mathbf{A} = [\mathbf{A}_1 \mid \mathbf{A}_2]$. And, it sets the trapdoor $T_{\mathbf{A}}$ to be the combined trapdoor information $T_{\mathbf{A}} = (T_{\mathbf{A}_1}, T_{\mathbf{A}_2})$.

$\mathsf{EnSamplePre}(\mathbf{A}, T_{\mathbf{A}}, \mathbf{Z}, \sigma) \to \mathbf{U}$. The pre-image sampling algorithm takes as input $\mathbf{A} = [\mathbf{A}_1 \mid \mathbf{A}_2]$, trapdoor $T_{\mathbf{A}} = (T_{\mathbf{A}_1}, T_{\mathbf{A}_2})$, parameter σ and matrix $\mathbf{Z} \in \mathbb{Z}_q^{n \times k}$. It chooses a uniformly random matrix $\mathbf{W} \leftarrow \mathbb{Z}_q^{n \times k}$, and sets $\mathbf{Y} = \mathbf{Z} - \mathbf{W}$. Next, it computes matrices $\mathbf{U}_1 \in \mathbb{Z}_q^{\lceil m/2 \rceil \times k}, \mathbf{U}_2 \in \mathbb{Z}_q^{\lfloor m/2 \rfloor \times k}$ as

$$\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}),$$
$$\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y}).$$

Finally, it computes the final output matrix $\mathbf{U} \in \mathbb{Z}_q^{m \times k}$ by row-wise appending matrices $\mathbf{U}_1$ and $\mathbf{U}_2$. Concretely, $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$.

Correctness. Correctness follows directly from the correctness of LT.

7.3 Proving Security of $\mathsf{LT}_{\mathsf{en}}$

We will now prove that our enhanced trapdoor sampling scheme satisfies well-sampledness of preimage, row removal and target switching properties. First, we show that it satisfies preimage well-sampledness property if the underlying trapdoor scheme satisfies preimage well-sampledness property.

Theorem 7.1. Fix any functions $q : \mathbb{N} \to \mathbb{N}$ and $\sigma : \mathbb{N} \to \mathbb{R}^+$. If LT satisfies (q, σ)-well sampledness of preimage (Definition 2.3), then $\mathsf{LT}_{\mathsf{en}}$ also satisfies (q, σ)-well sampledness of preimage.

Proof Sketch. This follows directly from our construction. The well-sampledness of preimage requires that the pre-image of a uniformly random matrix $\mathbf{Z}$ looks like a Gaussian sample with parameter σ. In our construction, the pre-image of a random matrix $\mathbf{Z}$ consists of pre-images of $\mathbf{W}$ and $\mathbf{Z} - \mathbf{W}$, where $\mathbf{W}$ is uniformly random. Since $\mathbf{Z}$ is random, so is $\mathbf{Z} - \mathbf{W}$. As a result, using the well-sampledness of preimage property of LT, we can argue that these two preimages look like two matrices drawn from $\mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times k}$ and $\mathcal{D}_{\mathbb{Z},\sigma}^{\lfloor m/2 \rfloor \times k}$ respectively.

7.3.1 Row Removal Property

Now we prove that our trapdoor sampling scheme satisfies row removal property. Formally, we prove the following.

Theorem 7.2. Fix any functions $q : \mathbb{N} \to \mathbb{N}$ and $\sigma : \mathbb{N} \to \mathbb{R}^+$. If LT satisfies (q, σ)-well sampledness of preimage (Definition 2.3) and q-well sampledness of matrix (Definition 2.2), then $\mathsf{LT}_{\mathsf{en}}$ also satisfies the (q, σ)-single row removal property (Definition 7.1).

Proof. Our proof follows from a sequence of hybrid experiments. We start by defining a sequence of hybrid experiments such that the first and last experiments correspond to the original row removal security game when the challenger chooses its challenge bit b to be 0 and 1, respectively. To complete the proof we show that the adversary's advantage must be negligible between any two consecutive hybrids.

We now define hybrids H_x for $x \in \{0, 1, \ldots, 12\}$. In all the hybrid experiments below, we set $q = q(\lambda)$ and $\sigma = \sigma(\lambda)$. Also, below in each successive hybrid step, we only describe the modifications. Later in Section C.1, we provide the detailed hybrids.

Hybrid H_0 : This corresponds to the original game (as per Definition 7.1, with the single row removal restriction) with $b = 0$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $(\mathbf{B}_1, T_{\mathbf{B}_1}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lceil m/2 \rceil}, q)$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B} = [\mathbf{B}_1 \mid \mathbf{B}_2]$.
   (b) It also chooses a vector $\mathbf{p} \leftarrow \mathbb{Z}_q^m$, and sets matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ as $\mathbf{A} = \mathsf{Arrange}(\mathbf{B}, \mathbf{p}, [n] \setminus \{i\})$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1) \times t}$. The challenger responds to each query as follows.
   (a) It chooses $\mathbf{W} \leftarrow \mathbb{Z}_q^{(n-1) \times t}$ and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{B}_1, T_{\mathbf{B}_1}, \sigma, \mathbf{W})$.
   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ (which is equal to $\mathbf{C} - \mathbf{W}$), and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.
   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.
3. The adversary outputs a bit b'.

Hybrid H_1 : In this experiment, the challenger chooses $\mathbf{U}_1$ to be a random Gaussian matrix with parameter σ for each query.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1) \times t}$. The challenger responds to each query as follows.
   (a) It samples $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.

Hybrid H_2 : In this hybrid, the challenger chooses $\mathbf{B}_1$ uniformly at random, instead of choosing it using $\mathsf{TrapGen}$. At this point, note that the left half of $\mathbf{A}$ is a uniformly random matrix.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $\mathbf{B}_1 \leftarrow \mathbb{Z}_q^{(n-1) \times \lceil m/2 \rceil}$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B} = [\mathbf{B}_1 \mid \mathbf{B}_2]$.

Hybrid H_3 : This hybrid involves syntactic changes. The challenger chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times \lceil m/2 \rceil}$, and derives $\mathbf{B}_1$ by removing the i^{th} row of $\mathbf{A}_1$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times \lceil m/2 \rceil}$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \mid \mathbf{B}_2]$.
   (b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2 \rfloor}$, and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n] \setminus \{i\})$, $\mathbf{A} = [\mathbf{A}_1 \mid \mathbf{A}_2]$.

Hybrid H_4 : In this hybrid, the challenger chooses the left half of $\mathbf{A}$ using $\mathsf{TrapGen}$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.

   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

Hybrid H_5 : In this hybrid, the challenger chooses $\mathbf{U}_1$ using $\mathsf{SamplePre}$ for each query.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.

   (a) It chooses $\mathbf{W}' \leftarrow \mathbb{Z}_q^{n\times t}$, sets $\mathbf{W} = \mathsf{Restrict}(\mathbf{W}', [n]\setminus\{i\})$, and samples $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

Hybrid H_6 : This hybrid represents a syntactic change, in which the challenger, for each query, chooses $\mathbf{Y}$ as a uniformly random matrix, and set $\mathbf{W} = \mathbf{C} - \mathbf{Y} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.

   (a) It chooses $\mathbf{Y} \leftarrow \mathbb{Z}_q^{(n-1)\times t}$, and samples $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.

   (b) Next, it sets $\mathbf{W} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$ (which is equal to $\mathbf{C} - \mathbf{Y}$), chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{W}' = \mathsf{Arrange}(\mathbf{W}, \mathbf{w}, [n] \setminus \{i\})$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

Hybrid H_7 : In this hybrid experiment, the challenger chooses $\mathbf{U}_2$ from a Gaussian distribution with parameter σ.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.

   (a) It samples $\mathbf{U}_2 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lfloor m/2 \rfloor \times t}$.

Hybrid H_8 : In this hybrid, the challenger chooses matrix $\mathbf{B}_2$ uniformly at random. Note that this means $\mathbf{A}_2$ is uniformly random in this hybrid.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.

   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $\mathbf{B}_2 \leftarrow \mathbb{Z}_q^{(n-1)\times \lfloor m/2 \rfloor}$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

Hybrid H_9 : In this hybrid, the matrix $\mathbf{A}_2$ is chosen using $\mathsf{TrapGen}$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.

   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $\mathbf{A}_2 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lfloor m/2 \rfloor}, q\right)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

Hybrid H_{10} : In this hybrid, the challenger chooses $\mathbf{U}_2$ using $\mathsf{SamplePre}$ for each query.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.

   (a) It chooses $\mathbf{Y}' \leftarrow \mathbb{Z}_q^{n\times t}$ and samples $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y}')$.

Hybrid H_{11} : This hybrid represents a syntactic change in which the i^{th} row of matrix $\mathbf{W}'$ is set as a difference of random vector $\mathbf{c}$ and i^{th} row of $\mathbf{A}_2 \cdot \mathbf{U}_2$ instead of being sampled uniformly at random directly.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.

   (b) Next, it chooses a uniformly random vector $\mathbf{c} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{C}' = \mathsf{Arrange}(\mathbf{C}, \mathbf{c}, [n] \setminus \{i\})$, sets $\mathbf{W}' = \mathbf{C}' - \mathbf{A}_2 \cdot \mathbf{U}_2$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

Hybrid H_{12} : This hybrid represents a syntactic change. It corresponds to the security game in Definition 7.1 with $b = 1$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.

   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $\mathbf{A}_2 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lfloor m/2 \rfloor}, q\right)$.

   (b) It sets $\mathbf{A} = [\mathbf{A}_1 \mid \mathbf{A}_2]$.

   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.

   (a) It chooses $\mathbf{W}' \leftarrow \mathbb{Z}_q^{n\times t}$ and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W})$.

   (b) Next, it chooses a uniformly random vector $\mathbf{c} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{C}' = \mathsf{Arrange}(\mathbf{C}, \mathbf{c}, [n] \setminus \{i\})$, sets $\mathbf{Y}' = \mathbf{C}' - \mathbf{A}_1 \cdot \mathbf{U}_1$ (which is equal to $\mathbf{C}' - \mathbf{W}'$), and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y}')$.

   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Analysis. We will now show that any PPT adversary has at most negligible advantage in distinguishing any two consecutive hybrids. For any adversary $\mathcal{A}$, let $p_{\mathcal{A},i} : \mathbb{N} \to [0,1]$ denote the function such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},i}(\lambda)$ is the probability that $\mathcal{A}$, on input 1^λ, outputs 1 in hybrid experiment H_i. From the definition of the hybrid experiments, it follows that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0}(\lambda) - p_{\mathcal{A},12}(\lambda) = 2\mathsf{pr}_{\mathsf{LT}_{\mathsf{en}},\mathcal{A}}^{\mathrm{row-rem},q,\sigma}(\lambda) - 1$. Therefore, to show that $\mathsf{LT}_{\mathsf{en}}$ satisfies the (q,σ)-row removal property, it suffices to show that for all $\mathcal{A}$ and $i \in [12]$, there exist negligible functions negl_i such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},i-1}(\lambda) - p_{\mathcal{A},i}(\lambda) \le \mathrm{negl}_i(\lambda)$.

Lemma 7.1. Assuming LT satisfies (q,σ)-well sampledness of preimage, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}_1(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0}(\lambda) - p_{\mathcal{A},1}(\lambda) \le \mathrm{negl}_1(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ and a non-negligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0}(\lambda) - p_{\mathcal{A},1}(\lambda) \ge \eta(\lambda)$. Moreover, let $s_{\mathcal{A}}$ denote the number of queries made by $\mathcal{A}$, and let $t_{\mathcal{A}}$ denote a bound on the number of columns in queried matrix $\mathbf{C}$ (note that the reduction algorithm is allowed to depend on the adversary, therefore it knows $s_{\mathcal{A}}$ and $t_{\mathcal{A}}$ corresponding to $\mathcal{A}$).[22] Then we can construct a reduction algorithm $\mathcal{B}$ such that $\mathsf{pr}_{\mathsf{LT},\mathcal{B}}^{\mathrm{preimg},q,\sigma}(\lambda) \ge \eta(\lambda)$ for all $\lambda \in \mathbb{N}$.

The reduction algorithm receives n, m, index $i \in [n]$ from $\mathcal{A}$ such that $m > 2n\log q + 2\lambda$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda$. It forwards $1^{n-1}, 1^{\lceil m/2 \rceil}, 1^{s_{\mathcal{A}} \cdot t_{\mathcal{A}}}$ to the challenger.[23]It receives $\mathbf{B}_1 \in \mathbb{Z}_q^{(n-1)\times \lceil m/2 \rceil}$ and $\widetilde{\mathbf{U}} \in \mathbb{Z}_q^{\lceil m/2 \rceil \times (s_{\mathcal{A}} \cdot t_{\mathcal{A}})}$. Note that the trapdoor for $\mathbf{B}_1$ is not used in hybrid H_1. The reduction algorithm

[22]Througout this section, we construct non-uniform reduction algorithm as our reduction algorithms depends on the number of queries made by the adversary as well as size of the matrices in each query. However, we would like to point out that the reduction could be made uniform by simply guessing both these bounds. This would result in a polynomial loss in the reduction algorithm's advantage.

[23]Note that the reduction algorithm chooses admissible parameters, since $\lceil m/2 \rceil > n\log q + \lambda > (n-1)\log q + \lambda$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda > \sqrt{(n-1) \cdot \log q \cdot \log m/2} + \lambda$.

chooses $(\mathbf{B}_2, T_{\mathbf{B}_2})$ using $\mathsf{TrapGen}$, computes $\mathbf{A}$ as in H_0 (and H_1) and sends $\mathbf{A}$ to $\mathcal{A}$. The challenger also partitions $\widetilde{\mathbf{U}} = \left[\widetilde{\mathbf{U}}_1 \mid \ldots \mid \widetilde{\mathbf{U}}_{sA}\right]$, where each $\widetilde{\mathbf{U}}_j \in \mathbb{Z}_q^{\lceil m/2 \rceil \times t_{\mathcal{A}}}$.

Next, the adversary sends queries. For the i^{*th} query, the adversary sends $(1^t, \mathbf{C})$ for some $t \leq t_{\mathcal{A}}$. The reduction algorithm sets $\mathbf{U}_1$ to be the first t columns of $\widetilde{\mathbf{U}}_{i^*}$. It computes $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ and $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$. It sets $\mathbf{U}$ as in H_0/H_1 and sends $\mathbf{U}$ to $\mathcal{A}$.

Finally, after all queries, if $\mathcal{A}$ outputs 1, $\mathcal{B}$ guesses that $\widetilde{\mathbf{U}}$ is sampled using $\mathsf{SamplePre}$, else it guesses that $\widetilde{\mathbf{U}}$ is a random Gaussian matrix sampled with parameter σ.

Note that depending on whether $\widetilde{\mathbf{U}}$ is sampled using $\mathsf{SamplePre}$ or sampled from Gaussian distribution, $\mathcal{B}$ simulates either H_0 or H_1 perfectly. As a result, $B's$ advantage in the preimage well-sampledness game is at least $\eta(\lambda)$. ∎

Lemma 7.2. Assuming LT satisfies q-well sampledness of matrix, for any adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},1}(\lambda) - p_{\mathcal{A},2}(\lambda) \leq \text{negl}_2(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ and a non-negligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},1}(\lambda) - p_{\mathcal{A},2}(\lambda) \geq \eta(\lambda)$. Then we can construct a reduction algorithm $\mathcal{B}$ such that $\mathsf{pr}_{\mathsf{LT},\mathcal{B}}^{\text{matrix},q,\sigma}(\lambda) \geq \eta(\lambda)$ for all $\lambda \in \mathbb{N}$.

The reduction algorithm receives $1^n, 1^m$, index $i \in [n]$ from $\mathcal{A}$. It forwards $1^{n-1}, 1^{\lceil m/2 \rceil}$ to the challenger.[24] It receives $\mathbf{B}_1$. The reduction algorithm chooses $(\mathbf{B}_2, T_{\mathbf{B}_2})$ using $\mathsf{TrapGen}$, sets $\mathbf{A}$ as in H_1 (and H_2) and sends $\mathbf{A}$ to $\mathcal{A}$.

Next, the adversary sends queries. For each query $\mathbf{C}$, the challenger chooses $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$, computes $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ and $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$. It chooses $\mathbf{p}$, sets $\mathbf{U}$ as in H_1/H_2 and sends $\mathbf{U}$ to $\mathcal{A}$.

After all the queries, if $\mathcal{A}$ outputs 1, $\mathcal{B}$ guesses that $\mathbf{B}_1$ is sampled using $\mathsf{TrapGen}$, else it guesses that $\mathbf{B}_1$ is a uniformly random matrix. Note that depending on whether $\mathbf{B}_1$ is sampled using $\mathsf{TrapGen}$ or sampled uniformly at random, $\mathcal{B}$ simulates either H_1 or H_2 perfectly. As a result, $B's$ advantage in the matrix well-sampledness game is at least $\eta(\lambda)$.

∎

Lemma 7.3. For any adversary $\mathcal{A}$, $p_{\mathcal{A},2}(\lambda) = p_{\mathcal{A},3}(\lambda)$.

Since the only changes from H_2 to H_3 are syntactical, it follows that any adversary has identical behavior in both hybrids.

Lemma 7.4. Assuming LT satisfies q-well sampledness of matrix, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_4(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},3}(\lambda) - p_{\mathcal{A},4}(\lambda) \leq \text{negl}_4(\lambda)$.

This proof is identical to the proof of Lemma 7.2, except that the reduction algorithm must send $1^n, 1^{\lceil m/2 \rceil}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}$.

Lemma 7.5. Assuming LT satisfies (q, σ)-well sampledness of pre-image, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_5(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},4}(\lambda) - p_{\mathcal{A},5}(\lambda) \leq \text{negl}_5(\lambda)$.

This proof is identical to the proof of Lemma 7.1, except that the reduction algorithm must send $1^n, 1^{\lceil m/2 \rceil}, 1^{s_{\mathcal{A}} \cdot t_{\mathcal{A}}}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}, 1^{s_{\mathcal{A}} \cdot t_{\mathcal{A}}}$.

Lemma 7.6. For any adversary $\mathcal{A}$, $p_{\mathcal{A},5}(\lambda) = p_{\mathcal{A},6}(\lambda)$.

Note that the distributions in H_5 and H_6 are identical. In hybrid H_5, the challenger chooses $\mathbf{W}' \leftarrow \mathbb{Z}_q^{n \times m}$, derives $\mathbf{W}$ from $\mathbf{W}'$ by removing the i^{th} row and sets $\mathbf{Y} = \mathbf{C} - \mathbf{W}$. In hybrid H_6, it chooses $\mathbf{Y} \leftarrow \mathbb{Z}_q^{(n-1) \times m}$, sets $\mathbf{W} = \mathbf{C} - \mathbf{Y}$, and $\mathbf{W}'$ to be a matrix extended from $\mathbf{W}$ by inserting a random vector at row i. The distribution of $(\mathbf{W}, \mathbf{W}', \mathbf{Y})$ is identical in both the hybrid experiments.

[24]Note that the reduction algorithm chooses admissible parameters, since $\lceil m/2 \rceil > n \log q + \lambda > (n-1) \log q + \lambda$.

Lemma 7.7. Assuming LT satisfies (q, σ)-well sampledness of pre-image, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_7(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},6}(\lambda) - p_{\mathcal{A},7}(\lambda) \leq \text{negl}_7(\lambda)$.

This proof is identical to the proof of Lemma 7.1, except that the reduction algorithm must send $1^{n-1}, 1^{\lfloor m/2 \rfloor}, 1^{s_{\mathcal{A}} \cdot t_{\mathcal{A}}}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}, 1^{s_{\mathcal{A}} \cdot t_{\mathcal{A}}}$. It uses the challenger's response for setting $\mathbf{B}_2, \mathbf{U}_2$, and chooses the remaining components by itself.

Lemma 7.8. Assuming LT satisfies q-well sampledness of matrix, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_8(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},7}(\lambda) - p_{\mathcal{A},8}(\lambda) \leq \text{negl}_8(\lambda)$.

This proof is identical to the proof of Lemma 7.2, except that the reduction algorithm must send $1^{n-1}, 1^{\lfloor m/2 \rfloor}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}$. It uses the challenger's response for setting $\mathbf{B}_2$, and chooses the remaining components by itself.

Lemma 7.9. Assuming LT satisfies q-well sampledness of matrix, for any adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_9(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},8}(\lambda) - p_{\mathcal{A},9}(\lambda) \leq \text{negl}_9(\lambda)$.

This proof is identical to the proof of Lemma 7.2, except that the reduction algorithm must send $1^{n}, 1^{\lfloor m/2 \rfloor}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}$. It uses the challenger's response for setting $\mathbf{A}_2$, and chooses the remaining components by itself.

Lemma 7.10. Assuming LT satisfies (q, σ)-well sampledness of pre-image, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_{10}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},9}(\lambda) - p_{\mathcal{A},10}(\lambda) \leq \text{negl}_{10}(\lambda)$.

This proof is identical to the proof of Lemma 7.1, except that the reduction algorithm must send $1^{n}, 1^{\lfloor m/2 \rfloor}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}$. It uses the challenger's response for setting $\mathbf{A}_2, \mathbf{U}_2$, and chooses the remaining components by itself.

Lemma 7.11. For any adversary $\mathcal{A}$, $p_{\mathcal{A},10}(\lambda) = p_{\mathcal{A},11}(\lambda)$.

In hybrid experiment H_{10}, the challenger chooses $\mathbf{Y}' \leftarrow \mathbb{Z}_q^{n \times m}$, derives $\mathbf{Y}'$ by removing the i^{th} row. It sets $\mathbf{W} = \mathbf{C} - \mathbf{Y}$, chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^m$ and constructs $\mathbf{W}'$ from $\mathbf{W}$ and $\mathbf{w}$. In hybrid H_{11}, the challenger chooses $\mathbf{Y}'$ uniformly at random, extends $\mathbf{C}'$ from $\mathbf{C}$ by inserting a random vector at i^{th} row, and sets $\mathbf{W}' = \mathbf{C}' - \mathbf{Y}'$. As a result, $(\mathbf{Y}', \mathbf{W}')$ are identically distributed in both hybrids. The remaining components in the hybrids are either identical, or can be derived from $\mathbf{Y}', \mathbf{W}'$.

Lemma 7.12. For any adversary $\mathcal{A}$, $p_{\mathcal{A},11}(\lambda) = p_{\mathcal{A},12}(\lambda)$.

Note that the distributions in $H_1 1$ and $H_1 2$ are identical. The proof is identical to that of Lemma 7.6.

Using the above lemmas, it follows that the advantage of an adversary in the row removal experiment is at most $\text{negl}(\lambda)$.

■

Theorem 7.3. Fix any function $q : \mathbb{N} \to \mathbb{N}$, $\sigma : \mathbb{N} \to \mathbb{R}^+$. Assuming $\mathsf{LT}_{\mathsf{en}} = (\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ satisfies the (q, σ)-single row removal property, $\mathsf{LT}_{\mathsf{en}}$ also satisfies the (q, σ)-row removal property.

Proof. This proof follows via a simple hybrid argument, where we gradually remove the non-targeted rows from $\mathbf{A}$ one by one. Suppose $\mathcal{A}$ outputs set S of size k. We will define $n - k + 1$ hybrids $H_1, \ldots, H_{n-k+1}$ as follows.

Hybrid H_i for $0 \leq i \leq n-k$ In hybrid H_i, the challenger does the following.

1. Let $(1^n, 1^m, S \subseteq [n]) \leftarrow \mathcal{A}(1^\lambda)$, and let $S = \{i_1, \ldots, i_k\}$, $\overline{S} = \{i'_1, \ldots, i'_{n-k}\}$. Let $S_i = S \cup \{i'_1, \ldots, i'_i\} = \{\tilde{i}_1, \ldots, \tilde{i}_{k+i}\}$ and $\overline{S_i} = [n] \setminus S_i = \{\tilde{i'}_1, \ldots, \tilde{i'}_{n-k-i}\}$.
2. It chooses $(\mathbf{B}, T_{\mathbf{B}}) \leftarrow \mathsf{EnTrapGen}(1^{k+i}, 1^m, q)$, $\mathbf{P} \leftarrow \mathbb{Z}_q^{(n-k-i)\times m}$.
3. It sets $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ where $\mathbf{A}[\tilde{i}_j] = \mathbf{B}[j]$ for all $j \leq k+i$, and $\mathbf{A}[\tilde{i'}_j] = \mathbf{P}[j]$ for all $j \leq n-k-i$, and sends $\mathbf{A}$ to $\mathcal{A}$.
4. Next, the adversary sends queries. For each query $\{\mathbf{c}_j\}_{j\in S}$, the challenger first chooses $\mathbf{c}_j \leftarrow \mathbb{Z}_q^t$ for each $j \in S_i \setminus S$, sets $\mathbf{C} \in \mathbb{Z}_q^{(k+i)\times t}$ where $\mathbf{C}[j] = \mathbf{c}_{\tilde{i}_j}$ for all $j \leq k+i$.
5. Next, it chooses $\mathbf{U} \leftarrow \mathsf{EnSamplePre}(\mathbf{B}, T_{\mathbf{B}}, \mathbf{C}, \sigma)$ and sends $(\mathbf{A}, \mathbf{U})$.
6. Finally, after all queries, the adversary $\mathcal{A}$ outputs a bit b'.

Note that H_0 and H_{n-k} correspond to $b = 0$ and $b = 1$ in the row removal experiment. For any adversary $\mathcal{A}$, let $p_{\mathcal{A},i}(\lambda)$ denote the probability of $\mathcal{A}$ outputting 1 in hybrid H_i. We will show that for any adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that $p_{\mathcal{A},i}(\lambda)$ and $p_{\mathcal{A},i+1}$ differ by at most $\text{negl}(\lambda)$.

Lemma 7.13. Fix any index $i \in \{0, 1, \ldots, n-k-1\}$. Assuming $\mathsf{LT}_{\mathsf{en}}$ satisfies the single row removal property, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},i}(\lambda) - p_{\mathcal{A},i+1}(\lambda) \leq \text{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ and a non-negligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},i}(\lambda) - p_{\mathcal{A},i+1}(\lambda) \geq \eta(\lambda)$. We can use $\mathcal{A}$ to build a reduction algorithm $\mathcal{B}$ that can break the single row removal property with advantage $\eta(\cdot)$.

The reduction algorithm first receives $1^n, 1^m, S = \{i_1, \ldots, i_k\}$. It defines $S_{i+1} = S \cup \{i'_1, \ldots, i'_{i+1}\} = \{\tilde{i}_1, \ldots, \tilde{i}_{k+i+1}\}$, and let $\mathsf{indx} \in [k+i+1]$ be the index such that $\tilde{i}_{\mathsf{indx}} = i'_{i+1}$. The reduction algorithm sends $1^{k+i+1}, 1^m$ and indx.[25] It receives $\mathbf{B}$ from the challenger. The reduction algorithm sets $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ such that for each $j \leq k+i+1$, $\mathbf{A}[\tilde{i}_j] = \mathbf{B}[j]$, and the remaining rows are chosen uniformly at random.

For each query $\{\mathbf{c}_j\}_{j\in S}$, the reduction algorithm chooses vectors $\{\mathbf{c}_{i'_1}, \ldots, \mathbf{c}_{i'_i}\}$ uniformly at random from $\mathbb{Z}_q^t$ and sends $\{\mathbf{c}_{\tilde{i}_j}\}_{j\in[k+i+1], j\neq \mathsf{indx}}$ to the challenger. The challenger sends $\mathbf{U}$ to the reduction algorithm. The reduction algorithm forwards $\mathbf{U}$ to $\mathcal{A}$. Finally, after all the queries, $\mathcal{B}$ outputs the adversary's final output bit.

Clearly, the reduction algorithm simulates either H_i or H_{i+1} depending on the challenger's output, and therefore the advantage of $\mathcal{B}$ is $p_{\mathcal{A},i}(\lambda) - p_{\mathcal{A},i+1}(\lambda)$. ■

■

7.3.2 Target Switching Property

Now we prove that our trapdoor sampling scheme satisfies target switching property. Formally, we prove the following.

Theorem 7.4. Fix any functions $q : \mathbb{N} \to \mathbb{N}$ and $\sigma : \mathbb{N} \to \mathbb{R}^+$, and error distribution family $\{\chi(\lambda)\}_\lambda$. If LT satisfies q-well distributedness of matrix (Definition 2.2), (q, σ)-well distributedness of preimage (Definition 2.3), and $\mathsf{LWE\text{-}sp}_{(d,q,\sigma,\chi)}$ holds (LWE with short public vectors : Assumption 3) where $d(\lambda) = 6\lambda \log q(\lambda)$, then $\mathsf{LT}_{\mathsf{en}}$ satisfies (q, σ, χ)-single target switching property.

[25]Note that the reduction algorithm chooses admissible parameters, since $m > n\log q + \lambda > (k+i+1)\log q + \lambda$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda > \sqrt{(k+i+1) \cdot \log q \cdot \log m} + \lambda$.

Proof. To prove the above theorem, we first define a sequence of hybrid games where the first game is the single target switching security game, and in the last game the adversary's advantage is exactly 0. Later we show that the adversary's advantage in any two consecutive hybrid games is negligible. For simplicity of notation, we will let $d = d(\lambda)$, $q = q(\lambda), \sigma = \sigma(\lambda)$ and $\chi = \chi(\lambda)$. Below in each successive hybrid game, we only describe the modifications. Later in Section C.2, we provide the detailed hybrid games.

Hybrid H_0 : This corresponds to the single target switching security game.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It chooses $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lceil m/2 \rceil}, q)$ and $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0,1\}$.
   (b) Next, it sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and sends $[\mathbf{B}_1 \,|\, \mathbf{B}_2]$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$ where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0, [n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows.
   (a) It chooses $\mathbf{W} \leftarrow \mathbb{Z}_q^{n \times t}$, computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W})$.
   (b) It also samples vector $\mathbf{e} \leftarrow \chi^t$, and sets $\mathbf{E} = \mathsf{Arrange}(\mathbf{0}^{(n-1) \times t}, \mathbf{e}, [n] \setminus \{i\})$.
   (c) Next, it sets $\mathbf{Y} = \mathbf{Z}_b - \mathbf{A}_1 \cdot \mathbf{U}_1 + \mathbf{E}$ (which is equal to $\mathbf{Z}_b - \mathbf{W} + \mathbf{E}$), and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.
   (d) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.
3. $\mathcal{A}$ outputs its guess b'.

Hybrid H_1 : In this hybrid experiment, the challenger sets $\mathbf{U}_1$ to be a Gaussian matrix for each query.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$ where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0, [n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows.
   (a) It computes $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.

Hybrid H_2 : In this hybrid experiment, the challenger sets $\mathbf{A}_1$ to be a uniformly random matrix (that is, sampled without a trapdoor).

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times \lceil m/2 \rceil}$ and $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0,1\}$.

Hybrid H_3 : This hybrid is a syntactic change. Here, we express $\mathbf{Y}$ in terms of $\mathbf{B}_1$ and the i^{th} row of $\mathbf{A}_1$. Note that the i^{th} row of $\mathbf{A}_1$ is used only for computing the i^{th} row of $\mathbf{Y}$.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$ where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0, [n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows.
   (b) It also samples vector $\mathbf{e} \leftarrow \chi^t$, and sets $\mathbf{Z}'_b = \mathsf{Restrict}(\mathbf{Z}_b, [n] \setminus \{i\})$.
   (c) Next, it sets $\mathbf{Y}' = \mathbf{Z}'_b - \mathbf{B}_1 \cdot \mathbf{U}_1$, $\mathbf{y} = \mathbf{Z}_b[i] - \mathbf{A}_1[i] \cdot \mathbf{U}_1 + \mathbf{e}$, and $\mathbf{Y} = \mathsf{Arrange}(\mathbf{Y}', \mathbf{y}, [n] \setminus \{i\})$. It then computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.

Hybrid H_4 : In this hybrid experiment, the challenger sets the i^{th} row of $\mathbf{Y}$ to be a uniformly random vector.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$ where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0, [n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows.

   (c) Next, it sets $\mathbf{Y}' = \mathbf{Z}_b' - \mathbf{B}_1 \cdot \mathbf{U}_1$, $\mathbf{y} \leftarrow \mathbb{Z}_q^t$, and $\mathbf{Y} = \mathsf{Arrange}(\mathbf{Y}', \mathbf{y}, [n] \setminus \{i\})$. It then computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.

Analysis. We will now analyse the adversary's advantage in the single target switching experiment. Let $p_{\mathcal{A},l}(\lambda)$ denote the probability of $\mathcal{A}$ guessing correctly (i.e. $b' = b$) at the end of hybrid experiment H_l. We will show that for every PPT adversary $\mathcal{A}$ and $l \in [4]$ there exist negligible functions $\text{negl}_l(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},l-1} - p_{\mathcal{A},l} \leq \text{negl}_i(\lambda)$.

Lemma 7.14. Assuming LT satisfies (q, σ)-well sampledness of preimage, for any adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_1(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0} - p_{\mathcal{A},1} \leq \text{negl}_1(\lambda)$.

The proof of this lemma is similar to the proof of Lemma 7.1.

Lemma 7.15. Assuming LT satisfies q-well sampledness of matrix, for any adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},1} - p_{\mathcal{A},2} \leq \text{negl}_2(\lambda)$.

The proof of this lemma is similar to the proof of Lemma 7.2.

Lemma 7.16. For any adversary $\mathcal{A}$, $p_{\mathcal{A},2} = p_{\mathcal{A},3}$.

Note that the only differences in H_2 and H_3 are syntactic changes with respect to matrix $\mathbf{Y}$. As a result, the distributions in the two hybrids are identical.

Lemma 7.17. If $\mathsf{LWE\text{-}sp}_{(d,q,\sigma,\chi)}$ holds (Assumption 3) where $d(\lambda) = 6\lambda \log q(\lambda)$, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_4(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},3} - p_{\mathcal{A},4} \leq \text{negl}_4(\lambda)$.

Proof. Suppose, on the contrary, there exists an adversary $\mathcal{A}$ and a non-negligible function $\eta(\cdot)$ such that $p_{\mathcal{A},3} - p_{\mathcal{A},4} \geq \eta(\lambda)$ for all $\lambda \in \mathbb{N}$. We will construct a reduction algorithm $\mathcal{B}$ such that $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE\text{-}sp},d,q,\sigma,\chi}(\lambda) \geq \eta(\lambda)$ for all $\lambda \in \mathbb{N}$.

The reduction algorithm first receives $1^n, 1^m, i \in [n], \mathbf{Z}_0, \mathbf{Z}_1$ from the adversary $\mathcal{A}$ (such that $m > 2n \log q + 12\lambda \log q$). The reduction algorithm chooses $\mathbf{B}_1 \leftarrow \mathbb{Z}_q^{(n-1)\times m}$, $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$ and derives $\mathbf{B}_2$ from $\mathbf{A}_2$ by removing i^{th} row. It defines $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$ and sends it to the adversary. It also chooses a random bit $b \leftarrow \{0, 1\}$.

Next, the reduction algorithm receives queries from the adversary, and it uses the $\mathsf{LWE\text{-}sp}$ challenger to define matrices $\mathbf{U}_1$ and $\mathbf{y}$. For each query, the adversary sends two matrices $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n\times t}$ such that all their rows are equal, except the i^{th} one. The reduction algorithm queries the $\mathsf{LWE\text{-}sp}$ challenger for t queries, and receives $\{(\mathbf{a}_r, u_r)\}_{r\in[t]}$, where $\mathbf{a}_r \in \mathbb{Z}_q^d$ for each $r \in [t]$. It chooses $\tilde{\mathbf{a}}_r \leftarrow \mathcal{D}_{\mathbb{Z}_q,\sigma}^{\lceil m/2 \rceil - d}$ for each $r \in [m]$, $\tilde{\mathbf{s}} \leftarrow \mathbb{Z}_q^{\lceil m/2 \rceil - d}$.[26] Next, it sets $\mathbf{U}_1 \in \mathbb{Z}_q^{\lceil m/2 \rceil \times m}$ to be a matrix whose r^{th} column is $[\mathbf{a} \,|\, \tilde{\mathbf{a}}_r]^T$ for each $r \in [m]$. It sets $\mathbf{y} \in \mathbb{Z}_q^m$, where $\mathbf{y}_r = \mathbf{Z}_b[i]_r - u_r - \tilde{\mathbf{s}} \cdot \tilde{\mathbf{a}}_r^T$.

Once $\mathbf{y}$ and $\mathbf{U}_1$ are determined, the reduction algorithm can compute $\mathbf{U}_2$ using $\mathbf{B}_1, \mathbf{U}_1, \mathbf{Z}_b, T_{\mathbf{A}_2}$. It sets $\mathbf{B}$ and $\mathbf{U}$ as in H_3/H_4 and sends $\mathbf{U}$ to $\mathcal{A}$.

Finally, after all the queries, the adversary outputs a bit b'. If $b = b'$, the reduction algorithm guesses 0 (i.e., u_r is an LWE sample), otherwise it guesses 1 (i.e., u_r is a uniformly random element).

[26]Since $m > 2n \log q + 12\lambda \cdot \log q$ and $d = 6\lambda \log q$, thus $\lceil m/2 \rceil - d \geq 0$. Here we would like to point out that in our target switching security game the adversary is allowed to choose the dimensions m, n; whereas as stated in our LWE assumption framework, the lattice dimensions are not chosen by the adversary. Due to this definitional inconsistency, as a reduction algorithm we always choose to attack the LWE problem for dimensions $d(\lambda) = 6\lambda \log q(\lambda)$. This could be avoided by adapting the existing definitions.

Now note thatI if the LWE-sp challenger uses oracle $O_2()$, then the reduction algorithm simulates H_4, else it simulates H_3. Therefore, the advantage of $\mathcal{B}$ is at least $p_{\mathcal{A},3} - p_{\mathcal{A},4}$. ■

Using the above lemmas, it follows that any PPT adversary has advantage at most $\text{negl}(\lambda)$ in the single target switching security game as in the last hybrid game (H_4), the challenger's response is independent of bit b and thus any adversary advantage is exactly 0.

■

Theorem 7.5. Fix any function $q : \mathbb{N} \to \mathbb{N}$, $\sigma : \mathbb{N} \to \mathbb{R}^+$ and distribution family $\{\chi(\lambda)\}_\lambda$. Assuming $\mathsf{LT_{en}}$ satisfies (q, σ, χ)-single target switching property, $\mathsf{LT_{en}}$ also satisfies (q, σ, χ)-target switching property.

The proof of this theorem follows from a hybrid argument similar to that in proof of Theorem 7.3.

8 Constructing 1-bounded Mixed Functional Encryption

In this section we describe our construction of mixed FE for input-circling branching programs with polynomial width and length. Concretely, $\mathcal{M}_\kappa = \{0,1\}^k$ and $\mathcal{F}_\kappa$ denotes the class of input-circling branching programs with input space $\{0,1\}^k$, width w and length $\ell = k \cdot L$, where $\kappa = (k, w, L)$. In other words, every branching program reads each input bit L times in a circular fashion. Before describing our construction, we introduce some shorthand notation that we will use throughout this section.

8.1 Notation

Consider a set of $4\ell\lambda$ matrices $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i\in[\ell],j\in[\lambda],\beta,b\in\{0,1\}}$ and $w\ell$ matrices $\{\mathbf{P}_{i,v}\}_{i\in[\ell],v\in[w]}$, where each individual matrix lies in $\mathbb{Z}_q^{n\times m}$. For $i \in [\ell]$, let $\mathbf{D}_i$ be another matrix defined as below:

$$\mathbf{D}_i = \begin{bmatrix} \mathbf{B}_{i,0}^{(1,0)} \\ \mathbf{B}_{i,1}^{(1,0)} \\ \mathbf{B}_{i,0}^{(1,1)} \\ \vdots \\ \mathbf{B}_{i,1}^{(\lambda,1)} \\ \mathbf{P}_{i,1} \\ \vdots \\ \mathbf{P}_{i,w} \end{bmatrix}$$

The matrix $\mathbf{D}_i$ consists of matrices $\mathbf{B}_{i,b}^{(j,\beta)}$ arranged as per adjoining well-defined ordering. Concretely, let (i, j, β, b) be the indices of any $\mathbf{B}$ matrix. The ordering we define is that

$$(i, j_1, \beta_1, b_1) \prec (i, j_2, \beta_2, b_2) \iff \begin{cases} j_1 < j_2, \text{ or} \\ j_1 = j_2 \wedge \beta_1 < \beta_2, \text{ or} \\ j_1 = j_2 \wedge \beta_1 = \beta_2 \wedge b_1 < b_2. \end{cases}$$

Thus, as per our ordering $(i,1,0,0), (i,1,0,1), (i,1,1,0), (i,1,1,1), \ldots, (i,\lambda,1,1)$ is an increasing sequence of indices. Similarly, we can define an ordering for matrices $\mathbf{P}_{i,v}$ for $v \in [w]$ (i.e., $(i, v_1) \prec (i, v_2) \iff v_1 < v_2$).

In words, the matrix $\mathbf{D}_i$ is defined by row-wise appending matrices $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{j\in[\lambda],\beta,b\in\{0,1\}}$ and $\{\mathbf{P}_{i,v}\}_{v\in[w]}$ in an increasing order as per the ordering '$\prec$' defined above. We will use the following shorthand notation for representing the above matrix more compactly.

$$\mathbf{D}_i = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{j\in[\lambda],\beta,b\in\{0,1\}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}$$

Similarly, for any (possibly empty) sets $S_1 \subseteq [\lambda] \times \{0,1\}^2, S_2 \subseteq [w]$, we will use the following shorthand

$$\mathbf{D}_i^{S_1,S_2} = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S_1} \\ \{\mathbf{P}_{i,v}\}_{v\in S_2} \end{bmatrix}$$

to represent the matrix generated by row-wise appending matrices $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S_1}$ and $\{\mathbf{P}_{i,v}\}_{v\in S_2}$ in an increasing order as per the ordering '$\prec$'.

8.2 Construction

In this section, we present our Mixed FE scheme. First, we provide the parameter constraints required by our correctness and security proof. For functionality indices (k, w, L) (where k denotes the input length, and w, ℓ $(= k \cdot L)$ is the width and length of branching programs), the setup algorithm chooses parameters n, m, q, σ, and noise distributions $\chi_{\mathsf{big}}, \chi_{\mathsf{last}}, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_s, \chi_{\mathsf{lwe}}$ as follows.

Fix any $\epsilon < 1/2$. Let χ_1 be a B_1-bounded discrete Gaussian distribution with parameter σ such that $B_1 = \sqrt{m}\cdot\sigma$, and χ_2 be a B_2-bounded discrete Gaussian distribution with parameter $\sqrt{2}\cdot\sigma$ such that $B_2 = \sqrt{2m}\cdot\sigma$. Also, for any $B > 0$, let U_B denote the uniform distribution on $\mathbb{Z} \cap [-B, B]$, i.e. integers between $\pm B$. The setup algorithm chooses parameters n, m, σ, q and sets noise distributions $\chi_{\mathsf{big}}, \chi_{\mathsf{last}}, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_s, \chi_{\mathsf{lwe}}$ with the following constraints:

- $n = \mathsf{poly}(\lambda)$, $\chi_{\mathsf{lwe}} = \chi_1$, $\chi_s = \chi_2$ and $q \le 2^{n^\epsilon}$ (for LWE security)
- $m > 2(4\lambda + w)\cdot n \cdot \log q + 12\lambda \cdot \log q$ (for Enhanced Trapdoor Sampling)
- $\sigma > \sqrt{(4\lambda + w)\cdot n \cdot \log q \cdot \log m} + \lambda$ (for Enhanced Trapdoor Sampling)
- $\chi_{\mathsf{pre}} = \chi_1$, $\chi_{\mathsf{appr}} = \chi_1$ (for Enhanced Trapdoor Sampling)
- $\chi_{\mathsf{big}} = U_{\sigma_{\mathsf{big}}}$ and $\chi_{\mathsf{last}} = U_{\sigma_{\mathsf{last}}}$ where $\sigma_{\mathsf{big}} = \sigma \cdot 2^\lambda, \sigma_{\mathsf{last}} = (m\cdot\sigma)^\ell \cdot 2^\lambda$ (for smudging/security)
- $(m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}}))^\ell \cdot (m \cdot (\sigma_{\mathsf{last}} + \sigma_{\mathsf{pre}})) \le q/16$ (for correctness)
- $\sqrt{n}\cdot\sigma_s \cdot (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^\ell \le q/16$ (for correctness)

First, note that it is not necessary to have distributions $\chi_{\mathsf{lwe}}, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}$ to be the same distribution. Keeping all these to be different distributions will only affect the underlying assumptions to which we reduce security. Also, one could also set χ_s to be χ_1 if the LWE modulus is a prime power. One possible setting of parameters is as follows: $n = (2\lambda\cdot\ell)^{1/\epsilon}$, $m = n^{1+2\epsilon}\cdot w$, $q = 2^{n^\epsilon}$, and $\sigma = n \cdot \sqrt{w}$.

We will now describe our Mixed FE construction.

- $\mathsf{Setup}(1^\lambda, (1^k, 1^w, 1^L)) \to (\mathsf{pp}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ, message length k, branching program width w and number of times it reads each bit L.[27] It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}$ as described above. Let $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It runs the $\mathsf{EnTrapGen}$ algorithm ℓ times as follows:

$$\forall\ i \in [\ell], \quad (\mathbf{M}_i, T_i) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$

For each $i \in [\ell]$, it interprets matrix $\mathbf{M}_i$ as $4\lambda + w$ matrices with dimensions $n \times m$ arranged as follows:

$$\mathbf{M}_i = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{j\in[\lambda],\beta,b\in\{0,1\}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}$$

Also, it samples $4\ell\lambda$ matrices $\left\{\mathbf{C}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}$ uniformly at random as $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0, 1\}$. Finally, it sets the public parameters and the master secret key as

$$\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}}), \quad \mathsf{msk} = \left(\left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{i\in[\ell],j\in[\lambda],\beta,b\in\{0,1\}}, \{\mathbf{P}_{i,v}\}_{i\in[\ell],v\in[w]}, \{T_i\}_{i\in[\ell]}\right).$$

[27] Note that here we slightly deviate from our definition as here we have 3 separate functionality parameters instead of a single index. This could simply be handled by extending the mixed FE definition to multiple indices.

- $\mathsf{KeyGen}(\mathsf{msk}, x \in \{0,1\}^k) \to \mathsf{sk}$. The key generation algorithm takes as input the master secret key msk and a message $x \in \{0,1\}^k$. Let

$$\mathsf{msk} = \left(\left\{ \mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)} \right\}_{i\in[\ell],j\in[\lambda],\beta,b\in\{0,1\}}, \{\mathbf{P}_{i,v}\}_{i\in[\ell],v\in[w]}, \{T_i\}_{i\in[\ell]} \right).$$

It chooses a secret vector $\widetilde{\mathbf{s}}$ of length n as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors $\mathbf{y}^{(j)}$ of length m as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda - 1]$. Next, it sets vector $\mathbf{y}^{(\lambda)}$ as

$$\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda-1]} \mathbf{y}^{(j)}.$$

The key generation algorithm then chooses $2\ell\lambda$ secret vectors $\left\{ \mathbf{s}_i^{(j,\beta)} \right\}_{i,j,\beta}$ and $2(\ell+1)\lambda$ error vectors $\left\{ \mathbf{e}_i^{(j,\beta)} \right\}_{i,j,\beta}$ of length n and m (respectively) as

$$\begin{aligned} &\forall\, i \in [\ell], j \in [\lambda], \beta \in \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\, i \in [\ell], j \in [\lambda], \beta \in \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\, j \in [\lambda], \beta \in \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m. \end{aligned}$$

Let $\widetilde{x} = x^L$, i.e. $\widetilde{x}$ is a $k \cdot L$-bit string obtained by appending string x to itself L times. Next, it computes $2(\ell+1)\lambda$ key vectors $\left\{ \mathbf{t}_i^{(j,\beta)} \right\}_{i,j,\beta}$ as follows.

$$\forall\, i \in [\ell+1], j \in [\lambda], \beta \in \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

Finally, it outputs the secret key sk as

$$\mathsf{sk} = \left(x, \left\{ \mathbf{t}_i^{(j,\beta)} \right\}_{i\in[\ell+1],j\in[\lambda],\beta\in\{0,1\}} \right).$$

- $\mathsf{Enc}(\mathsf{pp}) \to \mathsf{ct}$. The encryption algorithm takes as input the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$. It first chooses a λ-bit string $\mathsf{tag} \leftarrow \{0,1\}^\lambda$ and 2ℓ random short matrices $\{\mathbf{U}_{i,b}\}_{i,b}$ as

$$\forall\, i \in [\ell], b \in \{0,1\}, \quad \mathbf{U}_{i,b} \leftarrow \chi_{\mathsf{pre}}^{m\times m}.$$

Finally, it outputs the ciphertext ct as

$$\mathsf{ct} = \left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}} \right).$$

- $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \mathsf{BP}) \to \mathsf{ct}$. The secret key encryption algorithm takes as input the master secret key msk and an input-circling branching program BP. Let

$$\mathsf{msk} = \left(\left\{ \mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)} \right\}_{i\in[\ell],j\in[\lambda],\beta,b\in\{0,1\}}, \{\mathbf{P}_{i,v}\}_{i\in[\ell],v\in[w]}, \{T_i\}_{i\in[\ell]} \right),$$
$$\mathsf{BP} = \left(\{\pi_{i,b} : [w] \to [w]\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc} \in [w], \mathsf{rej} \in [w] \right).$$

It first chooses a λ-bit string $\mathsf{tag} \leftarrow \{0,1\}^\lambda$ and samples $8\ell\lambda$ matrices $\left\{\mathbf{D}_{i,b}^{(j,\beta)}, \widetilde{\mathbf{D}}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$ as follows:

$$\forall\ i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}, \qquad \begin{array}{l} \mathbf{D}_{i,b}^{(j,\beta)} = \begin{cases} \mathbf{C}_{i,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 0 \\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise.} \end{cases} \\ \widetilde{\mathbf{D}}_{i,b}^{(j,\beta)} = \begin{cases} \mathbf{C}_{i,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 1 \\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise.} \end{cases} \end{array}$$

In the above cases, we use '$\left(\leftarrow \mathbb{Z}_q^{n\times m}\right)$' to denote the operation of uniformly sampling a dimension $n \times m$ matrix in $\mathbb{Z}_q$. Note that here the sampling is performed *uniformly and independently each time*. Next, the algorithm samples w matrices $\{\mathbf{P}_{\ell+1,v}\}_{v\in[w]}$ for the top level, and $2w\ell$ error matrices $\left\{\mathbf{E}_{i,v}, \widetilde{\mathbf{E}}_{i,v}\right\}_{i\in[\ell],v\in[w]}$ as follows:

$$\forall\ v \in [w], \quad \mathbf{P}_{\ell+1,v} = \begin{cases} \mathbf{0}^{n\times m} & \text{if } v = \mathsf{rej} \\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise.} \end{cases}$$
$$\forall\ i \in [\ell], v \in [w], \quad \mathbf{E}_{i,v} \leftarrow \chi_{\mathsf{appr}}^{n\times m}, \widetilde{\mathbf{E}}_{i,v} \leftarrow \chi_{\mathsf{appr}}^{n\times m}.$$

The algorithm then sets $2w\ell$ matrices $\left\{\mathbf{Q}_{i,v}, \widetilde{\mathbf{Q}}_{i,v}\right\}_{i\in[\ell],v\in[w]}$ as follows:

$$\forall\ i \in [\ell], v \in [w], \qquad \begin{array}{l} \mathbf{Q}_{i,v} = \mathbf{P}_{i+1,\pi_{i,0}(v)} + \mathbf{E}_{i,v}, \\ \widetilde{\mathbf{Q}}_{i,v} = \mathbf{P}_{i+1,\pi_{i,1}(v)} + \widetilde{\mathbf{E}}_{i,v}. \end{array}$$

Next, for $i \in [\ell]$, we use matrices $\mathbf{M}_i, \mathbf{W}_i, \widetilde{\mathbf{W}}_i$ to represent the following $(4\lambda + w)n \times m$ dimension matrices:

$$\mathbf{M}_i = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{j\in[\lambda],\beta,b\in\{0,1\}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, \quad \mathbf{W}_i = \begin{bmatrix} \left\{\mathbf{D}_{i,b}^{(j,\beta)}\right\}_{j\in[\lambda],\beta,b\in\{0,1\}} \\ \{\mathbf{Q}_{i,v}\}_{v\in[w]} \end{bmatrix}, \quad \widetilde{\mathbf{W}}_i = \begin{bmatrix} \left\{\widetilde{\mathbf{D}}_{i,b}^{(j,\beta)}\right\}_{j\in[\lambda],\beta,b\in\{0,1\}} \\ \left\{\widetilde{\mathbf{Q}}_{i,v}\right\}_{v\in[w]} \end{bmatrix}.$$

Now, the secret key encryption algorithm runs the $\mathsf{EnSamplePre}$ to compute 2ℓ short matrices $\{\mathbf{U}_{i,b}\}_{i,b}$ as

$$\forall\ i \in [\ell], \qquad \begin{array}{l} \mathbf{U}_{i,0} \leftarrow \mathsf{EnSamplePre}(\mathbf{M}_i, T_i, \sigma_{\mathsf{pre}}, \mathbf{W}_i), \\ \mathbf{U}_{i,1} \leftarrow \mathsf{EnSamplePre}(\mathbf{M}_i, T_i, \sigma_{\mathsf{pre}}, \widetilde{\mathbf{W}}_i). \end{array}$$

Finally, it outputs the ciphertext ct as

$$\mathsf{ct} = \left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right).$$

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \{0,1\}$. The decryption algorithm takes as input a secret key sk and a ciphertext ct. Let

$$\mathsf{sk} = \left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i\in[\ell+1],j\in[\lambda],\beta\in\{0,1\}}\right), \qquad \mathsf{ct} = \left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right).$$

We will assume the algorithm knows the LWE modulus q (i.e., for instance the public parameters could be included in the secret keys). Let $\widetilde{x} = x^L$, i.e. $\widetilde{x}$ is a $k \cdot L$-bit string obtained by appending string x to itself L times. The decryption algorithm computes the following

$$\mathbf{z} = \sum_{j=1}^{\lambda} \sum_{i=1}^{\ell+1} \left(\mathbf{t}_i^{(j,\mathsf{tag}_j)} \cdot \prod_{\alpha=i}^{\ell} \mathbf{U}_{\alpha,\widetilde{x}_\alpha}\right).$$

Finally, if $\|\mathbf{z}\| \leq q/8$, it outputs 0, otherwise it outputs 1.

Theorem 8.1. Assuming the trapdoor sampling scheme $\mathsf{LT}_{\mathsf{en}}$ satisfies the $(q, \sigma_{\mathsf{pre}})$-well sampledness of preimage, $(q, \sigma_{\mathsf{pre}})$-row removal property and the $(q, \chi_{\mathsf{lwe}}, \sigma_{\mathsf{pre}})$-target switching property, assuming the $\mathsf{LWE}_{n,q,\chi_{\mathsf{lwe}}}$, $\mathsf{LWE\text{-}ss}_{n,q,\chi_{\mathsf{lwe}}}$ and $\mathsf{LWE\text{-}sp}_{\lambda \log q, \sigma_{\mathsf{pre}}, \chi_{\mathsf{appr}}}$ assumptions hold (where $n, m, q, \sigma_{\mathsf{pre}}, \chi_{\mathsf{lwe}}, \chi_{\mathsf{appr}}$ are defined as in the construction), for any PPT adversary $\mathcal{A}$ that outputs $(1^k, 1^w, 1^L)$ such that the parameter constraints as provided in the construction are satisfied, then there exist negligible functions $\mathrm{negl}_1(\cdot), \mathrm{negl}_2(\cdot)$ such that for every $\lambda \in \mathbb{N}$, $\mathcal{A}$'s advantage in the 1-bounded restricted function indistinguishability security (see Definition 5.2) and 1-bounded restricted accept indistinguishability (see Definition 5.4) is at most $\mathrm{negl}_1(\lambda)$ and $\mathrm{negl}_2(\lambda)$, respectively.

Remark 8.1 (Extending to r-bounded security)**.** We would like to point out that the above construction can be naturally extended to achieve r-bounded security for any a-priori fixed polynomial r. To understand the modification, we will look ahead to the security proof. Specifically, we will focus on the importance of the λ-bit string tag chosen during encryption. During the proof, we crucially rely on the tag strings tag and tag^* (first one chosen for answering the encryption query and second one used to answer the challenge query) being distinct at at least one index. Since they are chosen uniformly at random each time, thus we know that $\mathsf{tag} \neq \mathsf{tag}^*$ with probability $1 - \frac{1}{2^\lambda}$. Now if the challenger has to answer r encryption queries instead of just 1, then the modification we consider is to increase the alphabet size of tags such that the tag strings chosen during all encryption queries and the challenge query are distinct at at least one index. (Note that this would also mean that we will have to likewise increase the number of underlying matrices chosen and extend the trapdoor sampling procedure appropriately.) More formally, we will now sample tag strings as a uniformly random $2r^2$-ary string of length-λ (i.e., $\mathsf{tag} \leftarrow \{1, \ldots, 2r^2\}^\lambda$). With this modification we can argue that, with all but negligible probability over the choice of tag strings $\mathsf{tag}_1, \ldots, \mathsf{tag}_r$ and tag^*, there exists an index $i \leq \lambda$ such that the i^{th} elements in all these tag strings are (pairwise-)distinct. With this guarantee, the current proof could be extended to argue r-bounded security.

8.3 Correctness

We will prove that the mixed FE scheme described above satisfies the correctness property. Our correctness proof is divided into two parts. First, we show that if ct is a mixed FE encryption of branching program BP, then given any secret key sk_x, the decryption algorithm outputs $\mathsf{BP}(x)$ with all-but-negligible probability. Second, we show that if ct is a normal FE ciphertext, then given any secret key sk_x, the decryption algorithm outputs 1 with all-but-negligible probability.

Lemma 8.1. For every $\lambda, k, w, L \in \mathbb{N}$, for every length $k \cdot L$ and width w input-circling branching program BP with input space $\{0,1\}^k$, input $x \in \{0,1\}^k$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_x, \mathsf{ct}) = \mathsf{BP}(x) : \begin{array}{c} (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, (1^k, 1^w, 1^L)); \\ \mathsf{sk}_x \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x); \mathsf{ct} \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, \mathsf{BP}) \end{array}\right] \geq 1 - \mathrm{negl}_2(\lambda)$$

where $\mathrm{negl}_2(\cdot)$ is a negligible function.

Proof. Recall that the setup algorithm chooses matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}, \mathbf{P}_{i,v}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}, v \in [w]$. Here all matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{P}_{i,v}$ for any particular value i (i.e., any fixed level) are sampled along with trapdoor information. Now for any input $x \in \{0,1\}^k$, the key generation algorithm chooses vectors $\mathbf{y}^{(j)}, \widetilde{\mathbf{s}}$ such that $\widetilde{\mathbf{s}}$ is short and $\sum_j \mathbf{y}^{(j)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}$. It also samples secret vectors $\mathbf{s}_i^{(j,\beta)}$ and error vectors $\mathbf{e}_i^{(j,\beta)}$, and computes the secret key components $\mathbf{t}_i^{(j,\beta)}$ in the special way as described in the construction. Now the mixed encryption algorithm samples a λ-bit tag string tag, and it uses the trapdoor information to target $\mathbf{B}_{i,b}^{(j,\beta)}$ matrices to their corresponding $\mathbf{C}_{i,b}^{(j,\beta)}$ matrices only along the strands selected by the tag string tag. Additionally, it also targets the program matrices $\mathbf{P}_{i,v}$ at each level to their counterparts in the next level as per the branching program state transition function. For proving correctness we simply show that the final program matrix reached after decryption is either short or random depending upon outcome of the evaluation, and the $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ matrices get cancelled at each step, and the error terms are appropriately bounded.

We start by introducing some notations useful for the correctness proof.

- st_i: the state of BP after i steps when evaluated on input x,
- $\widetilde{\mathbf{t}}_i^{(j,\beta)} \stackrel{\text{def}}{=} \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} & \text{if } i = \ell + 1 \end{cases}$: the *error-free* secret key components, i.e. secret key vectors without adding error vectors $\mathbf{e}_i^{(j,\beta)}$.
- $\mathbf{\Delta}_i^{(j)} \stackrel{\text{def}}{=} \sum_{\gamma=1}^{i} \mathbf{t}_\gamma^{(j,\mathsf{tag}_j)} \cdot \prod_{\alpha=\gamma}^{i-1} \mathbf{U}_{\alpha,\widetilde{x}_\alpha}$: the partial sum computed during decryption after using first i components of the secret key along only the $(j, \mathsf{tag}_j)^{th}$ strand,
- $\widetilde{\mathbf{\Delta}}_i^{(j)} \stackrel{\text{def}}{=} \begin{cases} \mathbf{s}_i^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\mathsf{tag}_j)} + \mathbf{y}^{(j)} \cdot \prod_{\alpha=1}^{i-1} \mathbf{U}_{\alpha,\widetilde{x}_\alpha} & \text{if } i \leq \ell \\ \mathbf{y}^{(j)} \cdot \prod_{\alpha=1}^{\ell} \mathbf{U}_{\alpha,\widetilde{x}_\alpha} & \text{if } i = \ell + 1 \end{cases}$: the expected sum during decryption in absence of *errors* after using first i components of the secret key along the $(j, \mathsf{tag}_j)^{th}$ strand,
- $\mathbf{\Delta}_i \stackrel{\text{def}}{=} \sum_{j=1}^{\lambda} \mathbf{\Delta}_i^{(j)}$, and $\widetilde{\mathbf{\Delta}}_i = \sum_{j=1}^{\lambda} \widetilde{\mathbf{\Delta}}_i^{(j)}$,
- $\mathbf{err}_i^{(j)} \stackrel{\text{def}}{=} \mathbf{\Delta}_i^{(j)} - \widetilde{\mathbf{\Delta}}_i^{(j)}$, and $\mathbf{err}_i \stackrel{\text{def}}{=} \mathbf{\Delta}_i - \widetilde{\mathbf{\Delta}}_i$,
- $\mathbf{\Gamma}_i \stackrel{\text{def}}{=} \mathbf{P}_{1,1} \cdot \prod_{\alpha=1}^{i} \mathbf{U}_{\alpha,\widetilde{x}_\alpha}$: the matrix denoting partial branching program evaluation after i decryption steps, i.e. equal to $\mathbf{\Delta}_i$ term ignoring the $\sum_{j=1}^{\lambda} \mathbf{s}_i^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\mathsf{tag}_j)}$ blinding component, and short secret $\widetilde{\mathbf{s}}$ multiplied.

Observe that the decryption algorithm computes $\mathbf{\Delta}_{\ell+1}$ and tests whether it is close to zero or not. We start by proving that for all i, j, the error term $\mathbf{err}_i^{(j)}$ is small and bounded. This would help us in arguing that for every i, $\mathbf{err}_i$ is also small, thereby giving us that matrices $\mathbf{\Delta}_{\ell+1}$ and $\widetilde{\mathbf{\Delta}}_{\ell+1}$ are very close to each other as well. Combining this with the fact that $\widetilde{\mathbf{\Delta}}_{\ell+1}$ is either a random matrix or a short matrix depending upon the output $\mathsf{BP}(x)$, we get that the sum $\mathbf{\Delta}_{\ell+1}$ computed by decryption algorithm is close to zero if $\mathsf{BP}(x) = 0$, otherwise it is random vector with large entries.

Claim 8.1. There exists a negligible function $\mathrm{negl}(\cdot)$ such that

$$\forall\, i \in [\ell], j \in [\lambda], \quad \left\| \mathbf{err}_i^{(j)} \right\| \leq (m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}}))^i$$
$$\forall\, j \in [\lambda], \quad \left\| \mathbf{err}_{\ell+1}^{(j)} \right\| \leq (m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}}))^\ell \cdot (m \cdot (\sigma_{\mathsf{last}} + \sigma_{\mathsf{pre}}))$$

with probability $1 - \mathrm{negl}(\lambda)$.

Proof. We prove the above claim by inducting on the levels i. Our proof is insensitive to the choice of strand index j, thus for the purposes of this proof, it could be fixed to an arbitrary value.

Base case $(i = 1)$**.** Note that $\mathbf{\Delta}_1^{(j)} = \mathbf{t}_1^{(j,\mathsf{tag}_j)} = \mathbf{s}_1^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\mathsf{tag}_j)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\mathsf{tag}_j)}$, where $\mathbf{e}_1^{(j,\mathsf{tag}_j)}$ is a short error vector drawn from χ_{big}^m. Also, we have that $\widetilde{\mathbf{\Delta}}_1^{(j)} = \mathbf{s}_1^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\mathsf{tag}_j)} + \mathbf{y}^{(j)}$ by definition. Thus, we get that

$$\left\| \mathbf{err}_1^{(j)} \right\| = \left\| \mathbf{\Delta}_1^{(j)} - \widetilde{\mathbf{\Delta}}_1^{(j)} \right\| = \left\| \mathbf{e}_1^{(j,\mathsf{tag}_j)} \right\| \leq \sqrt{m} \cdot \sigma_{\mathsf{big}}$$

with all-but-negligible probability. This completes the proof of base case. For the induction step, we assume that the above lemma holds for i^*, and show that it holds for $i^* + 1$ as well.

Induction Step. We know that $\mathbf{\Delta}_{i^*+1}^{(j)} = \mathbf{\Delta}_{i^*}^{(j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} + \mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)}$. Since $\mathbf{\Delta}_{i^*}^{(j)} = \widetilde{\mathbf{\Delta}}_{i^*}^{(j)} + \mathbf{err}_{i^*}^{(j)}$, we get that

$$\begin{aligned}\mathbf{\Delta}_{i^*+1}^{(j)} &= (\widetilde{\mathbf{\Delta}}_{i^*}^{(j)} + \mathbf{err}_{i^*}^{(j)}) \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} + \mathbf{t}_{i^*+1}^{(j,\mathsf{tag}_j)} \\ &= \widetilde{\mathbf{\Delta}}_{i^*}^{(j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} + \mathbf{t}_{i^*+1}^{(j,\mathsf{tag}_j)} + \mathbf{err}_{i^*}^{(j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}}\end{aligned}$$

Now, from our construction we have that

$$\begin{aligned}&\mathbf{s}_{i^*}^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} = \mathbf{s}_{i^*}^{(j,\mathsf{tag}_j)} \cdot \mathbf{C}_{i^*,\widetilde{x}_{i^*}}^{(j,\mathsf{tag}_j)} \\ &\Rightarrow \mathbf{s}_{i^*}^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} + \widetilde{\mathbf{t}}_{i^*+1}^{(j,\mathsf{tag}_j)} = \mathbf{s}_{i^*+1}^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j,\mathsf{tag}_j)} \\ &\Rightarrow \widetilde{\mathbf{\Delta}}_{i^*}^{(j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} + \widetilde{\mathbf{t}}_{i^*+1}^{(j,\mathsf{tag}_j)} = \widetilde{\mathbf{\Delta}}_{i^*+1}^{(j)}\end{aligned}$$

Combining the fact that $\mathbf{t}_{i^*+1}^{(j,\mathsf{tag}_j)} = \widetilde{\mathbf{t}}_{i^*+1}^{(j,\mathsf{tag}_j)} + \mathbf{e}_{i^*+1}^{(j,\mathsf{tag}_j)}$ with above equations we get that, with all-but-negligible probability, the following holds

$$\begin{aligned}\mathbf{err}_{i^*+1}^{(j)} = \mathbf{\Delta}_{i^*+1}^{(j)} - \widetilde{\mathbf{\Delta}}_{i^*+1}^{(j)} &= \mathbf{e}_{i^*+1}^{(j,\mathsf{tag}_j)} + \mathbf{err}_{i^*}^{(j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} \\ \Rightarrow \left\|\mathbf{err}_{i^*+1}^{(j)}\right\| &\le \left\|\mathbf{e}_{i^*+1}^{(j,\mathsf{tag}_j)}\right\| + \left\|\mathbf{err}_{i^*}^{(j)}\right\| \cdot \|\mathbf{U}_{i^*,\widetilde{x}_{i^*}}\| \\ &\le \sqrt{m} \cdot \sigma^* + (m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}}))^{i^*} \cdot (m \cdot \sigma_{\mathsf{pre}}) \\ &\le (m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}}))^{i^*} \cdot (m \cdot (\sigma^* + \sigma_{\mathsf{pre}})),\end{aligned}$$

where $\sigma^* = \sigma_{\mathsf{big}}$ if $i^* < \ell$, otherwise $\sigma^* = \sigma_{\mathsf{last}}$. This completes the proof of above claim. ∎

From the above claim, we get that for every $j \in [\lambda]$, $\left\|\mathbf{err}_{\ell+1}^{(j)}\right\| \le (m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}}))^{\ell} \cdot (m \cdot (\sigma_{\mathsf{last}} + \sigma_{\mathsf{pre}}))$. Thus, by triangle inequality we can claim that (with all-but-negligible probability)

$$\|\mathbf{err}_{\ell+1}\| \le \lambda \cdot (m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}}))^{\ell} \cdot (m \cdot (\sigma_{\mathsf{last}} + \sigma_{\mathsf{pre}})) \le q/16.$$

Next, we show that $\mathbf{\Gamma}_i$ is close to $\mathbf{P}_{i+1,\mathsf{st}_i}$. In other words, the partial branching program evaluation is correct.

Claim 8.2. For all $i \in \{0, \dots, \ell\}$, $\|\mathbf{\Gamma}_i - \mathbf{P}_{i+1,\mathsf{st}_i}\| \le (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^i$ with probability $1 - \text{negl}(\lambda)$, where negl$(\cdot)$ is a negligible function.

Proof. We prove the above claim by inducting on the levels i.

Base case $(i = 0)$**.** Note that $\mathbf{\Gamma}_0$ is simply equal to $\mathbf{P}_{1,1}$ as starting state $\mathsf{st}_0 = 1$. Thus, we get that

$$\|\mathbf{\Gamma}_0 - \mathbf{P}_{1,\mathsf{st}_0}\| = 0.$$

This completes the proof of base case. For the induction step, we assume that the above lemma holds for $i^* - 1$, and show that it holds for i^* as well.

Induction Step. We know that $\mathbf{\Gamma}_{i^*} = \mathbf{\Gamma}_{i^*-1} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}}$. Recall that, as per our construction, $\mathbf{U}_{i^*,\widetilde{x}_{i^*}}$ targets $\mathbf{P}_{i^*,\mathsf{st}_{i^*-1}}$ to $\mathbf{P}_{i^*+1,\mathsf{st}_{i^*}} + \mathbf{Err}_1$, where $\mathbf{Err}_1$ is a $n \times m$ matrix sampled uniformly from $\chi_{\mathsf{appr}}^{n \times m}$. Concretely, this gives that

$$\mathbf{P}_{i^*,\mathsf{st}_{i^*-1}} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} = \mathbf{P}_{i^*+1,\mathsf{st}_{i^*}} + \mathbf{Err}_1, \text{ where } \|\mathbf{Err}_1\| \le m \cdot \sigma_{\mathsf{appr}}.$$

By our inductive hypothesis, we have that $\mathbf{\Gamma}_{i^*-1} = \mathbf{P}_{i^*,\mathsf{st}_{i^*-1}} + \mathbf{Err}_2$, where $\|\mathbf{Err}_2\| \leq (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^{i^*-1}$. Thus, we can rewrite matrix $\mathbf{\Gamma}_{i^*}$ as follows

$$\begin{aligned}\mathbf{\Gamma}_{i^*} &= \left(\mathbf{P}_{i^*,\mathsf{st}_{i^*-1}} + \mathbf{Err}_2\right) \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} \\ &= \mathbf{P}_{i^*+1,\mathsf{st}_{i^*}} + \left(\mathbf{Err}_1 + \mathbf{Err}_2 \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}}\right).\end{aligned}$$

Now we have that

$$\left\|\mathbf{Err}_1 + \mathbf{Err}_2 \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}}\right\| \leq m \cdot \sigma_{\mathsf{appr}} + (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^{i^*-1} \cdot m \cdot \sigma_{\mathsf{pre}} \leq (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^{i^*}.$$

Thus, the claim follows. ∎

From the above claim, we get that (with all-but-negligible probability) $\|\mathbf{\Gamma}_\ell - \mathbf{P}_{\ell+1,\mathsf{st}_\ell}\| \leq (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^{\ell}$. Next, we show that $\widetilde{\mathbf{\Delta}}_{\ell+1}$ has low norm if the output of branching program is 0, otherwise it is not upper-bounded with all-but-negligible probability.

Claim 8.3. There exists a negligible function negl$(\cdot)$ such that $\left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| = \begin{cases} \leq q/16 & \text{if } \mathsf{BP}(x) = 0 \\ \geq q/4 & \text{if } \mathsf{BP}(x) = 1 \end{cases}$ with probability $1 - \text{negl}(\lambda)$.

Proof. We know that $\widetilde{\mathbf{\Delta}}_{\ell+1}$ could be written as follows

$$\widetilde{\mathbf{\Delta}}_{\ell+1} = \left(\sum_{j=1}^{\lambda} \mathbf{y}^{(j)}\right) \cdot \prod_{\alpha=1}^{\ell} \mathbf{U}_{\alpha,\widetilde{x}_\alpha}.$$

Since $\left(\sum_{j=1}^{\lambda} \mathbf{y}^{(j)}\right) = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}$, this gives that $\widetilde{\mathbf{\Delta}}_{\ell+1} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} \cdot \prod_{\alpha=1}^{\ell} \mathbf{U}_{\alpha,\widetilde{x}_\alpha} = \widetilde{\mathbf{s}} \cdot \mathbf{\Gamma}_\ell$. Using Claim 8.2, we get that $\mathbf{\Gamma}_\ell = \mathbf{P}_{\ell+1,\mathsf{st}_\ell} + \mathbf{Err}$, where $\|\mathbf{Err}\| \leq (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^{\ell}$. Also, we know that $\mathbf{P}_{\ell+1,\mathsf{rej}} = \mathbf{0}^{n \times m}$, and $\mathbf{P}_{\ell+1,\mathsf{acc}}$ is a uniformly random $n \times m$ matrix. Thus, we get that with all-but-negligible probability

$$\begin{aligned}\mathsf{BP}(x) = 0 &\Rightarrow \left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| = \|\widetilde{\mathbf{s}} \cdot \mathbf{Err}\| \leq \|\widetilde{\mathbf{s}}\| \cdot (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^{\ell} \leq \sqrt{n} \cdot \sigma_s \cdot (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^{\ell} \leq q/16. \\ \mathsf{BP}(x) = 1 &\Rightarrow \left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| = \|\widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{acc}} + \widetilde{\mathbf{s}} \cdot \mathbf{Err}\| \geq \|\widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{acc}}\| - q/16 \geq q/4,\end{aligned}$$

where the last inequality follows from the fact that $\widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{acc}}$ is uniformly random vector. This completes the proof of above claim. ∎

By triangle inequality, we know that

$$\left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| - \|\mathbf{err}_{\ell+1}\| \leq \|\mathbf{\Delta}_{\ell+1}\| \leq \left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| + \|\mathbf{err}_{\ell+1}\|.$$

Combining this with above claims, we can conclude that with all-but-negligible probability

$$\begin{aligned}\mathsf{BP}(x) = 0 &\Rightarrow \|\mathbf{\Delta}_{\ell+1}\| \leq q/16 + q/16 \leq q/8. \\ \mathsf{BP}(x) = 1 &\Rightarrow \left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| \geq q/4 - q/16 > q/8.\end{aligned}$$

Thus, for any input x and branching program BP, the mixed encryption algorithm is correct with all-but-negligible probability. This concludes the proof of Lemma 8.1. ∎

Lemma 8.2. For every $\lambda, k, w, L \in \mathbb{N}$, for every length $k \cdot L$ and width w input-circling branching program BP with input space $\{0,1\}^k$, input $x \in \{0,1\}^k$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_x, \mathsf{ct}) = 1 : \begin{array}{c} (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, (1^k, 1^w, 1^L)); \\ \mathsf{sk}_x \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}) \end{array}\right] \geq 1 - \mathrm{negl}_1(\lambda)$$

where $\mathrm{negl}_1(\cdot)$ is a negligible function.

Proof. Recall that the output of a normal encryption algorithm is simply independently drawn 2ℓ short gaussian matrices $\{\mathbf{U}_{i,b}\}$. Now the decryption algorithm performs the following computation

$$\mathbf{z} = \sum_{j=1}^{\lambda} \sum_{i=1}^{\ell+1} \left(\mathbf{t}_i^{(j,\mathsf{tag}_j)} \cdot \prod_{\alpha=i}^{\ell} \mathbf{U}_{\alpha, \widetilde{x}_\alpha} \right).$$

Here we could rewrite $\mathbf{z}$ as

$$\mathbf{z} = \sum_{j=1}^{\lambda} \left(\sum_{i=1}^{\ell-1} \left(\mathbf{t}_i^{(j,\mathsf{tag}_j)} \cdot \prod_{\alpha=i}^{\ell} \mathbf{U}_{\alpha, \widetilde{x}_\alpha} \right) + \mathbf{t}_\ell^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell, \widetilde{x}_\ell} + \mathbf{t}_{\ell+1}^{(j,\mathsf{tag}_j)} \right).$$

From our construction we know that for any j,

$$\begin{aligned} \mathbf{t}_\ell^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell, \widetilde{x}_\ell} + \mathbf{t}_{\ell+1}^{(j,\mathsf{tag}_j)} &= -\mathbf{s}_{\ell-1}^{(j,\mathsf{tag}_j)} \cdot \mathbf{C}_{\ell-1, \widetilde{x}_{\ell-1}}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell, \widetilde{x}_\ell} + \mathbf{e}_\ell^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell, \widetilde{x}_\ell} + \mathbf{e}_{\ell+1}^{(j,\mathsf{tag}_j)} \\ &\quad + \mathbf{s}_\ell^{(j,\mathsf{tag}_j)} \cdot \left(\mathbf{B}_{\ell, \widetilde{x}_\ell}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell, \widetilde{x}_\ell} - \mathbf{C}_{\ell, \widetilde{x}_\ell}^{(j,\mathsf{tag}_j)} \right). \end{aligned}$$

Now note that since $\mathbf{U}_{\ell, \widetilde{x}_\ell}$ is sampled independently from $\chi_{\mathsf{pre}}^{m \times m}$ and $\mathbf{C}_{\ell, \widetilde{x}_\ell}^{(j,\mathsf{tag}_j)}$ is uniform $n \times m$ matrix, thus the matrix $\mathbf{B}_{\ell, \widetilde{x}_\ell}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell, \widetilde{x}_\ell} - \mathbf{C}_{\ell, \widetilde{x}_\ell}^{(j,\mathsf{tag}_j)}$ is also a uniformly random matrix. Also, secret vector $\mathbf{s}_\ell^{(j,\mathsf{tag}_j)}$ is a length n random vector, thus the component $\mathbf{s}_\ell^{(j,\mathsf{tag}_j)} \cdot \left(\mathbf{B}_{\ell, \widetilde{x}_\ell}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell, \widetilde{x}_\ell} - \mathbf{C}_{\ell, \widetilde{x}_\ell}^{(j,\mathsf{tag}_j)} \right)$ is a random vector as well. Now since this is *independent* of all other components as $\mathbf{s}_\ell^{(j,\mathsf{tag}_j)}$ and $\mathbf{C}_{\ell, \widetilde{x}_\ell}^{(j,\mathsf{tag}_j)}$ both do not appear in any other term in sum vector $\mathbf{z}$, thus the distribution of $\mathbf{z}$ is that of a uniformly random vector over the choice of coins used during setup, key generation, and encryption. Since we know that ℓ_2-norm of a random vector in $\mathbb{Z}_q^m$ is at least $q/8$ with all-but-negligible probability, therefore the claim follows. ■

8.4 Security Proof

We now prove that the mixed FE scheme described in Section 8.2 satisfies 1-bounded restricted function indistinguishability as well as 1-bounded restricted accept indistinguishability security properties. Our proof is divided in two components where we first prove function indistinguishability, and later prove accept indistinguishability. Both proofs proceed via a sequence of hybrid games.

8.4.1 1-Bounded Restricted Function Indistinguishability

Below we provide a sequence of hybrid games that we later use to argue function indistinguishability security.

$\mathsf{Game}\ 0$: This corresponds to the original 1-bounded restricted function indistinguishability security game.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows—
    1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$.

2. Next, it samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix}\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]}\end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.
4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$, and a λ-bit string $\mathsf{tag}^* \leftarrow \{0,1\}^\lambda$. Let

$$\begin{aligned}\mathsf{BP}^{(\gamma)} &= \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right), \\ S^* &= [\ell]\times[\lambda]\times\{0,1\}^2.\end{aligned}$$

The challenger then runs the Mixed-SubEnc routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)}\end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger chooses a λ-bit string $\mathsf{tag} \leftarrow \{0,1\}^\lambda$, and responds as follows.
     (a) Let $S = [\ell]\times[\lambda]\times\{0,1\}^2$. It runs the Mixed-SubEnc routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}\end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda-1]$. Next, it sets vector $\mathbf{y}^{(\lambda)}$ as

$$\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}}\cdot\mathbf{P}_{1,1} - \sum_{j\in[\lambda-1]} \mathbf{y}^{(j)}.$$

     (b) It then chooses secret vectors $\left\{\mathbf{s}_i^{(j,\beta)}\right\}_{i,j,\beta}$ and error vectors $\left\{\mathbf{e}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as

$$\begin{aligned}&\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\ (j,\beta) \in [\lambda]\times\{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m.\end{aligned}$$

(c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.
$\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Game 1 : This is identical to the previous game, except the challenger now chooses both tags tag^* and tag at the beginning during setup phase, and it aborts if $\mathsf{tag}^* = \mathsf{tag}$.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below.
  2. Next, it samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.
- **Challenge Phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let
$$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),$$
$$S^* = [\ell] \times [\lambda] \times \{0,1\}^2.$$
The challenger then runs the Mixed-SubEnc routine (described in Figure 6) as
$$\forall\ \alpha \in [\ell], \quad \left(\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{array}\right).$$
Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}_{i,b}^*\}_{i\in[\ell],b\in\{0,1\}}\right)$.
- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.

Mixed-SubEnc

Inputs:

- Tag tag, Level α, Set $S \subseteq [\ell] \times [\lambda] \times \{0,1\}^2$, Matrices $\left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}$, Trapdoors $\{T_i\}_{i\in[\ell]}$,
- Branching program $\mathsf{BP} = \left(\{\pi_{i,b} : [w] \to [w]\}_{(i,b)\in[\ell]\times\{0,1\}}, \mathsf{acc} \in [w], \mathsf{rej} \in [w]\right)$.

Output: Matrices $\{\mathbf{U}_0, \mathbf{U}_1\}$.
Execution: Let S_α denote the following set:

$$S_\alpha = \left\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2 \text{ such that } (\alpha,j,\beta,b) \in S\right\}.$$

Sample matrices $\left\{\mathbf{D}_b^{(j,\beta)}, \widetilde{\mathbf{D}}_b^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha}$ as:

$$\forall\ (j,\beta,b) \in S_\alpha, \qquad \begin{aligned} \mathbf{D}_b^{(j,\beta)} &= \begin{cases} \mathbf{C}_{\alpha,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 0 \\ (\leftarrow \mathbb{Z}_q^{n\times m}) & \text{otherwise.} \end{cases} \\ \widetilde{\mathbf{D}}_b^{(j,\beta)} &= \begin{cases} \mathbf{C}_{\alpha,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 1 \\ (\leftarrow \mathbb{Z}_q^{n\times m}) & \text{otherwise.} \end{cases} \end{aligned}$$

Sample $2w$ error matrices as $\mathbf{E}_v \leftarrow \chi_{\mathsf{appr}}^{n\times m}, \widetilde{\mathbf{E}}_v \leftarrow \chi_{\mathsf{appr}}^{n\times m}$ for $v \in [w]$. Also, if $\alpha = \ell$, sample w matrices $\{\mathbf{P}_{\ell+1,v}\}_{v\in[w]}$ for the top level as:

$$\forall\ v \in [w], \quad \mathbf{P}_{\ell+1,v} = \begin{cases} \mathbf{0}^{n\times m} & \text{if } v = \mathsf{rej} \\ (\leftarrow \mathbb{Z}_q^{n\times m}) & \text{otherwise.} \end{cases}$$

Next, set $2w$ matrices $\left\{\mathbf{Q}_v, \widetilde{\mathbf{Q}}_v\right\}_{v\in[w]}$ as:

$$\forall\ v \in [w], \qquad \begin{aligned} \mathbf{Q}_v &= \mathbf{P}_{\alpha+1,\pi_{\alpha,0}(v)} + \mathbf{E}_v, \\ \widetilde{\mathbf{Q}}_v &= \mathbf{P}_{\alpha+1,\pi_{\alpha,1}(v)} + \widetilde{\mathbf{E}}_v. \end{aligned}$$

Let matrices $\mathbf{M}, \mathbf{W}, \widetilde{\mathbf{W}}$ represent the following $(|S_\alpha| + w)n \times m$ dimension matrices:

$$\mathbf{M} = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} \left\{\mathbf{D}_b^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \left\{\mathbf{Q}_{i,v}\right\}_{v\in[w]} \end{bmatrix}, \quad \widetilde{\mathbf{W}} = \begin{bmatrix} \left\{\widetilde{\mathbf{D}}_b^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \left\{\widetilde{\mathbf{Q}}_{i,v}\right\}_{v\in[w]} \end{bmatrix}.$$

Run the $\mathsf{EnSamplePre}$ to compute matrices $\{\mathbf{U}_0, \mathbf{U}_1\}$ as

$$\begin{aligned} \mathbf{U}_0 &\leftarrow \mathsf{EnSamplePre}(\mathbf{M}, T_\alpha, \sigma_{\mathsf{pre}}, \mathbf{W}), \\ \mathbf{U}_1 &\leftarrow \mathsf{EnSamplePre}(\mathbf{M}, T_\alpha, \sigma_{\mathsf{pre}}, \widetilde{\mathbf{W}}). \end{aligned}$$

Figure 6: Routine Mixed-SubEnc

(a) Let $S = [\ell] \times [\lambda] \times \{0,1\}^2$. It runs the Mixed-SubEnc routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda - 1]$. Next, it sets vector $\mathbf{y}^{(\lambda)}$ as

$$\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j \in [\lambda-1]} \mathbf{y}^{(j)}.$$

(b) It then chooses secret vectors $\left\{\mathbf{s}_i^{(j,\beta)}\right\}_{i,j,\beta}$ and error vectors $\left\{\mathbf{e}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m.
\end{aligned}$$

(c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.
$\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell + 1 \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Notation. In all the following hybrid games, let j^* denote the smallest index in $\{1, \ldots, \lambda\}$ such that $\mathsf{tag}^*_{j^*} \neq \mathsf{tag}_{j^*}$, i.e. $j^* = \min\left\{j \in [\lambda] \ : \ \mathsf{tag}^*_j \neq \mathsf{tag}_j\right\}$. Since the challenger aborts whenever $\mathsf{tag}^* = \mathsf{tag}$, thus j^* always exists whenever the challenger does not abort. Additionally, let $\beta^* = \mathsf{tag}^*_{j^*}$.

$\mathsf{Game}\ 2$: This is identical to the previous game, except the challenger while answering a secret key query now puts the $\widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}$ component in $\mathbf{y}^{(j^*)}$ instead of $\mathbf{y}^{(\lambda)}$, and rest are sampled uniformly at random.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows—

1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below.
2. Next, it samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b) \in [\lambda] \times \{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.
4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

$$\begin{aligned}\mathsf{BP}^{(\gamma)} &= \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),\\ S^* &= [\ell] \times [\lambda] \times \{0,1\}^2.\end{aligned}$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*},\\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)}\end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}^*_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = [\ell] \times [\lambda] \times \{0,1\}^2$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S},\\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}\end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$. Next, it sets vector $\mathbf{y}^{(j^*)}$ as

$$\mathbf{y}^{(j^*)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}.$$

     (b) It then chooses secret vectors $\left\{\mathbf{s}_i^{(j,\beta)}\right\}_{i,j,\beta}$ and error vectors $\left\{\mathbf{e}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as

$$\begin{aligned}\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, &\quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,\\ \forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, &\quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,\\ \forall\ (j,\beta) \in [\lambda] \times \{0,1\}, &\quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m.\end{aligned}$$

     (c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.
     $\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases}\mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1\\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell\\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1\end{cases}$$

     (d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Next, we have a sequence of 4ℓ hybrid experiments $\mathsf{Game}\ 3.i^*.\{1,2,3,4\}$ for $i^* = 1$ to ℓ.

Game 3.i^*.1 : In hybrids Game 3.i^*.1, the $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ matrices for j^{th} strands and levels $i < i^*$ are **not** sampled (at all) along with other level i matrices (i.e., (j^*, β^*) and $(j^*, 1-\beta^*)$ strands). And, ciphertext components for levels $i < i^*$ are used to only target remaining matrices, i.e. the ciphertext matrices do not target $\mathbf{B}_{i,b}^{(j,\beta)}$ matrices for $j = j^*$ and $i < i^*$ to some pre-specified $\mathbf{C}_{i,b}^{(j,\beta)}$ or random matrices. Also, the first $i^* - 1$ components in each secret key are set to be uniformly random vectors, and the next component is hardwired such that correctness holds as well as some smudge-able noise is introduced in these components. Below we describe it in detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows—

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

  $$\forall\, i < i^*, \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2,$$
  $$\forall\, i \geq i^*, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$

  Also, let $\widetilde{n}_i = \begin{cases} \widetilde{n} - 4n & \text{for } i < i^* \\ \widetilde{n} & \text{for } i \geq i^* \end{cases}$, and set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j, \beta, b) \in S^{(i)}\right\}$.

  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

  $$\forall\, i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i, j, \beta, b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

  $$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell], b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),$$
  $$S^* = \hat{S}.$$

  The challenger then runs the Mixed-SubEnc routine (described in Figure 6) as

  $$\forall\, \alpha \in [\ell], \quad \left(\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{array}\right).$$

  Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}_{i,b}^*\}_{i\in[\ell], b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.

(a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}\end{array}\right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

(b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
&\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
&\forall\ (j,\beta) \in [\lambda]\times\{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
&\forall\ (i,\beta) \in [i^*-1]\times\{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
\end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after i^*-1 steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1]\times([\lambda]\setminus\{j^*\})\times\{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}$$

$$\forall\ (i,\beta,b) \in [\ell]\times\{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases}\mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*\end{cases}$$

Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases}\mathbf{s}_1^{(j,\beta)}\cdot\mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)}\cdot\mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)}\cdot\mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell \\ -\mathbf{s}_\ell^{(j,\beta)}\cdot\mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1\end{cases}$$

$$\forall\ (i,\beta) \in [i^*-1]\times\{0,1\}, \quad \mathbf{t}_i^{(j^*,\beta)} = \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)}$$

$$\begin{aligned}\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*}^{(j^*,\beta)} = &-\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)}\cdot\prod_{\delta=\alpha}^{i^*-1}\mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) - \widetilde{\mathbf{y}}\cdot\prod_{\delta=1}^{i^*-1}\mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\ &+ \widetilde{\mathbf{s}}\cdot\mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)}\cdot\mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} + \mathbf{e}_{i^*}^{(j^*,\beta)}\end{aligned}$$

$\forall\ (i,\beta) \in ([\ell+1]\setminus[i^*])\times\{0,1\}$,

$$\mathbf{t}_i^{(j^*,\beta)} = \begin{cases}-\mathbf{s}_{i-1}^{(j^*,\beta)}\cdot\mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j^*,\beta)} + \mathbf{s}_i^{(j^*,\beta)}\cdot\mathbf{B}_{i,\widetilde{x}_i}^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)} & \text{if } i \le \ell \\ -\mathbf{s}_\ell^{(j^*,\beta)}\cdot\mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)} & \text{if } i = \ell+1\end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Game $3.i^*.2$: This is identical to the previous game, except the $(i^*+1)^{th}$ key component in the j^{*th} strands is also hardwired. Below we describe it in detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
$$\forall\ i < i^*, \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2,$$
$$\forall\ i \geq i^*, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$
Also, let $\widetilde{n}_i = \begin{cases} \widetilde{n} - 4n & \text{for } i < i^* \\ \widetilde{n} & \text{for } i \geq i^* \end{cases}$, and set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j, \beta, b) \in S^{(i)}\right\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i, j, \beta, b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.
- **Challenge Phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let
$$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell], b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w] \right),$$
$$S^* = \hat{S}.$$
The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{array} \right).$$
Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}^*_{i,b}\}_{i\in[\ell], b\in\{0,1\}}\right)$.
- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
$$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array} \right).$$
     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell], b\in\{0,1\}}\right)$.

2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.
   (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,\\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,\\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,\\
&\forall\ (i,\beta) \in [i^*-1] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m,\\
&\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
\end{aligned}$$

   (c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after $i^* - 1$ steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times ([\lambda] \setminus \{j^*\}) \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)}$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^* \end{cases}$$

   Also, for $\beta \in \{0,1\}$, let $\mathbf{B}_{\ell+1,\widetilde{x}_{\ell+1}}^{(j^*,\beta)} = \mathbf{0}^{n \times m}$, and $\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)}$ denote the following vector.

$$\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)}.$$

   Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell + 1 \end{cases}$$

$$\begin{aligned}
\forall\ (i,\beta) \in [i^*-1] \times \{0,1\}, \quad & \mathbf{t}_i^{(j^*,\beta)} = \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)}\\
\forall\ \beta \in \{0,1\}, \quad & \mathbf{t}_{i^*}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} + \mathbf{e}_{i^*}^{(j^*,\beta)}\\
\forall\ \beta \in \{0,1\}, \quad & \mathbf{t}_{i^*+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\\
& \quad + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}_{i^*+1}^{(\beta)}} + \mathbf{s}_{i^*+1}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)} + \mathbf{e}_{i^*+1}^{(j^*,\beta)}
\end{aligned}$$

   $\forall\ (i,\beta) \in ([\ell+1] \setminus [i^*+1]) \times \{0,1\}$,

$$\mathbf{t}_i^{(j^*,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j^*,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j^*,\beta)} + \mathbf{s}_i^{(j^*,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)} & \text{if } i \le \ell \\ -\mathbf{s}_\ell^{(j^*,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)} & \text{if } i = \ell + 1 \end{cases}$$

   (d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Game $3.i^*.3$: This is identical to the previous game, except the matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ for strands $j = j^*$ and levels $i = i^*$ are **not** sampled along with other level i^* matrices, but instead they are sampled uniformly at random. Also, ciphertext components for level i^* are used to only target remaining matrices. Below we describe it in detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
  $$\forall\ i < i^* + 1, \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2,$$
  $$\forall\ i \geq i^* + 1, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$
  Also, let $\widetilde{n}_i = \begin{cases} \widetilde{n} - 4n & \text{for } i < i^* + 1 \\ \widetilde{n} & \text{for } i \geq i^* + 1 \end{cases}$, and set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j, \beta, b) \in S^{(i)}\right\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q),$$
  $$\forall\ (\beta, b) \in \{0,1\}^2, \quad \mathbf{B}_{i^*,b}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}.$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i, j, \beta, b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.
- **Challenge Phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let
  $$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),$$
  $$S^* = \hat{S}.$$
  The challenger then runs the Mixed-SubEnc routine (described in Figure 6) as
  $$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{matrix} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{matrix}\right).$$
  Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.
- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Figure 6) as
     $$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{matrix} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{matrix}\right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda-1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda]\setminus\{j^*\}$.
   (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,\\
&\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,\\
&\forall\ (j,\beta) \in [\lambda]\times\{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,\\
&\forall\ (i,\beta) \in [i^*-1]\times\{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m,\\
&\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
\end{aligned}$$

   (c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after i^*-1 steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1]\times([\lambda]\setminus\{j^*\})\times\{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}$$

$$\forall\ (i,\beta,b) \in [\ell]\times\{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^* \end{cases}$$

   Also, for $\beta \in \{0,1\}$, let $\mathbf{B}_{\ell+1,\widetilde{x}_{\ell+1}}^{(j^*,\beta)} = \mathbf{0}^{n\times m}$, and $\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)}$ denote the following vector.

$$\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)}.$$

   Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\begin{aligned}
\forall\ (i,\beta) \in [i^*-1]\times\{0,1\}, \quad \mathbf{t}_i^{(j^*,\beta)} &= \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)}\\
\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*}^{(j^*,\beta)} &= \widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} + \mathbf{e}_{i^*}^{(j^*,\beta)}\\
\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j^*,\beta)} &= -\sum_{\alpha=1}^{i^*}\left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\\
&\quad + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}_{i^*+1}^{(\beta)}} + \mathbf{s}_{i^*+1}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)} + \mathbf{e}_{i^*+1}^{(j^*,\beta)}
\end{aligned}$$

   $\forall\ (i,\beta) \in ([\ell+1]\setminus[i^*+1])\times\{0,1\}$,

$$\mathbf{t}_i^{(j^*,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j^*,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j^*,\beta)} + \mathbf{s}_i^{(j^*,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)} & \text{if } i \le \ell \\ -\mathbf{s}_\ell^{(j^*,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)} & \text{if } i = \ell+1 \end{cases}$$

   (d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Game 3.i^*.4 : This is identical to the previous game, except the i^{*th} level key component in j^{*th} strands is a uniformly random n length vector, i.e. all first i^* level components in j^{*th} strand are random elements. Also, we no longer sample the matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ for strands $j = j^*$ and levels $i = i^*$ at all. Below we describe it in detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
  $$\forall\ i < i^* + 1, \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2,$$
  $$\forall\ i \geq i^* + 1, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$
  Also, let $\widetilde{n}_i = \begin{cases} \widetilde{n} - 4n & \text{for } i < i^* + 1 \\ \widetilde{n} & \text{for } i \geq i^* + 1 \end{cases}$, and set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j, \beta, b) \in S^{(i)}\right\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i, j, \beta, b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.
- **Challenge Phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let
  $$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell], b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w] \right),$$
  $$S^* = \hat{S}.$$
  The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
  $$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{array} \right).$$
  Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell], b\in\{0,1\}}\right)$.
- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
     $$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array} \right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.
   (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,\\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,\\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,\\
&\forall\ (i,\beta) \in [i^*] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m,\\
&\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
\end{aligned}$$

   (c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs $\mathsf{BP}, \mathsf{BP}^{(\gamma)}$ after $i^* - 1$ steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times ([\lambda] \setminus \{j^*\}) \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^* \end{cases}$$

   Also, for $\beta \in \{0,1\}$, let $\mathbf{B}_{\ell+1,\widetilde{x}_{\ell+1}}^{(j^*,\beta)} = \mathbf{0}^{n\times m}$. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\begin{aligned}
\forall\ (i,\beta) \in [i^*] \times \{0,1\}, \quad \mathbf{t}_i^{(j^*,\beta)} &= \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)}\\
\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j^*,\beta)} &= -\sum_{\alpha=1}^{i^*}\left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\\
&\quad + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}_{i^*+1}^{(\beta)}} + \mathbf{s}_{i^*+1}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)} + \mathbf{e}_{i^*+1}^{(j^*,\beta)}
\end{aligned}$$

   $\forall\ (i,\beta) \in ([\ell+1] \setminus [i^*+1]) \times \{0,1\}$,

$$\mathbf{t}_i^{(j^*,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j^*,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j^*,\beta)} + \mathbf{s}_i^{(j^*,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)} & \text{if } i \le \ell \\ -\mathbf{s}_\ell^{(j^*,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)} & \text{if } i = \ell+1 \end{cases}$$

   (d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

$\mathsf{Game}\ 4:$ This is identical to the previous game, i.e. $\mathsf{Game}\ 3.\ell.4$. For the ease of exposition, we describe in it detail below.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
  $$\forall\ i \in [\ell], \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2.$$
  Also, let $\widetilde{n}_i = \widetilde{n} - 4n$ for all $i \in [\ell]$, and set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2\ :\ (j, \beta, b) \in S^{(i)}\right\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i, j, \beta, b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.
- **Challenge Phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let
  $$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell], b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),$$
  $$S^* = \hat{S}.$$
  The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
  $$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\begin{pmatrix} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{pmatrix}.$$
  Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell], b\in\{0,1\}}\right)$.
- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
     $$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\begin{pmatrix} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{pmatrix}.$$
     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell], b\in\{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

(b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
\forall\ (i,j,\beta) &\in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
\forall\ (i,j,\beta) &\in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
\forall\ (j,\beta) &\in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
\forall\ (i,\beta) &\in [\ell] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.
\end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after ℓ steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times ([\lambda] \setminus \{j^*\}) \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)}$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^* \end{cases}$$

For $v \in [w]$, let $\mathbf{P}_{\ell+1,v}^{(\beta^*)}$ be the top level matrices chosen while computing challenge ciphertext. Similarly, $\mathbf{P}_{\ell+1,v}^{(1-\beta^*)}$ be the top level matrices chosen while computing query ciphertext. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\forall\ (i,\beta) \in [\ell] \times \{0,1\}, \quad \mathbf{t}_i^{(j^*,\beta)} = \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)}$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta)}}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Next, we have a sequence of ℓ hybrid experiments $\mathsf{Game}\ 4.i^*$ for $i^* = 1$ to ℓ.

$\mathsf{Game}\ 4.i^*$: This is identical to the previous game, except the challenger uses $\mathsf{Mixed\text{-}SubEnc}^*$ routine to generate the first i^* components of both the challenge as well as query ciphertext.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows—

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit

Mixed-SubEnc*

Inputs:

- Tag tag, Level α, Set $S \subseteq [\ell] \times [\lambda] \times \{0,1\}^2$, Matrices $\left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}$, Trapdoors $\{T_i\}_{i\in[\ell]}$,

Output: Matrices $\{\mathbf{U}_0, \mathbf{U}_1\}$.

Execution: Let S_α denote the following set:

$$S_\alpha = \left\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2 \text{ such that } (\alpha, j, \beta, b) \in S\right\}.$$

Sample matrices $\left\{\mathbf{D}_b^{(j,\beta)}, \widetilde{\mathbf{D}}_b^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha}$ as:

$$\forall\ (j,\beta,b) \in S_\alpha, \qquad \begin{array}{l} \mathbf{D}_b^{(j,\beta)} = \begin{cases} \mathbf{C}_{\alpha,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 0 \\ (\leftarrow \mathbb{Z}_q^{n\times m}) & \text{otherwise.} \end{cases} \\ \widetilde{\mathbf{D}}_b^{(j,\beta)} = \begin{cases} \mathbf{C}_{\alpha,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 1 \\ (\leftarrow \mathbb{Z}_q^{n\times m}) & \text{otherwise.} \end{cases} \end{array}$$

Sample $2w$ matrices matrices $\left\{\mathbf{Q}_v, \widetilde{\mathbf{Q}}_v\right\}_{v\in[w]}$ as:

$$\forall\ v \in [w], \qquad \begin{array}{l} \mathbf{Q}_v \leftarrow \mathbb{Z}_q^{n\times m}, \\ \widetilde{\mathbf{Q}}_v \leftarrow \mathbb{Z}_q^{n\times m}. \end{array}$$

Let matrices $\mathbf{M}, \mathbf{W}, \widetilde{\mathbf{W}}$ represent the following $(|S_\alpha| + w)n \times m$ dimension matrices:

$$\mathbf{M} = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} \left\{\mathbf{D}_b^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \{\mathbf{Q}_{i,v}\}_{v\in[w]} \end{bmatrix}, \quad \widetilde{\mathbf{W}} = \begin{bmatrix} \left\{\widetilde{\mathbf{D}}_b^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \left\{\widetilde{\mathbf{Q}}_{i,v}\right\}_{v\in[w]} \end{bmatrix}.$$

Run the $\mathsf{EnSamplePre}$ to compute matrices $\{\mathbf{U}_0, \mathbf{U}_1\}$ as

$$\begin{array}{l} \mathbf{U}_0 \leftarrow \mathsf{EnSamplePre}(\mathbf{M}, T_\alpha, \sigma_{\mathsf{pre}}, \mathbf{W}), \\ \mathbf{U}_1 \leftarrow \mathsf{EnSamplePre}(\mathbf{M}, T_\alpha, \sigma_{\mathsf{pre}}, \widetilde{\mathbf{W}}). \end{array}$$

Figure 7: Routine Mixed-SubEnc*

strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i \in [\ell], \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2.$$

Also, let $\widetilde{n}_i = \widetilde{n} - 4n$ for all $i \in [\ell]$, and set $\hat{S} = \left\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ :\ (j,\beta,b) \in S^{(i)}\right\}$.

2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.
4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

$$\begin{aligned}
&\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),\\
&S^* = \hat{S}.
\end{aligned}$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ and $\mathsf{Mixed\text{-}SubEnc}^*$ routines (described in Figure 6 and Figure 7) as

$$\begin{aligned}
&\forall\ \alpha \in [i^*], \quad \left(\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}^*\left(\begin{array}{c}\mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*},\\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}\end{array}\right),\\
&\forall\ \alpha \in [\ell]\setminus[i^*], \quad \left(\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*},\\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)}\end{array}\right).
\end{aligned}$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}^*_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ and $\mathsf{Mixed\text{-}SubEnc}^*$ routines (described in Figure 6 and Figure 7) as

$$\begin{aligned}
&\forall\ \alpha \in [i^*], \quad \left(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}^*\left(\begin{array}{c}\mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S},\\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}\end{array}\right),\\
&\forall\ \alpha \in [\ell]\setminus[i^*], \quad \left(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S},\\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}\end{array}\right).
\end{aligned}$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda]\setminus\{j^*\}$.
     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,\\
&\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,\\
&\forall\ (j,\beta) \in [\lambda]\times\{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,\\
&\forall\ (i,\beta) \in [\ell]\times\{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.
\end{aligned}$$

     (c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after ℓ steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1]\times([\lambda]\setminus\{j^*\})\times\{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}$$

$$\forall\ (i,\beta,b) \in [\ell]\times\{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases}\mathbf{U}^*_{i,b} & \text{if } \beta = \beta^*\\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*\end{cases}$$

For $v \in [w]$, let $\mathbf{P}^{(\beta^*)}_{\ell+1,v}$ be the top level matrices chosen while computing challenge ciphertext. Similarly, $\mathbf{P}^{(1-\beta^*)}_{\ell+1,v}$ be the top level matrices chosen while computing query ciphertext. [28] Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}^{(j,\beta)}_{1,\widetilde{x}_1} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}^{(j,\beta)}_{i-1,\widetilde{x}_{i-1}} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}^{(j,\beta)}_{i,\widetilde{x}_i} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}^{(j,\beta)}_{\ell,\widetilde{x}_\ell} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\forall\ (i,\beta) \in [\ell] \times \{0,1\}, \quad \mathbf{t}_i^{(j^*,\beta)} = \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)}$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \widetilde{\mathbf{s}} \cdot \mathbf{P}^{(\beta)}_{\ell+1,\mathsf{st}^{(\beta)}_{\ell+1}} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Game 5 : This is identical to the previous game, i.e. Game 4.ℓ. For the ease of exposition, we describe in it detail below.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows—

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
  $$\forall\ i \in [\ell], \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2.$$
  Also, let $\widetilde{n}_i = \widetilde{n} - 4n$ for all $i \in [\ell]$, and set $\hat{S} = \left\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j,\beta,b) \in S^{(i)}\right\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let
$$\mathsf{BP}^{(\gamma)} = \left(\left\{ \pi_{i,b}^{(\gamma)} : [w] \to [w] \right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w] \right),$$
$$S^* = \hat{S}.$$

[28] Technically, in Game 4.$(\ell+1)$, $\mathbf{P}^{(\beta)}_{\ell+1,\mathsf{st}^{(\beta)}_{\ell+1}}$ is not sampled during Mixed-SubEnc*. For that experiment, we will assume these matrices are chosen for the first key query and used for all remaining keys.

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}^*$ routine (described in Figure 7) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}^* \left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S^*}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]} \end{array} \right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}^*$ routine (described in Figure 7) as

     $$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}^* \left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]} \end{array} \right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \left\{\mathbf{U}_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.
     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}$ as follows

     $$\begin{aligned} &\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\ (j,\beta) \in [\lambda]\times\{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\ &\forall\ (i,\beta) \in [\ell]\times\{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m. \end{aligned}$$

     (c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after ℓ steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

     $$\Gamma = [\ell+1]\times([\lambda]\setminus\{j^*\})\times\{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}$$

     $$\forall\ (i,\beta,b) \in [\ell]\times\{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}^*_{i,b} & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^* \end{cases}$$

     For the first key query, it samples matrices $\mathbf{P}^{(\beta)}_{\ell+1,v}$ for $v \in [w], \beta \in \{0,1\}$ as follows. [29]

     $$\mathbf{P}^{(\beta)}_{\ell+1,\mathsf{st}^{(\beta)}_{\ell+1}} = \begin{cases} \mathbf{0}^{n\times m} & \text{if } v = \mathsf{rej} \\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise.} \end{cases}$$

[29] Recall, as defined in Game 4.$(\ell+1)$, these matrices are sampled only for the first key query, and all remaining key queries use the same matrices.

Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\forall\ (i,\beta) \in [\ell] \times \{0,1\}, \quad \mathbf{t}_i^{(j^*,\beta)} = \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)}$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}^{(\beta)}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta)}} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

8.4.2 Indistinguishability of Hybrid Games in Section 8.4

We will now show that the hybrid experiments described above are computationally indistinguishable. For any PPT adversary $\mathcal{A}$, let $\mathsf{Adv}_{\mathcal{A},x}(\cdot)$ denote the advantage of $\mathcal{A}$ in Game x.

Lemma 8.3. There exists a negligible function negl$(\cdot)$ such that for any adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},0}(\lambda) - \mathsf{Adv}_{\mathcal{A},1}(\lambda) \leq \text{negl}(\lambda)$.

Proof. The only difference between Game 0 and Game 1 is that the challenger aborts if $\mathsf{tag}^* = \mathsf{tag}$. The probability of this event is $2^{-\lambda}$, and it is independent of the adversary's choice of (k, w, L) and $\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)}$ in the setup phase. As a result, for any adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A},0}(\lambda) - \mathsf{Adv}_{\mathcal{A},1}(\lambda) \leq 2^{-\lambda}$. ■

Lemma 8.4. For any adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},1}(\lambda) = \mathsf{Adv}_{\mathcal{A},2}(\lambda)$.

Proof. The only difference between the two hybrids is with respect to the keys. In Game 1, for each key query, the challenger chooses $\lambda - 1$ uniformly random vectors $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j < \lambda$, and sets $\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda-1]} \mathbf{y}^{(j)}$. In Game 2, the challenger chooses $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$, and sets $\mathbf{y}^{(j^*)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}$. Fix all $\mathbf{y}^{(j)}$ for $j \notin \{j^*, \lambda\}$ and $\widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}$. Then, the following distributions are identical:

$$\left\{ \left(\mathbf{y}^{(j^*)}, \mathbf{y}^{(\lambda)}\right) : \begin{array}{l} \mathbf{y}^{(j^*)} \leftarrow \mathbb{Z}_q^m; \\ \mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda-1]} \mathbf{y}^{(j)} \end{array} \right\} \equiv \left\{ \left(\mathbf{y}^{(j^*)}, \mathbf{y}^{(\lambda)}\right) : \begin{array}{l} \mathbf{y}^{(\lambda)} \leftarrow \mathbb{Z}_q^m; \\ \mathbf{y}^{(j^*)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)} \end{array} \right\}$$

This implies that the distributions in Game 1 and Game 2 are identical.

■

Lemma 8.5. For any PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.1.1}(\lambda) \leq \text{negl}(\lambda)$.

Proof. Let us first consider the differences between Game 2 and Game 3.1.1. The setup and challenge phase is identical in both games. The post-challenge ciphertext query is also handled identically in both games. The only difference in the two games is with respect to the key queries. In particular, for each key query x, the key components $\{\mathbf{t}_1^{(j^*,\beta)}\}_{\beta\in\{0,1\}}$ are computed differently in the two games. In Game 2, the challenger sets $\mathbf{t}_1^{(j^*,\beta)} = -\widetilde{\mathbf{y}} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} + \mathbf{s}_1^{(j^*,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j^*,\beta)} + \mathbf{e}_1^{(j^*,\beta)}$ while in Game 3.1.1, it sets $\mathbf{t}_1^{(j^*,\beta)} = -\widetilde{\mathbf{y}} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} + \mathbf{s}_1^{(j^*,\beta)} \cdot$

$\mathbf{B}^{(j^*,\beta)}_{1,\widetilde{x}_1} + \widetilde{\mathbf{e}}^{(j^*,\beta)}_1 + \mathbf{e}^{(j^*,\beta)}_1$. Using the smuding lemma (Lemma 2.1), since $\sigma_{\mathsf{big}}/\sigma_{\mathsf{lwe}} \geq 2^\lambda$, we can argue that there exists a negligible function $\mathrm{negl}_{\mathsf{smud}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $m \in \mathbb{N}$, $\mathsf{SD}(\mathcal{D}_1, \mathcal{D}_2) \leq 2m \cdot \mathrm{negl}_{\mathsf{smud}}(\lambda)$, where

$$\mathcal{D}_1 \equiv \left\{ \left(\mathbf{e}^{(j^*,0)}_1, \mathbf{e}^{(j^*,1)}_1\right) : \ \mathbf{e}^{(j^*,\beta)}_1 \leftarrow \chi^m_{\mathsf{big}} \text{ for } \beta \in \{0,1\} \ \right\};$$
$$\mathcal{D}_2 \equiv \left\{ \left(\widetilde{\mathbf{e}}^{(j^*,0)}_1 + \mathbf{e}^{(j^*,0)}_1, \widetilde{\mathbf{e}}^{(j^*,0)}_1 + \mathbf{e}^{(j^*,0)}_1\right) : \begin{array}{l} \mathbf{e}^{(j^*,\beta)}_1 \leftarrow \chi^m_{\mathsf{big}} \text{ for } \beta \in \{0,1\}; \\ \widetilde{\mathbf{e}}^{(j^*,\beta)}_1 \leftarrow \chi^m_{\mathsf{lwe}} \text{ for } \beta \in \{0,1\}; \end{array} \right\}$$

As a result, if an adversary $\mathcal{A}$ makes $q_{\mathsf{keys}}(\lambda)$ key queries, then for any $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.1.1}(\lambda) \leq q_{\mathsf{keys}}(\lambda) \cdot (2m \cdot \mathrm{negl}_{\mathsf{smud}}(\lambda))$.

∎

Lemma 8.6. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.1}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) \leq \mathrm{negl}(\lambda)$.

Proof. The main difference in these two games is in the key generation phase. In particular, for each key, the terms $\left(\mathbf{t}^{(j^*,0)}_{i^*+1}, \mathbf{t}^{(j^*,1)}_{i^*+1}\right)$ are computed differently in both games. In Game $3.i^*.1$, $\mathbf{t}^{(j^*,\beta)}_{i^*+1} = -\mathbf{s}^{(j^*,\beta)}_{i^*} \cdot \mathbf{C}^{(j^*,\beta)}_{i^*,\widetilde{x}_{i^*}} + \mathbf{s}^{(j^*,\beta)}_{i^*+1} \cdot \mathbf{B}^{(j^*,\beta)}_{i^*,\widetilde{x}_{i^*+1}} + \mathbf{e}^{(j^*,\beta)}_{i^*+1}$, while in Game $3.i^*.2$, the challenger sets

$$\begin{aligned} \mathbf{t}^{(j^*,\beta)}_{i^*+1} &= -\sum_{\alpha=1}^{i^*}\left(\widetilde{\mathbf{t}}^{(j^*,\beta)}_{\alpha} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}^{(\beta)}_{i^*+1}} + \mathbf{s}^{(j^*,\beta)}_{i^*+1} \cdot \mathbf{B}^{(j^*,\beta)}_{i^*+1,\widetilde{x}_{i^*+1}} + \mathbf{e}^{(j^*,\beta)}_{i^*+1} \\ &= -\mathbf{s}^{(j^*,\beta)}_{i^*} \cdot \mathbf{C}^{(j^*,\beta)}_{i^*,\widetilde{x}_{i^*}} + \mathbf{s}^{(j^*,\beta)}_{i^*+1} \cdot \mathbf{B}^{(j^*,\beta)}_{i^*,\widetilde{x}_{i^*+1}} + \mathbf{e}^{(j^*,\beta)}_{i^*+1} + \widetilde{\mathbf{e}}^{(j^*,\beta)}_{i^*+1} \cdot \mathbf{U}^{(\beta)}_{i^*,\widetilde{x}_{i^*}} + \widetilde{\mathbf{s}} \cdot \mathbf{E} \end{aligned}$$

where in the second equality, $\widetilde{\mathbf{e}}^{(j^*,\beta)}_{i^*+1} \leftarrow \chi_{\mathsf{lwe}}$, $\widetilde{\mathbf{s}} \leftarrow \chi^n_s$, $\mathbf{E}$ is sampled by $\mathsf{Mixed\text{-}SubEnc}$ from $\chi^{n\times m}_{\mathsf{appr}}$. The second equality follows by substituting the value of $\widetilde{\mathbf{t}}^{(j^*,\beta)}_{i^*} = -\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}^{(j^*,\beta)}_{\alpha} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}^{(\beta)}_{i^*}} + \mathbf{s}^{(j^*,\beta)}_{i^*} \cdot \mathbf{B}^{(j^*,\beta)}_{i^*,\widetilde{x}_{i^*}} + \widetilde{\mathbf{e}}^{(j^*,\beta)}_{i^*}$ (note that this is how $\widetilde{\mathbf{t}}^{(j^*,\beta)}_{i^*}$ is defined in Game $3.i^*.2$).

To prove this lemma, we will use the following fact, which follows from the smudging lemma (Lemma 2.1).

Fact 8.1. Let $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{lwe}}$ be families of distributions over $\mathbb{Z}$ as defined in the construction. For any polynomials $n(\cdot), m(\cdot)$, there exists a negligible function $\mathrm{negl}_{8.1}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $m = m(\lambda)$, $\chi_{\mathsf{big}} = \chi_{\mathsf{big}}(\lambda)$, $\chi_s = \chi_s(\lambda)$, $\chi_{\mathsf{appr}} = \chi_{\mathsf{appr}}(\lambda)$, $\chi_{\mathsf{lwe}} = \chi_{\mathsf{lwe}}(\lambda)$, $\mathsf{SD}(\mathcal{D}_1, \mathcal{D}_2) \leq \mathrm{negl}_{8.1}(\lambda)$, where

$$\mathcal{D}_1 = \{\mathbf{e} : \mathbf{e} \leftarrow \chi_{\mathsf{big}}\}; \quad \mathcal{D}_2 = \left\{ \mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3 : \begin{array}{l} \mathbf{e}_1 \leftarrow \chi^m_{\mathsf{big}}; \mathbf{s} \leftarrow \chi^n_s; \mathbf{E} \leftarrow \chi^{n\times m}_{\mathsf{appr}}; \\ \mathbf{e}_2 = \mathbf{s} \cdot \mathbf{E}; \mathbf{e}_3 \leftarrow \chi^m_{\mathsf{lwe}} \end{array} \right\}$$

As a result, if an adversary $\mathcal{A}$ makes $q_{\mathsf{keys}}(\lambda)$ key queries, then for any $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.1}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) \leq q_{\mathsf{keys}}(\lambda) \cdot (2 \cdot \mathrm{negl}_{8.1}(\lambda))$.

∎

Lemma 8.7. Assuming the trapdoor generation algorithms $\mathsf{LT}_{\mathsf{en}}$ satisfy $(q, \sigma_{\mathsf{pre}})$-row removal property, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $i^* \in [\ell]$, $\mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda) \leq \mathrm{negl}(\lambda)$.

Proof. First, let us consider the differences between Game $3.i^*.2$ and Game $3.i^*.3$.

1. Set $S^{(i^*)}$: In Game $3.i^*.2$, the challenger sets $S^{(i^*)} = [\lambda] \times \{0,1\}^2$, while in Game $3.i^*.3$, $S^{(i^*)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2$ ($\mathsf{tag}^*, \mathsf{tag}$ are chosen at the start of the security game, so j^* is well defined here). Also, $\widetilde{n}_{i^*} = \widetilde{n} = (4\lambda + w)n$ in Game $3.i^*.2$, while $\widetilde{n}_{i^*} = \widetilde{n} - 4n$ in Game $3.i^*.3$.

2. $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i=i^*}$ matrices : In $\mathsf{Game}\ 3.i^*.2$, the challenger chooses $(\mathbf{M}_{i^*}, T_{i^*}) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q)$, while in $\mathsf{Game}\ 3.i^*.3$, it chooses $(\mathbf{M}_{i^*}, T_{i^*}) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}-4n}, 1^m, q)$. As a result, in $\mathsf{Game}\ 3.i^*.2$, it derives all $\left\{\mathbf{B}_{i^*,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2}$ from $\mathbf{M}_{i^*}$. In $\mathsf{Game}\ 3.i^*.3$, the challenger chooses $\left\{\mathbf{B}_{i^*,b}^{(j^*,\beta)}\right\}_{b,\beta\in\{0,1\}}$ uniformly at random, while the remaining are derived from $\mathbf{M}_{i^*}$.

3. Ciphertexts: Since the set $S^{(i^*)}$ is different in both games, the challenge and query ciphertexts are constructed differently in both games.

Let us now discuss why the row removal property is applicable here. In particular, we will focus on $\left(\mathbf{U}_{i^*,0}^*, \mathbf{U}_{i^*,1}^*, \mathbf{U}_{i^*,0}, \mathbf{U}_{i^*,1}\right)$. In $\mathsf{Game}\ 3.i^*.2$, each of these four matrices maps $\left[\mathbf{B}_{i^*,0}^{(j^*,0)} \mid \mathbf{B}_{i^*,1}^{(j^*,0)} \mid \mathbf{B}_{i^*,0}^{(j^*,1)} \mid \mathbf{B}_{i^*,1}^{(j^*,1)}\right]$ to a uniformly random matrix. To see why, let us suppose $\mathsf{tag}_{j^*}^* = \beta^*$ and $\mathsf{tag}_{j^*} = 1-\beta^*$. Then

- $\mathbf{B}_{i^*,0}^{(j^*,\beta^*)} \cdot \mathbf{U}_{i^*,0}^* = \mathbf{C}_{i^*,0}^{(j^*,\beta^*)}$, the rest are mapped to random matrices.
- $\mathbf{B}_{i^*,1}^{(j^*,\beta^*)} \cdot \mathbf{U}_{i^*,1}^* = \mathbf{C}_{i^*,1}^{(j^*,\beta^*)}$, the rest are mapped to random matrices.
- $\mathbf{B}_{i^*,0}^{(j^*,1-\beta^*)} \cdot \mathbf{U}_{i^*,0} = \mathbf{C}_{i^*,0}^{(j^*,1-\beta^*)}$, the rest are mapped to random matrices.
- $\mathbf{B}_{i^*,1}^{(j^*,1-\beta^*)} \cdot \mathbf{U}_{i^*,1} = \mathbf{C}_{i^*,1}^{(j^*,1-\beta^*)}$, the rest are mapped to random matrices.

Also, it is important to note that the $\left\{\mathbf{C}_{i^*,b}^{(j^*,\beta)}\right\}_{b,\beta\in\{0,1\}}$ are not used for responding to key generation queries. Therefore, we can use the row removal property to remove the rows corresponding to $\mathbf{B}_{i^*,\beta}^{(j^*,b)}$ from the level i^* matrices.

Suppose, on the contrary, there exists an adversary $\mathcal{A}$ and a non-negligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda) \geq \eta(\lambda)$. We will use this adversary to build a reduction algorithm $\mathcal{B}$ that breaks the $(q, \sigma_{\mathsf{pre}})$-row removal property of $\mathsf{LT}_{\mathsf{en}}$.

The reduction algorithm first receives functionality index (k, w, L) from $\mathcal{A}$. Depending on the functionality index, the reduction algorithm sets $\ell = k \cdot L$, n, m, $\widetilde{n} = (4\lambda + w)n$ as in $\mathsf{Game}\ 3.i^*.2$ (and $\mathsf{Game}\ 3.i^*.3$) and sends these parameters to $\mathcal{A}$.

The reduction algorithm chooses $\mathsf{tag}^*, \mathsf{tag}$ and defines j^* as the first index where the two tags differ. For all $i \neq i^*$, $\mathcal{B}$ defines sets $S^{(i)}$, and samples $\left\{\left\{\mathbf{B}_{i,\beta}^{(j,b)}\right\}_{(j,\beta,b)\in S^{(i)}}, \{\mathbf{P}_{i,v}\}_{v\in[w]}, T_i\right\}_{i\neq i^*}$ as in $\mathsf{Game}\ 3.i^*.2$ (and $\mathsf{Game}\ 3.i^*.3$). The reduction algorithm defines a set $S_{\mathcal{B}}$ which represents the set of rows that are removed in the transition between the two games. Formally, the reduction algorithm defines the following sets

$$\mathsf{pos} = \{j : b, \beta \in \{0,1\}, (i^*, j^*, b, \beta) \text{ is at position } j \text{ in the set } \{i^*\} \times [\lambda] \times \{0,1\}^2\}$$

$$S_{\mathcal{B}} = \bigcup_{j\in\mathsf{pos}} \{n(j-1)+1, n(j-1)+2, \ldots, nj\}.$$

It sends $1^{\widetilde{n}}, 1^m, S_{\mathcal{B}}$ to the row removal property challenger. It receives $\mathbf{A}$ from the challenger, which it parses as

$$\mathbf{A} = \begin{bmatrix} \left\{\mathbf{B}_{i^*,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i^*,v}\}_{v\in[w]} \end{bmatrix}$$

The reduction algorithm also chooses $(4\lambda + w - 4)$ matrices $\{\mathbf{C}_{i,b}^{(j,\beta)}\}_{i\neq i^*, j\neq j^*}$ uniformly at random from $\mathbb{Z}_q^{n\times m}$.

Next, it receives the challenge programs $\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)}$. It chooses $\gamma \leftarrow \{0,1\}$. For all $i \neq i^*$, it computes $(\mathbf{U}_{i,0}^*, \mathbf{U}_{i,1}^*)$ components by itself (this step is identical in both games). For $i = i^*$, it uses the row removal property challenger. It sets matrices $\mathbf{D}_b^{(j,\beta)}$ and $\widetilde{\mathbf{D}}_b^{(j,\beta)}$ as below.

$$\forall\ (j,\beta,b)\in([\lambda]\setminus\{j^*\})\times\{0,1\}^2,\qquad \begin{aligned}\mathbf{D}_b^{(j,\beta)} &= \begin{cases}\mathbf{C}_{i,b}^{(j,\beta)} & \text{if } \beta=\mathsf{tag}_j \text{ and } b=0\\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise.}\end{cases}\\ \widetilde{\mathbf{D}}_b^{(j,\beta)} &= \begin{cases}\mathbf{C}_{i,b}^{(j,\beta)} & \text{if } \beta=\mathsf{tag}_j \text{ and } b=1\\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise.}\end{cases}\end{aligned}$$

Next, it sets matrices $\left\{\mathbf{Q}_{i,v}\right\}_{v\in[w]}$, $\left\{\widetilde{\mathbf{Q}}_{i,v}\right\}_{v\in[w]}$ as in Figure 6, and sets matrices $\mathbf{W}$ and $\widetilde{\mathbf{W}}$ as

$$\mathbf{W}=\begin{bmatrix}\left\{\mathbf{D}_b^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha}\\ \left\{\mathbf{Q}_{i,v}\right\}_{v\in[w]}\end{bmatrix},\quad \widetilde{\mathbf{W}}=\begin{bmatrix}\left\{\widetilde{\mathbf{D}}_b^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha}\\ \left\{\widetilde{\mathbf{Q}}_{i,v}\right\}_{v\in[w]}\end{bmatrix}.$$

It sends them as queries to the row removal challenger (note that $\mathbf{C}_{i^*,b}^{(j^*,\beta)}$ is not required for defining $\mathbf{W}$ and $\widetilde{\mathbf{W}}$). The challenger responds by sending $\mathbf{U}_{i^*,0}^*$ and $\mathbf{U}_{i^*,1}^*$ respectively. The reduction algorithm forwards $\left\{\left(\mathbf{U}_{i,0}^*,\mathbf{U}_{i,1}^*\right)\right\}_{i\in[\ell]}$ to the adversary. The ciphertext query is handled similarly, and the reduction algorithm receives $\{(\mathbf{U}_{i,0},\mathbf{U}_{i,1})\}_{i\in[\ell]}$, which it forwards to $\mathcal{A}$ (the remaining ciphertext components can be computed by the reduction algorithm).

Next, the adversary sends polynomially many key queries. Note that the keys are generated in an identical manner in both games. Moreover, these keys can be generated without having $\{\mathbf{C}_{i^*,b}^{(j^*,\beta)}\}_{b,\beta}$ and the trapdoor for $\mathbf{M}_{i^*}$.

Finally, the adversary sends its guess, which the reduction algorithm forwards to the row removal property challenger. Clearly, if the row removal challenger chose $b=0$, then the reduction algorithm perfectly simulates Game $3.i^*.3$. If the challenger chose $b=1$, then the reduction algorithm perfectly simulates Game $3.i^*.2$ (here we use the fact that in Game $3.i^*.2$, each of the matrices $\left\{\mathbf{U}_{i^*,0}^*,\mathbf{U}_{i^*,1}^*,\mathbf{U}_{i^*,0},\mathbf{U}_{i^*,1}\right\}$ maps $\left[\mathbf{B}_{i^*,0}^{(j^*,0)}\mid\mathbf{B}_{i^*,1}^{(j^*,0)}\mid\mathbf{B}_{i^*,0}^{(j^*,1)}\mid\mathbf{B}_{i^*,1}^{(j^*,1)}\right]$ to a uniformly random matrix.

Therefore, the reduction algorithm has advantage at least $\eta(\cdot)$. ■

Lemma 8.8. Assuming the $(n,q,\sigma_{\mathsf{lwe}})$-LWE assumption, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda\in\mathbb{N}$, $i^*\in[\ell]$, $\mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda)-\mathsf{Adv}_{\mathcal{A},3.i^*.4}(\lambda)\le\mathrm{negl}(\lambda)$.

Proof. In Game $3.i^*.3$, for each key query x, for each $\beta\in\{0,1\}$, the component $\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)}=-\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)}\cdot\prod_{\delta=\alpha}^{i^*-1}\mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right)-\widetilde{\mathbf{y}}\cdot\prod_{\delta=1}^{i^*-1}\mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}+\widetilde{\mathbf{s}}\cdot\mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}}+\mathbf{s}_{i^*}^{(j^*,\beta)}\cdot\mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)}+\widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)}$. In Game $3.i^*.4$, $\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)}\leftarrow\mathbb{Z}_q^m$. Let $q_{\mathsf{keys}}=q_{\mathsf{keys}}(\lambda)$ denote the number of keys queried by $\mathcal{A}(1^\lambda)$. To prove that these two games are computationally indistinguishable, we will define q_{keys} hybrid experiments.

Hybrid $H_{o,0}$ for $o\in\{0,1,\ldots,q_{\mathsf{keys}}\}$: In this hybrid, for the first o keys, the $\widetilde{\mathbf{t}}_{i^*}^{(j^*,0)}$ components are sampled uniformly at random in the first o queries, while the $\widetilde{\mathbf{t}}_{i^*}^{(j^*,1)}$ components are sampled as in Game $3.i^*.3$. For the remaining $q_{\mathsf{keys}}-o$ key queries, the keys are generated as in Game $3.i^*.3$ in the remaining queries.

Hybrid $H_{o,1}$ for $o\in\{0,1,\ldots,q_{\mathsf{keys}}\}$: In this hybrid, for all keys, the $\widetilde{\mathbf{t}}_{i^*}^{(j^*,0)}$ components are sampled uniformly at random. For the first o queries, the $\widetilde{\mathbf{t}}_{i^*}^{(j^*,1)}$ components are sampled uniformly at random, while the remaining are sampled as in Game $3.i^*.3$.

Clearly, $H_{0,0}$ corresponds to $\mathsf{Game}\ 3.i^*.3$, while $H_{q_{\mathsf{keys}},1}$ is identical to $\mathsf{Game}\ 3.i^*.4$, and $H_{q_{\mathsf{keys}},0} \equiv H_{0,1}$. Let $a_{\mathcal{A},i,b}(\lambda)$ denote the advantage of $\mathcal{A}$ in $H_{i,b}$.

Claim 8.4. Assuming the $(n, q, \sigma_{\mathsf{lwe}})$-LWE assumption, for any PPT adversary $\mathcal{A}$ making $q_{\mathsf{keys}}(\cdot)$ key queries, there exists a negligible function $\mathsf{n}_{o,0}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ and all indices $o \in [q_{\mathsf{keys}}]$, $a_{\mathcal{A},o-1,0} - a_{\mathcal{A},o,0} \leq \mathsf{n}_{o,0}(\lambda)$.

Proof. Suppose there exists an adversary making q_{keys} key queries, and a non-negligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, there exists an index $o \in [q_{\mathsf{keys}}]$ such that $a_{\mathcal{A},o-1,0} - a_{\mathcal{A},o,0} \geq \eta(\lambda)$. We will use $\mathcal{A}$ to build a reduction algorithm $\mathcal{B}$ that breaks the $(n, q, \sigma_{\mathsf{lwe}})$-LWE assumption.

The reduction algorithm first receives $(1^k, 1^w, 1^L)$ from $\mathcal{A}$. It sets $\widetilde{n} = (4\lambda+w)\cdot n$. The reduction algorithm queries the LWE challenger m times, and receives $\{(\mathbf{a}_i, u_i)\}_{i\leq m}$. It sets a matrix $\mathbf{A} = [\mathbf{a}_1^T\ \mathbf{a}_2^T \ldots \mathbf{a}_m^T]$ (that is, $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$), and $\mathbf{u} = [u_1 u_2 \ldots u_m]$ (that is, $\mathbf{u} \in \mathbb{Z}_q^m$).

The reduction algorithm then chooses two tags $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$, and let j^* be the first index where they differ. Next, the reduction algorithm defines set $S^{(i)}$ for each i, set $\hat{S}$, matrices $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{(i,j,\beta,b)\in\hat{S}}$, $\{\mathbf{P}_{i,v}\}_{i\in[\ell],v\in[w]}$ and $\{T_i\}_{i\in[\ell]}$ as in $3.i^*.3$ (and $3.i^*.4$). Note that $(i^*, j^*, \beta, b) \notin \hat{S}$ for $b, \beta \in \{0,1\}$. The reduction algorithm chooses $\mathbf{B}_{i^*,b}^{(j^*,1)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $b \in \{0,1\}$. It sends the public parameters to $\mathcal{A}$.

The adversary sends two challenge functions $\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)}$, and a ciphertext query BP. Note that in $\mathsf{Game}\ 3.i^*.3$ (and $\mathsf{Game}\ 3.i^*.4$), the challenge and query ciphertext are computed identically, and the reduction algorithm has all the matrices/trapdoors required for computing the ciphertext components.

Next, after receiving the challenge ciphertext and the query ciphertext, the adversary queries for secret keys. For the first $o-1$ secret keys, the reduction algorithm responds as in $H_{o-1,0}$ (which is identical to the response in $H_{o,0}$). In particular, to handle these key queries, the reduction algorithm does not require $\mathbf{B}_{i^*,\beta}^{(j^*,b)}$, since in both hybrids, $\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m$. For the o^{th} query, the reduction algorithm receives $\mathbf{x} \in \{0,1\}^k$. It sets $\widetilde{\mathbf{x}}$ by repeating the input L times, and sets $\mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,0)} = \mathbf{A}$ (the LWE public matrix) and chooses $\mathbf{B}_{i^*,1-\widetilde{x}_{i^*}}^{(j^*,0)} \leftarrow \mathbb{Z}_q^{n\times m}$ (this might be used for the later key queries). The reduction algorithm sets

$$\widetilde{\mathbf{t}}_{i^*}^{(j^*,0)} = -\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,0)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(0)}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(0)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(0)}} + \mathbf{u}.$$

It computes $\widetilde{\mathbf{t}}_{i^*}^{(j^*,1)}$ as in $H_{o-1,0}$ (and $H_{o,0}$). All the remaining key queries are handled identically in $H_{o-1,0}$ and $H_{o,0}$, and the reduction algorithm has all the matrices required to compute them (in particular, after responding to the o^{th} query, all $\{\mathbf{B}_{i^*,b}^{(j^*,\beta)}\}_{(b,\beta)\in\{0,1\}^2}$ are well-defined).

Finally, the adversary sends its guess, and the reduction algorithm forwards it to the LWE challenger. Clearly, if $\mathbf{u}$ is a uniformly random vector, then so is the $\widetilde{\mathbf{t}}_{i^*}^{(j^*,0)}$ component for o^{th} query, and therefore $\mathcal{B}$ perfectly simulates $H_{o,0}$. If $\widetilde{\mathbf{t}}_{i^*}^{(j^*,0)} = \mathbf{s} \cdot \mathbf{B}_{i^*,\widetilde{x}_i}^{(j^*,0)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,0)}$, then the reduction algorithm implicitly sets $\mathbf{s}_{i^*}^{(j^*,0)} = \mathbf{s}$. Also, note that $\mathbf{s}_{i^*}^{(j^*,0)}$ is chosen afresh for each key query, and hence $\mathbf{s}$ will not be required anywhere else in simulating $H_{o-1,0}$. Therefore the reduction algorithm perfectly simulates $H_{o-1,0}$. As a result, it breaks the LWE assumption with advantage η.

■

Claim 8.5. Assuming the $(n, q, \sigma_{\mathsf{lwe}})$-LWE assumption, for any PPT adversary $\mathcal{A}$ making $q_{\mathsf{keys}}(\cdot)$ key queries, there exists a negligible function $\mathsf{n}_{o,1}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ and all indices $o \in [q_{\mathsf{keys}}]$, $a_{\mathcal{A},o-1,1} - a_{\mathcal{A},o,1} \leq \mathsf{n}_{o,1}(\lambda)$.

This proof is identical to the proof of Claim 8.4.

Using the above claims, it follows that for any PPT adversary, there exists a negligible function $\mathrm{negl}_{3.i^*.4}$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.4}(\lambda) \leq \mathrm{negl}_{3.i^*.4}(\lambda)$. ■

Lemma 8.9. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_{3.(i^*+1).1}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.4}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.(i^*+1).1}(\lambda) \leq \text{negl}_{3.(i^*+1).1}(\lambda)$.

Proof. The only difference between $\mathsf{Game}\ 3.i^*.4$ and $\mathsf{Game}\ 3.(i^*+1).1$ is that for each key query, the components $\left(\mathbf{t}_{i^*+1}^{(j^*,0)}, \mathbf{t}_{i^*+1}^{(j^*,1)}\right)$ are computed differently. In particular, in $\mathsf{Game}\ 3.(i^*+1).1$, the term $\widetilde{\mathbf{t}}_{i^*+1}^{(j^*,\beta)}$ has an additional term $\widetilde{\mathbf{e}}_{i^*+1}^{(j^*,\beta)}$ which is drawn from the χ_{lwe}^m distribution.

The proof of this lemma is identical to the proof of Lemma 8.5 by setting $\text{negl}_{3.(i^*+1).1}(\cdot) = q_{\mathsf{keys}}(\cdot) \cdot (2m \cdot \text{negl}_{\mathsf{smud}}(\cdot))$ (recall $\text{negl}_{\mathsf{smud}}(\cdot)$ is the negligible function given by Lemma 2.1). ∎

Next, we will look at $\mathsf{Game}\ 4.i$ for $i \in [\ell]$. For notational convenience, we call $\mathsf{Game}\ 4$ to be $\mathsf{Game}\ 4.0$. First, recall that $\mathsf{Game}\ 4.0$ is identical to $3.\ell.4$. Note that in $\mathsf{Game}\ 4.0$, the challenger uses $\{\mathbf{P}_{1,v}\}_{v\in[w]}$ only for computing $\left(\mathbf{U}_{1,0}^*, \mathbf{U}_{1,1}^*\right)$ (for the challenge ciphertext) and $(\mathbf{U}_{1,0}, \mathbf{U}_{1,1})$ (for the ciphertext query). In particular, it is not used in the key generation phase. More generally, for all $i \in [\ell]$, $\{\mathbf{P}_{i,v}\}_{v\in[w]}$ is used in $\mathsf{Game}\ 4.(i-1)$ only for computing $\left(\mathbf{U}_{i,0}^*, \mathbf{U}_{i,1}^*\right)$ and $(\mathbf{U}_{i,0}, \mathbf{U}_{i,1})$. This observation is useful for the following lemma.

Lemma 8.10. Assuming $\mathsf{LT}_{\mathsf{en}}$ satisfies the $(q, \chi_{\mathsf{appr}}, \sigma_{\mathsf{pre}})$-target switching property, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}_{4.i^*}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $i^* \in [\ell]$, $\mathsf{Adv}_{\mathcal{A},4.(i^*-1)} - \mathsf{Adv}_{\mathcal{A},4.i^*} \leq \text{negl}_{4.i^*}(\lambda)$.

Proof. The only difference between $\mathsf{Game}\ 4.(i^*-1)$ and $\mathsf{Game}\ 4.i^*$ is with respect to the matrices $\left(\mathbf{U}_{i^*,0}^*, \mathbf{U}_{i^*,1}^*\right)$ (in the challenge ciphertext) and $(\mathbf{U}_{i^*,0}, \mathbf{U}_{i^*,1})$ (in the ciphertext query). In $\mathsf{Game}\ 4.(i^*-1)$, these matrices are computed using $\mathsf{Mixed\text{-}SubEnc}$, while they are computed using $\mathsf{Mixed\text{-}SubEnc}^*$ in $\mathsf{Game}\ 4.i^*$. The only difference between the $\mathsf{Mixed\text{-}SubEnc}$ and $\mathsf{Mixed\text{-}SubEnc}^*$ ciphertext components is that the $\mathsf{Mixed\text{-}SubEnc}^*$ outputs map the $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ matrices to uniformly random matrices (instead of mapping to $\left\{\mathbf{P}_{i^*+1,\pi(v)}\right\}_{v\in[w]}$ as in $\mathsf{Mixed\text{-}SubEnc}$). An important point to note is that the $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ matrices are not used anywhere else in the security game in both games. In particular, note that $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ are not used for computing $\left\{\mathbf{U}_{i^*-1,b}^*, \mathbf{U}_{i^*-1,b}\right\}_{b\in\{0,1\}}$.

Suppose there exists an adversary $\mathcal{A}$ and a non-negligible function $\eta(\cdot)$ such that $\mathsf{Adv}_{4.i^*-1}(\lambda) - \mathsf{Adv}_{4.i^*}(\lambda) \geq \eta(\lambda)$. We will construct a reduction algorithm that breaks the target switching property with advantage $\eta(\cdot)$.

Setup Phase The reduction algorithm first performs the following steps from the setup phase, which are common for both $\mathsf{Game}\ 4.(i^*-1)$ and $\mathsf{Game}\ 4.i^*$. It defines $\widetilde{n}_i$, $S^{(i)}$ for all $i \in [\ell]$, chooses $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$ and defines j^* as the first index where tag^* and tag differ. Next, it defines $\left\{\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}}, \{\mathbf{P}_{i,v}\}_{v\in[w]}, T_i\right\}_{i\neq i^*}$ as in $\mathsf{Game}\ 4.(i^*-1)$. It also defines $\hat{S}$ and chooses $\left\{\mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in\hat{S}}$ as in $\mathsf{Game}\ 4.(i^*-1)$.

The reduction algorithm sets $\widetilde{k} = \widetilde{n}_i - w \cdot n$ and queries the target switching property challenger by sending $1^{\widetilde{n}_i}, 1^m$ and set $S_{\mathcal{B}} = [\widetilde{k}]$. It receives a matrix $\mathbf{A} \in \mathbb{Z}_q^{\widetilde{k}\times m}$, and parses $\mathbf{A}$ as follows :

$$\mathbf{A} = \left[\left\{\mathbf{B}_{j,b}^{(1,\beta)}\right\}_{(j,\beta,b)\in S^{(i^*)}}\right]$$

Challenge Phase The reduction algorithm receives $\mathsf{BP}^{(1)}$, $\mathsf{BP}^{(2)}$. It chooses $\gamma \leftarrow \{0,1\}$, and for all $\alpha < i^*$, it computes $\left(\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right)$ using $\mathsf{Mixed\text{-}SubEnc}^*$ (as in $\mathsf{Game}\ 4.(i^*-1)$ and $\mathsf{Game}\ 4.i^*$). Note in particular that $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ are not used for computing these matrices. It then sends its target switching property query matrices $\mathbf{Z}_{0,b}^*, \mathbf{Z}_{1,b}^*$ of dimensions $\widetilde{n}_i \times m$ defined below.

$$\mathbf{Z}_{0,b}^* = \begin{bmatrix} \left\{\mathbf{C}_{i^*,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i^*)}} \\ \left\{\mathbf{P}_{i^*,\pi_{i^*,b}^{\gamma}(v)}\right\}_{v\in[w]} \end{bmatrix} \mathbf{Z}_{1,b}^* = \begin{bmatrix} \left\{\mathbf{C}_{i^*,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i^*)}} \\ \leftarrow \mathbb{Z}_q^{w\cdot n\times m} \end{bmatrix}$$

It receives $\mathbf{U}^*_{i^*,b}$ from the challenger.

Query Phase The ciphertext query is handled similar to the challenge ciphertext. The key queries are handled identically in both $\mathsf{Game}\ 4.(i^*-1)$ and $\mathsf{Game}\ 4.i^*$. ■

Lemma 8.11. For any adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A},5} = 0$.

Proof. We will argue that any adversary $\mathcal{A}$ has advantage 0 in $\mathsf{Game}\ 5$. First, note that the challenge phase uses $\mathsf{Mixed\text{-}SubEnc}^*$. As a result, it does not have any information about the choice $\gamma \leftarrow \{0,1\}$. Next, in the key query phase, for each key query, the challenger chooses $\mathbf{P}^{(\beta)}_{\ell+1,\mathsf{st}^{(\beta)}_{\ell+1}}$, which might depend on γ. However, the important point here is that for each key query x, both the challenge programs have identical output. Therefore, the $\mathbf{P}^{(\beta)}_{\ell+1,\mathsf{st}^{(\beta)}_{\ell+1}}$ matrices are independent of γ. As a result, the adversary has zero advantage in $\mathsf{Game}\ 5$. ■

8.4.3 1-Bounded Restricted Accept Indistinguishability

To prove restricted accept indistinguishability security, we take a slighty different approach. First, we show that our construction achieves *complete* accept indistinguishability security which is defined as follows.

Definition 8.1 (q-bounded Complete Accept Indistinguishability)**.** Let $q(\cdot)$ be any fixed polynomial. A mixed functional encryption scheme $\mathsf{Mixed\text{-}FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{SK\text{-}Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to satisfy q-bounded *complete* accept indistinguishability security if there exists algorithms $\mathsf{SK\text{-}Enc}^*, \mathsf{KeyGen}^*$ such that for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{O_1^b(\cdot),O_2^b(\cdot)}(\mathsf{pp},\mathsf{ct}_b) = b : \begin{array}{c} (1^\kappa, f^*) \leftarrow \mathcal{A}(1^\lambda); (\mathsf{pp},\mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa) \\ b \leftarrow \{0,1\};\ \mathsf{ct}_1 \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f^*) \\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}) \end{array}\right] \leq \frac{1}{2} + \text{negl}(\lambda)$$

where

- oracles $O_1^b(\cdot), O_2^b(\cdot)$ are defined as $O_1^0(\cdot) = \mathsf{KeyGen}^*(\mathsf{pp},\cdot)$, $O_1^1(\cdot) = \mathsf{KeyGen}(\mathsf{msk},\cdot)$, $O_2^0(\cdot) = \mathsf{SK\text{-}Enc}^*(\mathsf{pp},\cdot)$, $O_2^1(\cdot) = \mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)$.
- $\mathcal{A}$ can make at most $q(\lambda)$ queries to $O_2^b(\cdot)$ oracle,
- every secret key query m made by adversary $\mathcal{A}$ to the $O_1^b(\cdot)$ oracle must satisfy the condition that $f^*(m) = 1$ as well as $f(m) = 1$ for every ciphertext query f made by $\mathcal{A}$ to the $O_2^b(\cdot)$ oracle, and
- $\mathcal{A}$ must make all (at most $q(\lambda)$) $O_2^b(\cdot)$ oracle queries before making any query to $O_1^b(\cdot)$ oracle.

At a high level, it says that if the adversary only queries for keys for inputs m and ciphertexts for functions f such that $f(m) = 1$ on all combinations, then there exists special encryption and key generation algorithms $(\mathsf{SK\text{-}Enc}^*, \mathsf{KeyGen}^*)$ such that they only take public parameters as inputs and the adversary can not distinguish between correctly computed keys and ciphertexts from these (simulated) special keys and ciphertexts.

Below we provide a sequence of hybrid games that we later use to argue complete accept indistinguishability security. To complete the argument, later (in Section 8.4.5) we simply argue that complete accept indistinguishability implies restricted accept indistinguishability.

$\mathsf{Game}\ 0$: This corresponds to the original 1-bounded restricted accept indistinguishability security game in which the challenger encrypts the challenge branching program BP^* sent by the adversary.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—

1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$.
2. Next, it samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$
3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.
4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** The challenger chooses a random λ-bit string $\mathsf{tag}^* \leftarrow \{0,1\}^\lambda$. Let
$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = [\ell] \times [\lambda] \times \{0,1\}^2.$$
The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$
Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger chooses a λ-bit string $\mathsf{tag} \leftarrow \{0,1\}^\lambda$, and responds as follows.
     (a) Let $S = [\ell] \times [\lambda] \times \{0,1\}^2$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
$$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$
     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda - 1]$. Next, it sets vector $\mathbf{y}^{(\lambda)}$ as
$$\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda-1]} \mathbf{y}^{(j)}.$$
     (b) It then chooses secret vectors $\left\{\mathbf{s}_i^{(j,\beta)}\right\}_{i,j,\beta}$ and error vectors $\left\{\mathbf{e}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as
$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
$$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m.$$

(c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.
$\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Game 1 : This is identical to the previous game, except the challenger now chooses both tags tag^* and tag at the beginning during setup phase, and it aborts if $\mathsf{tag}^* = \mathsf{tag}$.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below.
  2. Next, it samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.
- **Challenge Phase.** Let
$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = [\ell] \times [\lambda] \times \{0,1\}^2.$$
The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
$$\forall\ \alpha \in [\ell], \quad \left(\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\begin{pmatrix} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{pmatrix}.$$
Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}_{i,b}^*\}_{i\in[\ell],b\in\{0,1\}}\right)$.
- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.

(a) Let $S = [\ell] \times [\lambda] \times \{0,1\}^2$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda - 1]$. Next, it sets vector $\mathbf{y}^{(\lambda)}$ as

$$\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda-1]} \mathbf{y}^{(j)}.$$

(b) It then chooses secret vectors $\left\{\mathbf{s}_i^{(j,\beta)}\right\}_{i,j,\beta}$ and error vectors $\left\{\mathbf{e}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as

$$\begin{aligned} &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m. \end{aligned}$$

(c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.
$\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell + 1 \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Notation. In all the following hybrid games, let diff denote the set of indices j such that $\mathsf{tag}_j^* \neq \mathsf{tag}_j$. Similarly, let comm denote the set of indices j such that $\mathsf{tag}_j^* = \mathsf{tag}_j$. Concretely, in all the following hybrids sets diff and comm are defined as follows:

$$\mathsf{diff} \stackrel{\text{def}}{=} \left\{j \in [\lambda] \ : \ \mathsf{tag}_j^* \neq \mathsf{tag}_j\right\}, \quad \mathsf{comm} \stackrel{\text{def}}{=} \mathsf{comm}.$$

Additionally, let j^* denote the smallest index in diff (i.e. $j^* = \min_j j \in \mathsf{diff}$), and $\beta^* = \mathsf{tag}_{j^*}^*$. Note that since the challenger aborts whenever $\mathsf{tag}^* = \mathsf{tag}$, thus j^*, β^* always exist whenever the challenger does not abort. Also, we will use $\widehat{\mathsf{diff}}$ to denote the set diff excluding index j^*, i.e. $\widehat{\mathsf{diff}} = \mathsf{diff} \setminus \{j^*\}$.

$\mathsf{Game}\ 2$: This is identical to the previous game, except the challenger while answering a secret key query now puts the $\widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}$ component in $\mathbf{y}^{(j^*)}$ instead of $\mathbf{y}^{(\lambda)}$, and rest are sampled uniformly at random.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—

1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below.
2. Next, it samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$
3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.
4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** Let
$$\mathsf{BP}^* = \left(\{\pi_{i,b}^* : [w] \to [w]\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = [\ell] \times [\lambda] \times \{0,1\}^2.$$

The challenger then runs the Mixed-SubEnc routine (described in Figure 6) as
$$\forall\ \alpha \in [\ell], \quad \left(\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}_{i,b}^*\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
   (a) Let $S = [\ell] \times [\lambda] \times \{0,1\}^2$. It runs the Mixed-SubEnc routine (described in Figure 6) as
   $$\forall\ \alpha \in [\ell], \quad \left(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$
   (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.
2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$. Next, it sets vector $\mathbf{y}^{(j^*)}$ as
   $$\mathbf{y}^{(j^*)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}.$$
   (b) It then chooses secret vectors $\left\{\mathbf{s}_i^{(j,\beta)}\right\}_{i,j,\beta}$ and error vectors $\left\{\mathbf{e}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as
   $$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
   $$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
   $$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m.$$

(c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.
$\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Next, we have a sequence of 4ℓ hybrid experiments $\mathsf{Game}\ 3.i^*.\{1,2,3,4\}$ for $i^* = 1$ to ℓ.

$\mathsf{Game}\ 3.i^*.1$: In hybrids $\mathsf{Game}\ 3.i^*.1$, the $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ matrices for all diff strands and levels $i < i^*$ are **not** sampled (at all) along with other level i matrices. And, ciphertext components for levels $i < i^*$ are used to only target remaining matrices, i.e. the ciphertext matrices do not target $\mathbf{B}_{i,b}^{(j,\beta)}$ matrices for $j \in \mathsf{diff}$ and $i < i^*$ to some pre-specified $\mathbf{C}_{i,b}^{(j,\beta)}$ or random matrices. Also, the first $i^* - 1$ components in each secret key are set to be uniformly random vectors, and the next component is hardwired such that correctness holds as well as some smudge-able noise is introduced in these components. Below we describe it in detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
  $$\forall\ i < i^*, \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2,$$
  $$\forall\ i \ge i^*, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$
  Also, let $\widetilde{n}_i = \begin{cases} \widetilde{n} - |\mathsf{diff}| \cdot 4n & \text{for } i < i^* \\ \widetilde{n} & \text{for } i \ge i^* \end{cases}$, and set $\hat{S} = \left\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j,\beta,b) \in S^{(i)}\right\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.
- **Challenge Phase.** Let
$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S^*}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \left\{\mathbf{U}_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.
     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\begin{aligned} &\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\ (j,\beta) \in [\lambda]\times\{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\ &\forall\ (i,j,\beta) \in [i^*-1]\times \mathsf{diff}\times\{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\ &\forall\ (j,\beta) \in \mathsf{diff}\times\{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m. \end{aligned}$$

     (c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after i^*-1 steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1]\times\mathsf{comm}\times\{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}$$

$$\forall\ (i,\beta,b) \in [\ell]\times\{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}^*_{i,b} & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^* \end{cases}$$

     Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)}\cdot\mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)}\cdot\mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)}\cdot\mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell \\ -\mathbf{s}_\ell^{(j,\beta)}\cdot\mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\forall\ (i,j,\beta) \in [i^*-1] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}$$

$$\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{i^*}^{(j,\beta)} = -\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{s}_{i^*}^{(j,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)} + \mathbf{e}_{i^*}^{(j,\beta)}$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} + \mathbf{e}_{i^*}^{(j^*,\beta)}$$

$\forall\ (i,j,\beta) \in ([\ell+1] \setminus [i^*]) \times \mathsf{diff} \times \{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Game $3.i^*.2$: This is identical to the previous game, except the $(i^*+1)^{th}$ key component in all diff strands is also hardwired. Below we describe it in detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

  $$\forall\ i < i^*, \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2,$$
  $$\forall\ i \geq i^*, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$

  Also, let $\widetilde{n}_i = \begin{cases} \widetilde{n} - |\mathsf{diff}| \cdot 4n & \text{for } i < i^* \\ \widetilde{n} & \text{for } i \geq i^* \end{cases}$, and set $\hat{S} = \left\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j,\beta,b) \in S^{(i)}\right\}$.

  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S^*}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \left\{\mathbf{U}_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.
     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
&\forall\ (i,j,\beta) \in [i^*-1] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ (j,\beta) \in \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
\end{aligned}$$

     (c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs $\mathsf{BP}, \mathsf{BP}^*$ after i^*-1 steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}^*_{i,b} & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^* \end{cases}$$

     Also, for $(j,\beta) \in \mathsf{diff} \times \{0,1\}$, let $\mathbf{B}^{(j,\beta)}_{\ell+1,\widetilde{x}_{\ell+1}} = \mathbf{0}^{n\times m}$, and $\widetilde{\mathbf{t}}_{i^*}^{(j,\beta)}$ denote the following vector.

$$\forall\ j \in \widehat{\mathsf{diff}}, \quad \widetilde{\mathbf{t}}_{i^*}^{(j,\beta)} = -\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta}\right) - \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \mathbf{s}_{i^*}^{(j,\beta)} \cdot \mathbf{B}^{(j,\beta)}_{i^*,\widetilde{x}_{i^*}} + \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)}.$$

$$\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot \mathbf{B}^{(j^*,\beta)}_{i^*,\widetilde{x}_{i^*}} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)}.$$

Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta)\in\Gamma,\quad \mathbf{t}_i^{(j,\beta)}=\begin{cases}\mathbf{s}_1^{(j,\beta)}\cdot\mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)}+\mathbf{y}^{(j)}+\mathbf{e}_1^{(j,\beta)} & \text{if } i=1\\ -\mathbf{s}_{i-1}^{(j,\beta)}\cdot\mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)}+\mathbf{s}_i^{(j,\beta)}\cdot\mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)}+\mathbf{e}_i^{(j,\beta)} & \text{if } 1<i\le\ell\\ -\mathbf{s}_\ell^{(j,\beta)}\cdot\mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)}+\mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i=\ell+1\end{cases}$$

$$\forall\ (i,j,\beta)\in[i^*-1]\times\mathsf{diff}\times\{0,1\},\quad \mathbf{t}_i^{(j,\beta)}=\widetilde{\mathbf{t}}_i^{(j,\beta)}+\mathbf{e}_i^{(j,\beta)}$$

$$\forall\ (j,\beta)\in\widehat{\mathsf{diff}}\times\{0,1\},\quad \mathbf{t}_{i^*}^{(j,\beta)}=\widetilde{\mathbf{t}}_{i^*}^{(j,\beta)}+\mathbf{e}_{i^*}^{(j,\beta)}$$

$$\forall\ \beta\in\{0,1\},\quad \mathbf{t}_{i^*}^{(j^*,\beta)}=\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)}+\mathbf{e}_{i^*}^{(j^*,\beta)}$$

$$\forall\ (j,\beta)\in\widehat{\mathsf{diff}}\times\{0,1\},\quad \mathbf{t}_{i^*+1}^{(j,\beta)}=-\sum_{\alpha=1}^{i^*}\left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)}\cdot\prod_{\delta=\alpha}^{i^*}\mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right)+\mathbf{y}^{(j)}\cdot\prod_{\delta=1}^{i^*}\mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}$$
$$+\ \mathbf{s}_{i^*+1}^{(j,\beta)}\cdot\mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j,\beta)}+\mathbf{e}_{i^*+1}^{(j,\beta)}$$

$$\forall\ \beta\in\{0,1\},\quad \mathbf{t}_{i^*+1}^{(j^*,\beta)}=-\sum_{\alpha=1}^{i^*}\left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)}\cdot\prod_{\delta=\alpha}^{i^*}\mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right)-\widetilde{\mathbf{y}}\cdot\prod_{\delta=1}^{i^*}\mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}$$
$$+\ \widetilde{\mathbf{s}}\cdot\mathbf{P}_{i^*+1,\mathsf{st}_{i^*+1}^{(\beta)}}+\mathbf{s}_{i^*+1}^{(j^*,\beta)}\cdot\mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)}+\mathbf{e}_{i^*+1}^{(j^*,\beta)}$$

$\forall\ (i,j,\beta)\in([\ell+1]\setminus[i^*+1])\times\mathsf{diff}\times\{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)}=\begin{cases}-\mathbf{s}_{i-1}^{(j,\beta)}\cdot\mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)}+\mathbf{s}_i^{(j,\beta)}\cdot\mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)}+\mathbf{e}_i^{(j,\beta)} & \text{if } i\le\ell\\ -\mathbf{s}_\ell^{(j,\beta)}\cdot\mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)}+\mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i=\ell+1\end{cases}$$

(d) Finally, it sends the secret key as $\left(x,\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Game $3.i^*.3$: This is identical to the previous game, except the matrices $\mathbf{B}_{i,b}^{(j,\beta)},\mathbf{C}_{i,b}^{(j,\beta)}$ for diff strands and levels $i=i^*$ are **not** sampled along with other level i^* matrices, but instead they are sampled uniformly at random. Also, ciphertext components for level i^* are used to only target remaining matrices. Below we describe it in detail.

- **Setup Phase.** The adversary sends the functionality index (k,w,L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n,m, and also distributions $\chi_{\mathsf{big}},\chi_s,\chi_{\mathsf{appr}},\chi_{\mathsf{pre}},\chi_{\mathsf{last}},\chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell=k\cdot L$ and $\widetilde{n}=(4\lambda+w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*,\mathsf{tag}\leftarrow\{0,1\}^\lambda$. If $\mathsf{tag}^*=\mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i<i^*+1,\quad S^{(i)}=\mathsf{comm}\times\{0,1\}^2,$$
$$\forall\ i\ge i^*+1,\quad S^{(i)}=[\lambda]\times\{0,1\}^2.$$

Also, let $\widetilde{n}_i=\begin{cases}\widetilde{n}-|\mathsf{diff}|\cdot 4n & \text{for } i<i^*+1\\ \widetilde{n} & \text{for } i\ge i^*+1\end{cases}$, and set $\hat{S}=\left\{(i,j,\beta,b)\in[\ell]\times[\lambda]\times\{0,1\}^2\ :\ (j,\beta,b)\in S^{(i)}\right\}$.

2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q),$$
$$\forall\ (j,\beta,b) \in \mathsf{diff} \times \{0,1\}^2, \quad \mathbf{B}_{i^*,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}.$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.
4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell], b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}_{i,b}^*\}_{i\in[\ell], b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
   (a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

   (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell], b\in\{0,1\}}\right)$.
2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.
   (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
$$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,$$
$$\forall\ (i,j,\beta) \in [i^*-1] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,$$
$$\forall\ (j,\beta) \in \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.$$

(c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after $i^* - 1$ steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)}$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^* \end{cases}$$

Also, for $(j,\beta) \in \mathsf{diff} \times \{0,1\}$, let $\mathbf{B}_{\ell+1,\widetilde{x}_{\ell+1}}^{(j,\beta)} = \mathbf{0}^{n \times m}$, and $\widetilde{\mathbf{t}}_{i^*}^{(j,\beta)}$ denote the following vector.

$$\forall\ j \in \widehat{\mathsf{diff}}, \quad \widetilde{\mathbf{t}}_{i^*}^{(j,\beta)} = -\sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{s}_{i^*}^{(j,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)}.$$

$$\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)}.$$

Next, it computes key vectors $\left\{ \mathbf{t}_i^{(j,\beta)} \right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\forall\ (i,j,\beta) \in [i^*-1] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}$$

$$\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{i^*}^{(j,\beta)} = \widetilde{\mathbf{t}}_{i^*}^{(j,\beta)} + \mathbf{e}_{i^*}^{(j,\beta)}$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} + \mathbf{e}_{i^*}^{(j^*,\beta)}$$

$$\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j,\beta)} = -\sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{s}_{i^*+1}^{(j,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j,\beta)} + \mathbf{e}_{i^*+1}^{(j,\beta)}$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}_{i^*+1}^{(\beta)}} + \mathbf{s}_{i^*+1}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)} + \mathbf{e}_{i^*+1}^{(j^*,\beta)}$$

$\forall\ (i,j,\beta) \in ([\ell+1] \setminus [i^*+1]) \times \mathsf{diff} \times \{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \left\{ \mathbf{t}_i^{(j,\beta)} \right\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}} \right)$.

- **Guess.** The adversary finally sends the guess γ'.

$\mathsf{Game}\ 3.i^*.4$: This is identical to the previous game, except the i^{*th} level key component in diff strands is a uniformly random n length vector, i.e. all first i^* level components in diff strands are random elements. Also, we no longer sample the matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ for diff strands and levels $i = i^*$ at all. Below we describe it in detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
$$\begin{aligned} \forall\ i < i^* + 1, \quad & S^{(i)} = \mathsf{comm} \times \{0,1\}^2, \\ \forall\ i \geq i^* + 1, \quad & S^{(i)} = [\lambda] \times \{0,1\}^2. \end{aligned}$$
     Also, let $\widetilde{n}_i = \begin{cases} \widetilde{n} - |\mathsf{diff}| \cdot 4n & \text{for } i < i^* + 1 \\ \widetilde{n} & \text{for } i \geq i^* + 1 \end{cases}$, and set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j, \beta, b) \in S^{(i)}\right\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b) \in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $(i, j, \beta, b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** Let
$$\begin{aligned} & \mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i \in [\ell], b \in \{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w] \right), \\ & S^* = \hat{S}. \end{aligned}$$
  The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b) \in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v) \in [\ell] \times [w]}, \{T_i\}_{i \in [\ell]}, \mathsf{BP}^* \end{array} \right).$$
  Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i \in [\ell], b \in \{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
$$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b) \in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v) \in [\ell] \times [w]}, \{T_i\}_{i \in [\ell]}, \mathsf{BP} \end{array} \right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.

   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

   (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,\\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,\\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,\\
&\forall\ (i,j,\beta) \in [i^*] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,\\
&\forall\ (j,\beta) \in \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
\end{aligned}$$

   (c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after $i^* - 1$ steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^* \end{cases}$$

   Also, for $(j,\beta) \in \mathsf{diff} \times \{0,1\}$, let $\mathbf{B}_{\ell+1,\widetilde{x}_{\ell+1}}^{(j,\beta)} = \mathbf{0}^{n\times m}$. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\forall\ (i,j,\beta) \in [i^*] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}$$

$$\begin{aligned}
\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j,\beta)} = {} & -\sum_{\alpha=1}^{i^*}\left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\
& + \mathbf{s}_{i^*+1}^{(j,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j,\beta)} + \mathbf{e}_{i^*+1}^{(j,\beta)}
\end{aligned}$$

$$\begin{aligned}
\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j^*,\beta)} = {} & -\sum_{\alpha=1}^{i^*}\left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\
& + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}_{i^*+1}^{(\beta)}} + \mathbf{s}_{i^*+1}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)} + \mathbf{e}_{i^*+1}^{(j^*,\beta)}
\end{aligned}$$

   $\forall\ (i,j,\beta) \in ([\ell+1] \setminus [i^*+1]) \times \mathsf{diff} \times \{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } i \le \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

   (d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

$\mathsf{Game}\ 4:$ This is similar to $\mathsf{Game}\ 3.\ell.4$, except that except that the terms $\mathbf{t}_{\ell+1}^{(j^*,\beta)}$ have an additional small error $\widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta)}$, which is smudged by the main error term $\mathbf{e}_{\ell+1}^{(j^*,\beta)}$.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
  $$\forall\ i \in [\ell], \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2.$$
  Also, let $\widetilde{n}_i = \widetilde{n} - |\mathsf{diff}| \cdot 4n$ for all $i \in [\ell]$, and set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ :\ (j, \beta, b) \in S^{(i)}\right\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i, j, \beta, b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.
- **Challenge Phase.** Let
  $$\begin{aligned}\mathsf{BP}^* &= \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell], b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right), \\ S^* &= \hat{S}.\end{aligned}$$
  The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
  $$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$
  Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell], b\in\{0,1\}}\right)$.
- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
     $$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$
     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell], b\in\{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

(b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,\\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,\\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,\\
&\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,\\
&\forall \beta \in \{0,1\}, \quad \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
\end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after ℓ steps (respectively). Also, let $\Gamma, \widetilde{\mathbf{y}}$ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)}$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^* \end{cases}$$

For $v \in [w]$, let $\mathbf{P}_{\ell+1,v}^{(\beta^*)}$ be the top level matrices chosen while computing challenge ciphertext. Similarly, $\mathbf{P}_{\ell+1,v}^{(1-\beta^*)}$ be the top level matrices chosen while computing query ciphertext. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\begin{aligned}
\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} &= \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}\\
\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j,\beta)} &= -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\\
&\quad + \mathbf{e}_{\ell+1}^{(j,\beta)}\\
\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} &= -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\\
&\quad + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta)}}^{(\beta^*)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta)}
\end{aligned}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Game 5 : This is identical to the previous game, except the $(\ell+1)^{th}$ key components in the special strand (i.e., j^{*th} strand) are random elements. we describe in it detail below.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—

1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
$$\forall\, i \in [\ell], \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2.$$
Also, let $\widetilde{n}_i = \widetilde{n} - |\mathsf{diff}| \cdot 4n$ for all $i \in [\ell]$, and set $\hat{S} = \left\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j,\beta,b) \in S^{(i)}\right\}$.
2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
$$\forall\, i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b) \in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$
3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $(i,j,\beta,b) \in \hat{S}$.
4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** Let
$$\begin{aligned} \mathsf{BP}^* &= \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i \in [\ell], b \in \{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right), \\ S^* &= \hat{S}. \end{aligned}$$
The challenger then runs the Mixed-SubEnc routine (described in Figure 6) as
$$\forall\, \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{matrix} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b) \in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v) \in [\ell] \times [w]}, \{T_i\}_{i \in [\ell]}, \mathsf{BP}^* \end{matrix}\right).$$
Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i \in [\ell], b \in \{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Figure 6) as
     $$\forall\, \alpha \in [\ell], \quad \left(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{matrix} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b) \in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v) \in [\ell] \times [w]}, \{T_i\}_{i \in [\ell]}, \mathsf{BP} \end{matrix}\right).$$
     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.
     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows
     $$\begin{aligned} &\forall\, (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\, (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\, (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\ &\forall\, (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\ &\forall\, \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m. \end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after ℓ steps (respectively). Also, let Γ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times \mathsf{comm} \times \{0,1\}$$
$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^* \end{cases}$$

For $v \in [w]$, let $\mathbf{P}_{\ell+1,v}^{(\beta^*)}$ be the top level matrices chosen while computing challenge ciphertext. Similarly, $\mathbf{P}_{\ell+1,v}^{(1-\beta^*)}$ be the top level matrices chosen while computing query ciphertext. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}$$
$$\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)}$$
$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Next, we have a sequence of ℓ hybrid experiments $\mathsf{Game}\ 5.i^*$ for $i^* = 1$ to ℓ.

$\mathsf{Game}\ 5.i^*$: This is identical to the previous game, except the matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ for $j \in \mathsf{comm}, \beta = 1 - \mathsf{tag}_j$ strands (i.e., strands in which $\mathbf{B}_{i,b}^{(j,\beta)}$ were targetting random matrices themselves) and levels $i \leq i^*$ are **not** sampled along with other level $i \leq i^*$ matrices, but instead they are sampled uniformly at random. Also, ciphertext components for levels $i \leq i^*$ are used to only target remaining matrices. Below we describe it in detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i \leq i^*, \quad S^{(i)} = \left\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2 \ :\ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\},$$
$$\forall\ i > i^*, \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2.$$

Also, let $\widetilde{n}_i = \begin{cases} (2 \cdot |\mathsf{comm}| + w)n & \text{for } i \leq i^* \\ \widetilde{n} - |\mathsf{diff}| \cdot 4n & \text{for } i > i^* \end{cases}$, and set $\hat{S} = \{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 : (j,\beta,b) \in S^{(i)}\}$.

2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q),$$
$$\forall\ (i,j,\beta,b) \in ([i^*] \times \mathsf{comm} \times \{0,1\}^2) \setminus \hat{S}, \quad \mathbf{B}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}.$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.
4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}_{i,b}^*\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad (\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.
     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
$$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,$$
$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,$$
$$\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.$$

(c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after ℓ steps (respectively). Also, let Γ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times \mathsf{comm} \times \{0,1\}$$
$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^* \end{cases}$$

For $v \in [w]$, let $\mathbf{P}_{\ell+1,v}^{(\beta^*)}$ be the top level matrices chosen while computing challenge ciphertext. Similarly, $\mathbf{P}_{\ell+1,v}^{(1-\beta^*)}$ be the top level matrices chosen while computing query ciphertext. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}$$
$$\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)}$$
$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Next, we have a sequence of ℓ hybrid experiments $\mathsf{Game}\ 6.i^*$ for $i^* = 2$ to $\ell+1$.

$\mathsf{Game}\ 6.i^*$: This is identical to the previous game (i.e. $\mathsf{Game}\ 5.\ell$), except in the comm strands, for which we sample $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ matrices uniformly at random, the key components for levels $i \leq i^*$ are random elements. Below we describe in it detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
  $$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2 \ : \ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$
  Also, let $\widetilde{n}_i = (2 \cdot |\mathsf{comm}| + w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 :$ $(j,\beta,b) \in S^{(i)}\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q),$$
  $$\forall\ (i,j,\beta,b) \in ([\ell] \times \mathsf{comm} \times \{0,1\}^2) \setminus \hat{S}, \quad \mathbf{B}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}.$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.
4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell], b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP}^*\end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell], b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
   (a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP}\end{array}\right).$$

   (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \left\{\mathbf{U}_{i,b}\right\}_{i\in[\ell], b\in\{0,1\}}\right)$.
2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.
   (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
&\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ (i,j) \in [i^*] \times \mathsf{comm}, \quad \widetilde{\mathbf{t}}_i^{(j,1-\mathsf{tag}_j)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.
\end{aligned}$$

   (c) Let $\widetilde{x} = x^L$, and $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, BP* after ℓ steps (respectively). Also, let Γ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = ([\ell+1] \setminus [i^*]) \times \mathsf{comm} \times \{0,1\} \cup \left\{(i,j,\beta) \in [i^*] \times \mathsf{comm} \times \{0,1\} \ : \ \beta = \mathsf{tag}_j\right\}$$
$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases}\mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^*\end{cases}$$

For $v \in [w]$, let $\mathbf{P}^{(\beta^*)}_{\ell+1,v}$ be the top level matrices chosen while computing challenge ciphertext. Similarly, $\mathbf{P}^{(1-\beta^*)}_{\ell+1,v}$ be the top level matrices chosen while computing query ciphertext. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}^{(j,\beta)}_{1,\widetilde{x}_1} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1 \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}^{(j,\beta)}_{i-1,\widetilde{x}_{i-1}} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}^{(j,\beta)}_{i,\widetilde{x}_i} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}^{(j,\beta)}_{\ell,\widetilde{x}_\ell} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\forall\ (i,j) \in [i^*] \times \mathsf{comm}, \quad \mathbf{t}_i^{(j,1-\mathsf{tag}_j)} = \widetilde{\mathbf{t}}_i^{(j,1-\mathsf{tag}_j)} + \mathbf{e}_i^{(j,1-\mathsf{tag}_j)}$$

$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}$$

$$\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \mathbf{e}_{\ell+1}^{(j,\beta)}$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Next, we have a sequence of $\ell+1$ hybrid experiments $\mathsf{Game}\ 7.i^*$ for $i^* = 1$ to $\ell+1$.

$\mathsf{Game}\ 7.i^*$: This is identical to the previous game (i.e. $\mathsf{Game}\ 6.(\ell+1)$), except in the comm strands, for which we **still** sample $\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}$ matrices using $\mathsf{EnTrapGen}$, the key components for levels $i \leq i^*$ are random elements. Also, we no longer sample the matrices $\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}$ at all which were sampled uniformly at random in the previous game. Below we describe in it detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

  $$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2 \ : \ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$

  Also, let $\widetilde{n}_i = (2 \cdot |\mathsf{comm}| + w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 : (j,\beta,b) \in S^{(i)}\}$.

  2. It samples $\left\{\mathbf{B}^{(j,\beta)}_{i,b}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}^{(j,\beta)}_{i,b}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

  3. It then samples matrices $\mathbf{C}^{(j,\beta)}_{i,b} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** Let

$$\mathsf{BP}^* = \left(\left\{ \pi^*_{i,b} : [w] \to [w] \right\}_{i \in [\ell], b \in \{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w] \right),$$
$$S^* = \hat{S}.$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

$$\forall\, \alpha \in [\ell], \quad \left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b) \in S^*}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v) \in [\ell] \times [w]}, \left\{T_i\right\}_{i \in [\ell]}, \mathsf{BP}^* \end{array} \right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i \in [\ell], b \in \{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.

     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

     $$\forall\, \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b) \in S}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v) \in [\ell] \times [w]}, \left\{T_i\right\}_{i \in [\ell]}, \mathsf{BP} \end{array} \right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \left\{\mathbf{U}_{i,b}\right\}_{i \in [\ell], b \in \{0,1\}}\right)$.

  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.

     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

     (b) It then chooses vectors $\mathbf{s}^{(j,\beta)}_i, \mathbf{e}^{(j,\beta)}_i, \widetilde{\mathbf{t}}^{(j,\beta)}_i, \widetilde{\mathbf{e}}^{(j,\beta)}_i$ as follows

     $$\begin{aligned} &\forall\, (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}^{(j,\beta)}_i \leftarrow \mathbb{Z}_q^n, \\ &\forall\, (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}^{(j,\beta)}_i \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\, (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}^{(j,\beta)}_{\ell+1} \leftarrow \chi_{\mathsf{last}}^m, \\ &\forall\, (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}^{(j,\beta)}_i \leftarrow \mathbb{Z}_q^m, \\ &\forall\, (i,j) \in [\ell+1] \times \mathsf{comm}, \quad \widetilde{\mathbf{t}}^{(j,1-\mathsf{tag}_j)}_i \leftarrow \mathbb{Z}_q^m, \\ &\forall\, (i,j) \in [i^*] \times \mathsf{comm}, \quad \widetilde{\mathbf{t}}^{(j,\mathsf{tag}_j)}_i \leftarrow \mathbb{Z}_q^m, \\ &\forall\, \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}^{(j^*,\beta)}_{\ell+1} \leftarrow \mathbb{Z}_q^m. \end{aligned}$$

     (c) Let $\widetilde{x} = x^L$, and $\mathsf{st}^{(1-\beta^*)}_{\ell+1}, \mathsf{st}^{(\beta^*)}_{\ell+1}$ denote the state of branching programs $\mathsf{BP}, \mathsf{BP}^*$ after ℓ steps (respectively). Also, let Γ and $\mathbf{U}^{(\beta)}_{i,b}$ denote the following:

     $$\Gamma = \left\{ (i,j,\beta) \in ([\ell+1] \setminus [i^*]) \times \mathsf{comm} \times \{0,1\} \;:\; \beta = \mathsf{tag}_j \right\}$$
     $$\forall\, (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}^{(\beta)}_{i,b} = \begin{cases} \mathbf{U}^*_{i,b} & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^* \end{cases}$$

For $v \in [w]$, let $\mathbf{P}_{\ell+1,v}^{(\beta^*)}$ be the top level matrices chosen while computing challenge ciphertext. Similarly, $\mathbf{P}_{\ell+1,v}^{(1-\beta^*)}$ be the top level matrices chosen while computing query ciphertext. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\forall\ (i,j) \in [i^*] \times \mathsf{comm}, \quad \mathbf{t}_i^{(j,\mathsf{tag}_j)} = \widetilde{\mathbf{t}}_i^{(j,\mathsf{tag}_j)} + \mathbf{e}_i^{(j,\mathsf{tag}_j)}$$

$$\forall\ (i,j,\beta) \in \Gamma, \quad \mathbf{t}_i^{(j,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1 \end{cases}$$

$$\forall\ (i,j) \in [\ell+1] \times \mathsf{comm}, \quad \mathbf{t}_i^{(j,1-\mathsf{tag}_j)} = \widetilde{\mathbf{t}}_i^{(j,1-\mathsf{tag}_j)} + \mathbf{e}_i^{(j,1-\mathsf{tag}_j)}$$

$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}$$

$$\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)}$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Game 8 : This is identical to the previous game (i.e. Game 7.$(\ell+1)$). For the ease of exposition, we describe in it detail below.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

     $$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2 \ : \ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$

     Also, let $\widetilde{n}_i = (2 \cdot |\mathsf{comm}| + w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i,j,\beta,b) \in [\ell]\times [\lambda] \times \{0,1\}^2 : (j,\beta,b) \in S^{(i)}\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

     $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.
- **Challenge Phase.** Let

  $$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
  $$S^* = \hat{S}.$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S^*}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as

$$\forall\ \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \left\{\mathbf{U}_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.
  2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
     (a) It chooses $|\mathsf{diff}| - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in \widehat{\mathsf{diff}}$.
     (b) It then chooses vectors $\mathbf{e}^{(j,\beta)}_i, \widetilde{\mathbf{t}}^{(j,\beta)}_i, \widetilde{\mathbf{e}}^{(j,\beta)}_i$ as follows

$$\begin{aligned} \forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, &\quad \mathbf{e}^{(j,\beta)}_i \leftarrow \chi^m_{\mathsf{big}}, \\ \forall\ (j,\beta) \in [\lambda]\times\{0,1\}, &\quad \mathbf{e}^{(j,\beta)}_{\ell+1} \leftarrow \chi^m_{\mathsf{last}}, \\ \forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, &\quad \widetilde{\mathbf{t}}^{(j,\beta)}_i \leftarrow \mathbb{Z}_q^m, \\ \forall\ (j,\beta) \in \mathsf{comm}\times\{0,1\}, &\quad \widetilde{\mathbf{t}}^{(j,\beta)}_{\ell+1} \leftarrow \mathbb{Z}_q^m, \\ \forall\ \beta \in \{0,1\}, &\quad \widetilde{\mathbf{t}}^{(j^*,\beta)}_{\ell+1} \leftarrow \mathbb{Z}_q^m. \end{aligned}$$

     (c) Let $\widetilde{x} = x^L$, and let $\mathbf{U}^{(\beta)}_{i,b}$ denote the following:

$$\forall\ (i,\beta,b) \in [\ell]\times\{0,1\}^2, \quad \mathbf{U}^{(\beta)}_{i,b} = \begin{cases} \mathbf{U}^*_{i,b} & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^* \end{cases}$$

     Next, it computes key vectors $\left\{\mathbf{t}^{(j,\beta)}_i\right\}_{i,j,\beta}$ as follows.

$$\begin{aligned} \forall\ (i,j,\beta) \in [\ell+1]\times\mathsf{comm}\times\{0,1\}, &\quad \mathbf{t}^{(j,\beta)}_i = \widetilde{\mathbf{t}}^{(j,\beta)}_i + \mathbf{e}^{(j,\beta)}_i \\ \forall\ (i,j,\beta) \in [\ell]\times\mathsf{diff}\times\{0,1\}, &\quad \mathbf{t}^{(j,\beta)}_i = \widetilde{\mathbf{t}}^{(j,\beta)}_i + \mathbf{e}^{(j,\beta)}_i \\ \forall\ (j,\beta) \in \widehat{\mathsf{diff}}\times\{0,1\}, &\quad \mathbf{t}^{(j,\beta)}_{\ell+1} = -\sum_{\alpha=1}^{\ell}\left(\widetilde{\mathbf{t}}^{(j,\beta)}_\alpha \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta}\right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \mathbf{e}^{(j,\beta)}_{\ell+1} \\ \forall\ \beta \in \{0,1\}, &\quad \mathbf{t}^{(j^*,\beta)}_{\ell+1} = \widetilde{\mathbf{t}}^{(j^*,\beta)}_{\ell+1} + \mathbf{e}^{(j^*,\beta)}_{\ell+1} \end{aligned}$$

     (d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}^{(j,\beta)}_i\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.
- **Guess.** The adversary finally sends the guess γ'.

Next, we have a sequence of ℓ hybrid experiments $\mathsf{Game}\ 8.i^*$ for $i^* = 1$ to ℓ.

$\mathsf{Game}\ 8.i^*$: This is identical to the previous game, except now the challenger samples the first i^* ciphertext components (both challenge and queried) as random gaussian matrices. Below we describe in it detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
  $$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j, \beta, b) \in [\lambda] \times \{0,1\}^2\ :\ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$
  Also, let $\widetilde{n}_i = (2 \cdot |\mathsf{comm}| + w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i, j, \beta, b) \in [\ell]\times [\lambda] \times \{0,1\}^2 : (j, \beta, b) \in S^{(i)}\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix}\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]}\end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i, j, \beta, b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.
- **Challenge Phase.** Let
  $$\begin{aligned}\mathsf{BP}^* &= \left(\{\pi^*_{i,b} : [w] \to [w]\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),\\ S^* &= \hat{S}.\end{aligned}$$
  The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
  $$\forall\ (\alpha, b) \in [i^*] \times \{0,1\}, \quad \mathbf{U}^*_{\alpha,b} \leftarrow \chi_{\mathsf{pre}}^{m\times m},$$
  $$\forall\ \alpha \in [\ell]\setminus[i^*], \quad \left(\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\begin{pmatrix}\mathsf{tag}^*, \alpha, S^*, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^*\end{pmatrix}.$$
  Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}^*_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.
- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.
  1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Figure 6) as
     $$\forall\ (\alpha, b) \in [i^*] \times \{0,1\}, \quad \mathbf{U}_{\alpha,b} \leftarrow \chi_{\mathsf{pre}}^{m\times m},$$
     $$\forall\ \alpha \in [\ell]\setminus[i^*], \quad \left(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\begin{pmatrix}\mathsf{tag}, \alpha, S, \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}\end{pmatrix}.$$
     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
   (a) It chooses $|\mathsf{diff}| - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in \widehat{\mathsf{diff}}$.
   (b) It then chooses vectors $\mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ (j,\beta) \in \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.
\end{aligned}$$

   (c) Let $\widetilde{x} = x^L$, and let $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^* \end{cases}$$

   Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\begin{aligned}
\forall\ (i,j,\beta) \in [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad & \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} \\
\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad & \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} \\
\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad & \mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} \\
\forall\ \beta \in \{0,1\}, \quad & \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}
\end{aligned}$$

   (d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}} \right)$.

- **Guess.** The adversary finally sends the guess γ'.

$\mathsf{Game}\ 9$: This is identical to the previous game (i.e., $\mathsf{Game}\ 8.\ell$), except the last secret key components in all $\widehat{\mathsf{diff}}$ strands also include an additional noise term which is much smaller than the overall noise added in those components. Below we describe in it detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—

1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2 \ : \ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$

   Also, let $\widetilde{n}_i = (2 \cdot |\mathsf{comm}| + w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 : (j,\beta,b) \in S^{(i)}\}$.

2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.
4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge Phase.** The challenger generates ciphertext components as

$$\forall\ (i,b) \in [\ell] \times \{0,1\}, \quad \mathbf{U}_{i,b}^* \leftarrow \chi_{\mathsf{pre}}^{m\times m}.$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}_{i,b}^*\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
   (a) It generates ciphertext components as

$$\forall\ (i,b) \in [\ell] \times \{0,1\}, \quad \mathbf{U}_{i,b} \leftarrow \chi_{\mathsf{pre}}^{m\times m}.$$

   (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
   (a) It chooses $|\mathsf{diff}| - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in \widehat{\mathsf{diff}}$.
   (b) It then chooses vectors $\mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ (j,\beta) \in \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \widetilde{\mathbf{e}}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
\end{aligned}$$

   (c) Let $\widetilde{x} = x^L$, and let $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^* \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^* \end{cases}$$

Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.

$$\begin{aligned}
\forall\ (i,j,\beta) \in [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad & \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} \\
\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad & \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} \\
\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad & \mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell}\left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\
& \quad + \widetilde{\mathbf{e}}_{\ell+1}^{(j,\beta)} \cdot \prod_{\delta=2}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} \\
\forall\ \beta \in \{0,1\}, \quad & \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}
\end{aligned}$$

(d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Game 10 : This is identical to the previous game, except the last secret key components in all $\widehat{\mathsf{diff}}$ strands are random vectors as well. Below we describe in it detail.

- **Setup Phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows—
  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:
  $$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2 \ : \ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$
  Also, let $\widetilde{n}_i = (2 \cdot |\mathsf{comm}| + w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 : (j,\beta,b) \in S^{(i)}\}$.
  2. It samples $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.
- **Challenge Phase.** The challenger generates ciphertext components as
  $$\forall\ (i,b) \in [\ell] \times \{0,1\}, \quad \mathbf{U}_{i,b}^* \leftarrow \chi_{\mathsf{pre}}^{m\times m}.$$
  Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}_{i,b}^*\}_{i\in[\ell],b\in\{0,1\}}\right)$.
- **Post-Challenge Phase.** The adversary is allowed make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext Query.** The adversary sends a branching program BP for encryption. The challenger responds as follows.
   (a) It generates ciphertext components as
   $$\forall\ (i,b) \in [\ell] \times \{0,1\}, \quad \mathbf{U}_{i,b} \leftarrow \chi_{\mathsf{pre}}^{m \times m}.$$
   (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}\right)$.
2. **Secret Key Queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.
   (a) It chooses $|\mathsf{diff}| - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in \widehat{\mathsf{diff}}$.
   (b) It then chooses vectors $\mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows
   $$\begin{aligned} &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\ &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\ &\forall\ (j,\beta) \in \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\ &\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m, \\ &\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j,\beta)} \leftarrow \mathbb{Z}_q^m. \end{aligned}$$
   (c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i,j,\beta}$ as follows.
   $$\begin{aligned} \forall\ (i,j,\beta) \in [\ell+1] \times \mathsf{comm} \times \{0,1\}, &\quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} \\ \forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, &\quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} \\ \forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, &\quad \mathbf{t}_{\ell+1}^{(j,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} \\ \forall\ \beta \in \{0,1\}, &\quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)} \end{aligned}$$
   (d) Finally, it sends the secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

8.4.4 Indistinguishability of Hybrid Games in Section 8.4.3

Now we show that the hybrid experiments described in Section 8.4.3 are computationally indistinguishable. For any PPT adversary $\mathcal{A}$, let $p_{\mathcal{A},x}(\cdot)$ denote the probability that adversary $\mathcal{A}$ outputs $\gamma' = 1$ in $\mathsf{Game}\ x$.

Lemma 8.12. There exists a negligible function $\mathrm{negl}(\cdot)$ such that for any adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0}(\lambda) - p_{\mathcal{A},1}(\lambda) \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this lemma is identical to that of Lemma 8.3. ∎

Lemma 8.13. For any adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, $p_{\mathcal{A},1}(\lambda) = p_{\mathcal{A},2}(\lambda)$.

Proof. The proof of this lemma is identical to that of Lemma 8.4. ∎

Lemma 8.14. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.1.1}(\lambda) \leq \mathrm{negl}(\lambda)$.

Proof. First, let us list the differences between $\mathsf{Game}\ 2$ and $\mathsf{Game}\ 3.1.1$. The setup, challenge phase and ciphertext query are handled in an identical manner. The key queries, however, are handled differently. For each key query x, the challenger outputs $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell]\times[\lambda]\times\{0,1\}}$ as the secret key. The components $\{\mathbf{t}_1^{(j,\beta)}\}_{j\in\mathsf{diff},\beta\in\{0,1\}}$ are computed differently in $\mathsf{Game}\ 2$ and $\mathsf{Game}\ 3.1.1$. In particular, in $\mathsf{Game}\ 3.1.1$, the challenger adds an additional error term $\widetilde{\mathbf{e}}_1^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m$ in $\mathbf{t}_1^{(j,\beta)}$.

The proof of this lemma is similar to the proof of Lemma 8.5, (and uses the smudging lemma — Lemma 2.1). Therefore, $\mathsf{Adv}_{\mathcal{A},2}(\lambda)-\mathsf{Adv}_{\mathcal{A},3.1.1}(\lambda) \leq q_{\mathsf{keys}}(\lambda)\cdot(2\cdot|\mathsf{diff}|\cdot \mathrm{negl}_{\mathsf{smud}}(\lambda)) \leq q_{\mathsf{keys}}(\lambda)\cdot(2\lambda\cdot\mathrm{negl}_{\mathsf{smud}}(\lambda))$, and the lemma follows by setting $\mathrm{negl}_{3.1.1} = (2\lambda\cdot\mathrm{negl}_{\mathsf{smud}})$. ∎

Lemma 8.15. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda\in\mathbb{N}$ and $i^*\in[\ell]$, $\mathsf{Adv}_{\mathcal{A},3.i^*.1}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) \leq \mathrm{negl}(\lambda)$.

Proof. Let us first consider the differences between $\mathsf{Game}\ 3.i^*.1$ and $\mathsf{Game}\ 3.i^*.2$. The setup, challenge phase and ciphertext query are handled in an identical manner in both games. The key generation queries are computed differently (in particular the components $\left\{\mathbf{t}_{i^*+1}^{(j,\beta)}\right\}_{j\in\mathsf{diff},\beta\in\{0,1\}}$ in each secret key).

The proof of this lemma is similar to the proof of Lemma 8.6, and the main idea is to use 8.1 to argue that $\mathbf{e}_{i^*+1}^{(j,\beta)}$ drowns $\widetilde{\mathbf{e}}_{i^*+1}^{(j,\beta)}\cdot\mathbf{U}_{i^*,\widetilde{x}_{i^*}}^{(\beta)}$ and $\widetilde{\mathbf{e}}_{i^*+1}^{(j,\beta)}$. ∎

Lemma 8.16. Assuming the trapdoor generation algorithms $\mathsf{LT}_{\mathsf{en}}$ satisfy $(q,\sigma_{\mathsf{pre}})$-row removal property, for any PPT adversary $\mathcal{A}$ and $i^*\in[\ell]$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda\in\mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda) \leq \mathrm{negl}(\lambda)$.

Proof. This proof is similar to the proof of Lemma 8.7, and we will be using the row removal property to prove it. We will first present the differences between the two games and then discuss why row removal property is applicable here. The exact reduction from the row removal property to indistinguishability of $\mathsf{Game}\ 3.i^*.2$ and $\mathsf{Game}\ 3.i^*.3$ can be found in the proof of Lemma 8.7.

Differences between $\mathsf{Game}\ 3.i^*.2$ and $\mathsf{Game}\ 3.i^*.3$:

1. Set $S^{(i^*)}$: In $\mathsf{Game}\ 3.i^*.2$, the challenger sets $S^{(i^*)} = [\lambda]\times\{0,1\}^2$, while in $\mathsf{Game}\ 3.i^*.3$, $S^{(i^*)} = \mathsf{comm}\times\{0,1\}^2$ ($\mathsf{tag}^*, \mathsf{tag}$ are chosen at the start of the security game, so the set diff is well defined here). Also, $\widetilde{n}_{i^*} = \widetilde{n} = (4\lambda+w)n$ in $\mathsf{Game}\ 3.i^*.2$, while $\widetilde{n}_{i^*} = \widetilde{n} - |\mathsf{diff}|\cdot 4n$ in $\mathsf{Game}\ 3.i^*.3$.

2. $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i=i^*}$ matrices : In $\mathsf{Game}\ 3.i^*.2$, the challenger chooses $(\mathbf{M}_{i^*}, T_{i^*}) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q)$, while in $\mathsf{Game}\ 3.i^*.3$, it chooses $(\mathbf{M}_{i^*}, T_{i^*}) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}-|\mathsf{diff}|\cdot 4n}, 1^m, q)$. As a result, in $\mathsf{Game}\ 3.i^*.2$, it derives all $\left\{\mathbf{B}_{i^*,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2}$ from $\mathbf{M}_{i^*}$. In $\mathsf{Game}\ 3.i^*.3$, the challenger chooses $\left\{\mathbf{B}_{i^*,b}^{(j,\beta)}\right\}_{j\in\mathsf{diff},b,\beta\in\{0,1\}}$ uniformly at random, while the remaining are derived from $\mathbf{M}_{i^*}$.

3. Ciphertexts: Since the set $S^{(i^*)}$ is different in both games, the challenge and query ciphertexts are constructed differently in both games. In particular, the challenge ciphertext components $(\mathbf{U}_{i^*,0}^*, \mathbf{U}_{i^*,1}^*)$ and the ciphertext query components $(\mathbf{U}_{i^*,0}, \mathbf{U}_{i^*,1})$ are computed using $\mathbf{M}_{i^*}$ and T_{i^*}, which are computed differently in $\mathsf{Game}\ 3.i^*.2$ and $\mathsf{Game}\ 3.i^*.3$.

Let us now discuss why row removal property suffices for proving this lemma.

- Consider the four matrices $\left(\mathbf{U}_{i^*,0}^*, \mathbf{U}_{i^*,1}^*, \mathbf{U}_{i^*,0}, \mathbf{U}_{i^*,1}\right)$. Fix any $j\in\mathsf{diff}$, and let $\mathsf{tag}_j^* = \beta$ and $\mathsf{tag}_j = 1-\beta$. Then, from the definition of $\mathbf{D}_b^{(j,\beta)}$ and $\widetilde{\mathbf{D}}_b^{(j,\beta)}$ in $\mathsf{Mixed\text{-}SubEnc}$, it follows that
  - $\mathbf{B}_{i^*,0}^{(j,\beta)}\cdot\mathbf{U}_{i^*,0}^* = \mathbf{C}_{i^*,0}^{(j,\beta)}$, the rest are mapped to random matrices.
  - $\mathbf{B}_{i^*,1}^{(j,\beta)}\cdot\mathbf{U}_{i^*,1}^* = \mathbf{C}_{i^*,1}^{(j,\beta)}$, the rest are mapped to random matrices.

  - $\mathbf{B}_{i^*,0}^{(j,1-\beta)} \cdot \mathbf{U}_{i^*,0} = \mathbf{C}_{i^*,0}^{(j,1-\beta)}$, the rest are mapped to random matrices.
  - $\mathbf{B}_{i^*,1}^{(j,1-\beta)} \cdot \mathbf{U}_{i^*,1} = \mathbf{C}_{i^*,1}^{(j,1-\beta)}$, the rest are mapped to random matrices.

- Next, note that the $\left\{\mathbf{C}_{i^*,b}^{(j,\beta)}\right\}_{j \in \mathsf{diff}, b, \beta \in \{0,1\}}$ are not used for responding to key generation queries.
- Using the above points, we can conclude that in Game 3.i^*.2, for each $j \in \mathsf{diff}$, each of these four matrices $\left(\mathbf{U}^*_{i^*,0}, \mathbf{U}^*_{i^*,1}, \mathbf{U}_{i^*,0}, \mathbf{U}_{i^*,1}\right)$ maps $\left[\mathbf{B}_{i^*,0}^{(j,0)} \mid \mathbf{B}_{i^*,1}^{(j,0)} \mid \mathbf{B}_{i^*,0}^{(j,1)} \mid \mathbf{B}_{i^*,1}^{(j,1)}\right]$ to a uniformly random matrix.

The proof of this lemma therefore follows using the row removal property. ■

Lemma 8.17. Assuming the $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ assumption, for any PPT adversary $\mathcal{A}$ and $i^* \in [\ell]$, there exists a negligible function $\mathrm{negl}_{3.i^*.4}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.4}(\lambda) \le \mathrm{negl}_{3.i^*.4}(\lambda)$.

Proof. The proof of this lemma is similar to the proof of Lemma 8.8, except that it involves more hybrid experiments.

In Game 3.i^*.3, for each key query x, for each $\beta \in \{0,1\}$ and $j \in \mathsf{diff}$, the component $\widetilde{\mathbf{t}}_{i^*}^{(j,\beta)} = -\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}_{\alpha}^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)}$. In Game 3.$i^*$.4, $\widetilde{\mathbf{t}}_{i^*}^{(j,\beta)} \leftarrow \mathbb{Z}_q^m$. Let $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ denote the number of keys queried by $\mathcal{A}(1^\lambda)$. To prove that these two games are computationally indistinguishable, we will define $q_{\mathsf{keys}} \cdot \lambda$ hybrid experiments.

Hybrid $H_{o,\hat{j},0}$ for $o \in \{0,1,\ldots,q_{\mathsf{keys}}\}$, $\hat{j} \in [\lambda]$: In this hybrid, for the first o keys, for $j \in \mathsf{diff} \cap [\hat{j}]$, the $\widetilde{\mathbf{t}}_{i^*}^{(j,0)}$ components are sampled uniformly at random, while the remaining components are sampled as in Game 3.i^*.3.

Hybrid $H_{o,\hat{j},1}$ for $o \in \{0,1,\ldots,q_{\mathsf{keys}}\}, \hat{j} \in [\lambda]$: In this hybrid, for all keys and $j \in \mathsf{diff}$, the $\widetilde{\mathbf{t}}_{i^*}^{(j,0)}$ components are sampled uniformly at random. For the first o queries and $j \in \mathsf{diff} \cap [\hat{j}]$, the $\widetilde{\mathbf{t}}_{i^*}^{(j,1)}$ components are sampled uniformly at random, while the remaining are sampled as in Game 3.i^*.3.

Clearly, $H_{0,0,0}$ corresponds to Game 3.i^*.3, $H_{q_{\mathsf{keys}},\lambda,1}$ is identical to Game 3.i^*.4, $H_{o-1,\lambda,b} \equiv H_{o,0,b}$ and $H_{q_{\mathsf{keys}},\lambda,0} \equiv H_{0,0,1}$. Let $a_{\mathcal{A},i,\hat{j},b}(\lambda)$ denote the advantage of $\mathcal{A}$ in $H_{i,\hat{j},b}$.

Claim 8.6. Assuming the $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ assumption, for any PPT adversary $\mathcal{A}$ making $q_{\mathsf{keys}}(\cdot)$ key queries, there exists a negligible function $\mathsf{n}_{o,\hat{j},0}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ and all indices $o \in [q_{\mathsf{keys}}]$ and $\hat{j} \in [\lambda]$, $a_{\mathcal{A},o,j-1,0} - a_{\mathcal{A},o,j,0} \le \mathsf{n}_{o,\hat{j},0}(\lambda)$.

Claim 8.7. Assuming the $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ assumption, for any PPT adversary $\mathcal{A}$ making $q_{\mathsf{keys}}(\cdot)$ key queries, there exists a negligible function $\mathsf{n}_{o,0,0}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ and all indices $o \in [q_{\mathsf{keys}}]$ and $\hat{j} \in [\lambda]$, $a_{\mathcal{A},o,\hat{j}-1,1} - a_{\mathcal{A},o,\hat{j},1} \le \mathsf{n}_{o,\hat{j},1}(\lambda)$.

The proofs of these claims are similar to the proof of Claim 8.4. If $\hat{j} \notin \mathsf{diff}$, then $H_{o,\hat{j}-1,b} \equiv H_{o,\hat{j},b}$. Else, we can reduce LWE to the indistinguishability of these two hybrids. ■

Lemma 8.18. For any PPT adversary $\mathcal{A}$ and $i^* \in [\ell-1]$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.4}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.(i^*+1).1}(\lambda) \le \mathrm{negl}(\lambda)$.

This proof is identical to the proof of Lemma 8.14.

Lemma 8.19. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.\ell.4}(\lambda) - \mathsf{Adv}_{\mathcal{A},4}(\lambda) \le \mathrm{negl}(\lambda)$.

Proof. The only difference between Game 3.ℓ.4 and Game 4 is that the $\mathbf{t}_{\ell+1}^{(j^*,\beta)}$ terms contain an additional noise term $\widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta)}$. The proof of this lemma is identical to the proof of Lemma 8.14 and follows via the smudging lemma (Lemma 2.1).

■

Lemma 8.20. Let $\sigma : \mathbb{N} \to \mathbb{R}^+$ and $q : \mathbb{N} \to \mathbb{N}$ be functions, and $\chi_s(\lambda) \equiv \mathcal{D}_{\sqrt{2}\sigma(\lambda)}$ and $\chi_{\mathsf{lwe}}(\lambda) \equiv \mathcal{D}_{\sigma(\lambda)}$ for each $\lambda \in \mathbb{N}$. Assuming the $\mathsf{LWE\text{-}ss}_{(n,q,\sigma_{\mathsf{lwe}})}$ assumption, for any PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},4}(\lambda) - \mathsf{Adv}_{\mathcal{A},5}(\lambda) \le \text{negl}(\lambda)$.

Proof. The only difference between Game 4 and Game 5 is in the key generation phase. In Game 4, for each key query, the term $\mathbf{t}_{\ell+1}^{(j^*,\beta^*)} = -\sum_{\alpha=1}^{\ell}\left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta^*)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta^*)}\right) + \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}^{(\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta^*)}} + \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta^*)} + \mathbf{e}_{\ell+1}^{(j^*,\beta^*)}$, and similarly the term $\mathbf{t}_{\ell+1}^{(j^*,1-\beta^*)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}^{(1-\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(1-\beta^*)}} + \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,1-\beta^*)} +$ other terms . In Game 5, for each key query, both $\mathbf{t}_{\ell+1}^{(j^*,0)}$ and $\mathbf{t}_{\ell+1}^{(j^*,1)}$ are set to be uniformly random. Using the short secrets version of LWE, we can switch both $\widetilde{\mathbf{s}} \cdot \mathbf{P}^{(\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta^*)}} + \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta^*)}$ and $\widetilde{\mathbf{s}} \cdot \mathbf{P}^{(1-\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(1-\beta^*)}} + \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,1-\beta^*)}$ to uniformly random matrices. This switch is possible because

- $\widetilde{\mathbf{s}}$ is chosen from χ_s^n, $\widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta^*)}, \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,1-\beta^*)}$ are chosen from χ_{lwe}^m
- $\widetilde{\mathbf{s}}$, $\widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta^*)}$ and $\widetilde{\mathbf{e}}_{\ell+1}^{(j^*,1-\beta^*)}$ are not required anywhere else in Game 4 or Game 5. Each of these three terms is chosen afresh for each key query.
- $\mathbf{P}^{(\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta^*)}}$ and $\mathbf{P}^{(1-\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(1-\beta^*)}}$ are uniformly random matrices.

Formally, we will show that Game 4 and Game 5 are computationally indistinguishable via a sequence of hybrid experiments. Let $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ denote the number of keys queried by $\mathcal{A}(1^\lambda)$. To prove that these two games are computationally indistinguishable, we will define q_{keys} hybrid experiments.

Hybrid H_o for $o \in \{0, 1, \ldots, q_{\mathsf{keys}}\}$: In this hybrid, for the first o keys, the $\mathbf{t}_{i^*+1}^{(j^*,0)}, \mathbf{t}_{lvl+1}^{(j^*,1)}$ components are sampled uniformly at random in the first o queries. For the remaining $q_{\mathsf{keys}} - o$ key queries, the keys are generated as in Game 4 in the remaining queries.

Clearly, H_0 corresponds to Game 4, while $H_{q_{\mathsf{keys}}}$ is identical to Game 5. Let $a_{\mathcal{A},i}(\lambda)$ denote the advantage of $\mathcal{A}$ in H_i.

Claim 8.8. Let $\sigma : \mathbb{N} \to \mathbb{R}^+$ and $q : \mathbb{N} \to \mathbb{N}$ be functions, and $\chi_s(\lambda) \equiv \mathcal{D}_{\sqrt{2}\sigma(\lambda)}$ and $\chi_{\mathsf{lwe}}(\lambda) \equiv \mathcal{D}_{\sigma(\lambda)}$ for each $\lambda \in \mathbb{N}$. Assuming the $\mathsf{LWE\text{-}ss}_{(n,q,\sigma_{\mathsf{lwe}})}$ assumption, for any PPT adversary $\mathcal{A}$ making $q_{\mathsf{keys}}(\cdot)$ key queries, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ and all indices $o \in [q_{\mathsf{keys}}]$, $a_{\mathcal{A},o-1} - a_{\mathcal{A},o} \le \text{negl}(\lambda)$.

Proof. Suppose there exists an adversary making q_{keys} key queries, and a non-negligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, there exists an index $o \in [q_{\mathsf{keys}}]$ such that $a_{\mathcal{A},o-1} - a_{\mathcal{A},o} \ge \eta(\lambda)$. We will use $\mathcal{A}$ to build a reduction algorithm $\mathcal{B}$ that breaks the $\mathsf{LWE\text{-}ss}_{(n,q,\sigma_{\mathsf{lwe}})}$ assumption.

The reduction algorithm receives (k, w, L) from the adversary, and sets the parameters as in H_{o-1}/H_o. It makes $2m$ queries to the LWE-ss challenger, and receives $\{(\mathbf{a}_j, u_j)\}_{j \le 2m}$. It chooses $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$ and j^* is the first position where tag^* and tag differ. Next, it sets $\widetilde{n}_i$ as in H_{o-1}/H_o, chooses $(\mathbf{M}_i, T_i) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q)$.

Challenge Phase The reduction algorithm receives challenge ciphertext BP^*, which specifies the reject state rej^*, and uses Mixed-SubEnc for computing the challenge ciphertext. Note that Mixed-SubEnc chooses $\mathbf{P}_{\ell+1,v}$ uniformly at random. The reduction algorithm sets $\mathbf{P}_{\ell+1,\mathsf{rej}^*}$ to be a matrix whose j^{th} column is $\mathbf{a}_j^T$. All other $\mathbf{P}_{\ell+1,v}$ are chosen uniformly at random.

Ciphertext Query The reduction algorithm receives ciphertext query BP, and uses $\mathsf{Mixed\text{-}SubEnc}$ for computing the ciphertext query. Let rej denote the reject state of BP. It sets $\mathbf{P}_{\ell+1,\mathsf{rej}}$ to be a matrix whose j^{th} column is $\mathbf{a}_{m+j}^T$. The remaining $\mathbf{P}_{\ell+1,v}$ matrices are chosen uniformly at random.

Key Queries The reduction algorithm first sets $\mathbf{u}^{\beta^*} = [u_1 \dots u_m]$ and $\mathbf{u}^{1-\beta^*} = [u_{m+1} \dots u_{2m}]$. For the first $o-1$ key queries, the $\mathbf{t}_{\ell+1}^{(j^*,\beta)}$ components are chosen uniformly at random. For the o^{th} key query, the reduction algorithm sets $\mathbf{t}_{\ell+1}^{(j^*,\beta^*)} = -\sum_{\alpha=1}^{\ell}\left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta^*)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta^*)}\right) + \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta^*)} + \mathbf{e}_{\ell+1}^{(j^*,\beta^*)} + \mathbf{u}^{\beta^*}$, and $\mathbf{t}_{\ell+1}^{(j^*,1-\beta^*)} = -\sum_{\alpha=1}^{\ell}\left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,1-\beta^*)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(1-\beta^*)}\right) + \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(1-\beta^*)} + \mathbf{e}_{\ell+1}^{(j^*,1-\beta^*)} + \mathbf{u}^{1-\beta^*}$. The remaining key queries are handled as in H_{o-1}/H_o.

Now, if all the u_j terms output by the LWE challenger are uniformly random, then $\mathbf{t}_{\ell+1}^{(j^*,\beta^*)}$ and $\mathbf{t}_{\ell+1}^{(j^*,1-\beta^*)}$ are uniformly random, and hence the reduction algorithm simulates H_o. If each $u_j = \widetilde{\mathbf{s}} \cdot \mathbf{a}_j^T + \widetilde{\mathbf{e}}_j$, then the reduction algorithm simulates H_{o-1}. ∎

∎

Lemma 8.21. Assuming the trapdoor system $\mathsf{LT}_{\mathsf{en}}$ satisfies $(q, \sigma_{\mathsf{pre}})$-row removal property, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},5}(\lambda) - \mathsf{Adv}_{\mathcal{A},5.1}(\lambda) \leq \text{negl}(\lambda)$.

Proof. Let us first consider the differences between Game 5 and Game 5.1.

- Set $S^{(1)}$: In Game 5, the challenger sets $S^{(1)} = \mathsf{comm} \times \{0,1\}^2$, while in Game 5.1, $S^{(1)} = \{(j,\beta,b) : j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\}$ ($\mathsf{tag}^*, \mathsf{tag}$ are chosen at the start of the security game, so these sets are well defined here). Also, $\widetilde{n}_1 = (4|\mathsf{comm}| + w)n$ in Game 5, while $\widetilde{n}_1 = (2|\mathsf{comm}| + w)n$ in Game 5.1.
- $\left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{i=1}$ matrices : In Game 5, the challenger chooses $(\mathbf{M}_1, T_1) \leftarrow \mathsf{EnTrapGen}(1^{(4|\mathsf{comm}|+w)n}, 1^m, q)$, while in Game 3.i^*.3, it chooses $(\mathbf{M}_1, T_1) \leftarrow \mathsf{EnTrapGen}(1^{(2|\mathsf{comm}|+w)n}, 1^m, q)$. As a result, in Game 5, it derives all $\left\{\mathbf{B}_{1,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in\mathsf{comm}\times\{0,1\}^2}$ from $\mathbf{M}_1$. In Game 5.1, the challenger chooses $\left\{\mathbf{B}_{1,b}^{(j,\beta)}\right\}_{j\in\mathsf{comm},\beta\neq\mathsf{tag}_j,b\in\{0,1\}}$ uniformly at random, while the remaining are derived from $\mathbf{M}_1$.
- Ciphertexts: Since the set $S^{(1)}$ is different in both games, the challenge ciphertext components $(\mathbf{U}_{1,0}^*, \mathbf{U}_{1,1}^*)$ and the ciphertext query components $(\mathbf{U}_{1,0}, \mathbf{U}_{1,1})$ are computed using $\mathbf{M}_1$ and T_1, which are computed differently in Game 5 and Game 5.1.

Let us now discuss why row removal property suffices for proving this lemma. Consider any matrix $\mathbf{U} \in \left\{\mathbf{U}_{1,0}^*, \mathbf{U}_{1,1}^*, \mathbf{U}_{1,0}, \mathbf{U}_{1,1}\right\}$, fix any $j \in \mathsf{comm}$, and let $\beta = \mathsf{tag}_j$. Then, from the definition of $\mathbf{D}_b^{(j,\beta)}$ and $\widetilde{\mathbf{D}}_b^{(j,\beta)}$ in $\mathsf{Mixed\text{-}SubEnc}$, it follows that $\mathbf{B}_{1,b}^{(j,1-\beta)} \cdot \mathbf{U}$ is a random matrix for both $b \in \{0,1\}$ (because $\mathsf{tag}_j^* = \beta$).

The proof of this lemma therefore follows using the row removal property (the reduction algorithm is similar to the one described in the proof of Lemma 8.7). ∎

Lemma 8.22. Assuming the trapdoor system $\mathsf{LT}_{\mathsf{en}}$ satisfies $(q, \sigma_{\mathsf{pre}})$-row removal property, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $i^* \in [\ell]$, $\mathsf{Adv}_{\mathcal{A},5.(i^*-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A},5.\ell}(\lambda) \leq \text{negl}(\lambda)$.

The proof of this lemma is identical to the proof of Lemma 8.22.

Lemma 8.23. Assuming the $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ assumption, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},5.\ell}(\lambda) - \mathsf{Adv}_{\mathcal{A},6.2}(\lambda) \leq \text{negl}(\lambda)$.

Proof. The proof of this lemma is similar to the proof of Lemma 8.8.

In Game 5.ℓ, for each key query x, for each $j \in \mathsf{comm}$, the component $\mathbf{t}_1^{(j,\beta)} = \mathbf{s}_1^{(j,\beta)}\mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)}$ and $\mathbf{t}_2^{(j,\beta)} = -\mathbf{s}_1^{(j,\beta)} \cdot \mathbf{C}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{s}_2^{(j,\beta)} \cdot \mathbf{B}_{2,\widetilde{x}_2}^{(j,\beta)} + \mathbf{e}_2^{(j,\beta)}$. In Game 6.1, then $\mathbf{t}_1^{(j,1-\mathsf{tag}_j)}, \mathbf{t}_2^{(j,1-\mathsf{tag}_j)} \leftarrow \mathbb{Z}_q^m$. Let $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ denote the number of keys queried by $\mathcal{A}(1^\lambda)$. To prove that these two games are computationally indistinguishable, we will define $q_{\mathsf{keys}} \cdot \lambda$ hybrid experiments.

Hybrid $H_{o,\hat{j},0}$ for $o \in \{0, 1, \ldots, q_{\mathsf{keys}}\}$, $\hat{j} \in [\lambda]$: In this hybrid, for the first o keys, for $j \in \mathsf{comm} \cap [\hat{j}]$, the $\mathbf{t}_1^{(j,1-\mathsf{tag}_j)}, \mathbf{t}_2^{(j,1-\mathsf{tag}_j)}$ components are sampled uniformly at random, while the remaining components are sampled as in Game 5.ℓ.

Hybrid $H_{o,\hat{j},1}$ for $o \in \{0, 1, \ldots, q_{\mathsf{keys}}\}, \hat{j} \in [\lambda]$: This hybrid is similar to the previous one, except that for $j = \hat{j} + 1$, it adds an additional χ_{lwe} noise to $\mathbf{t}_1^{(j,1-\mathsf{tag}_j)}$ and $\mathbf{t}_2^{(j,1-\mathsf{tag}_j)}$.

Clearly, $H_{0,0,0}$ corresponds to Game 5.ℓ, $H_{q_{\mathsf{keys}},\lambda,1}$ is identical to Game 6.2, and $H_{o-1,\lambda,0} \equiv H_{o,0,0}$. Let $a_{\mathcal{A},o,\hat{j},b}(\lambda)$ denote the advantage of $\mathcal{A}$ in $H_{o,\hat{j},b}$.

Claim 8.9. For every PPT adversary $\mathcal{A}$ making q_{keys} queries, there exists a negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $o \in [q_{\mathsf{keys}}]$ and $j \in [\lambda]$, $a_{\mathcal{A},o,\hat{j},0} - a_{\mathcal{A},o,\hat{j},1} \leq \text{negl}(\lambda)$.

The proof of this claim follows via the smudging lemma (Lemma 2.1).

Claim 8.10. Assuming the $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ assumption, for any PPT adversary $\mathcal{A}$ making $q_{\mathsf{keys}}(\cdot)$ key queries, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ and all indices $o \in [q_{\mathsf{keys}}]$ and $\hat{j} \in [\lambda]$, $a_{\mathcal{A},o,\hat{j}-1,1} - a_{\mathcal{A},o,\hat{j},0} \leq \text{negl}(\lambda)$.

Proof. The proof of this claim follows from the LWE assumption. The reduction algorithm makes $4m$ queries to the LWE challenger, and receives $\{\mathbf{a}_j, u_j\}_{j \in 4m}$. It sets $\mathbf{B}_{1,0}^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = [\mathbf{a}_1^T \ldots \mathbf{a}_m^T]$, $\mathbf{C}_{1,0}^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = [\mathbf{a}_{m+1}^T \ldots \mathbf{a}_{2m}^T]$, $\mathbf{B}_{1,1}^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = [\mathbf{a}_{2m+1}^T \ldots \mathbf{a}_{3m}^T]$, $\mathbf{C}_{1,1}^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = [\mathbf{a}_{3m+1}^T \ldots \mathbf{a}_{4m}^T]$. It also sets $\mathbf{u}_{1,0} = [u_1 \ldots u_m]$, $\mathbf{u}_{2,0} = [u_{m+1} \ldots u_{2m}]$, $\mathbf{u}_{1,1} = [u_{2m+1} \ldots u_{3m}]$, $\mathbf{u}_{2,1} = [u_{3m+1} \ldots u_{4m}]$.

For the o^{th} key query x, the reduction algorithm sets $\mathbf{t}_1^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = \mathbf{u}_{1,\widetilde{x}_1} + \mathbf{y}^{(\hat{j})} + \mathbf{e}_1^{(\hat{j},1-\mathsf{tag}_{\hat{j}})}$ and $\mathbf{t}_2^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = \mathbf{u}_{2,\widetilde{x}_1} + \mathbf{e}_2^{(\hat{j},1-\mathsf{tag}_{\hat{j}})}$. The rest of the key components can be handled without the LWE challenge terms. ■

■

Lemma 8.24. Assuming the $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ assumption, for any PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $i^* \in \{3, \ldots, \ell\}$, $\mathsf{Adv}_{\mathcal{A},6.i^*-1}(\lambda) - \mathsf{Adv}_{\mathcal{A},6.i^*}(\lambda) \leq \text{negl}(\lambda)$.

Proof. The only difference between Game 6.(i^*-1) and Game 6.i^* is with respect to the key queries. In Game 6.(i^*-1), for each $j \in \mathsf{comm}$, the challenger sets $\mathbf{t}_{i^*}^{(j,1-\mathsf{tag}_j)} = -\mathbf{s}_{i^*-1}^{(j,1-\mathsf{tag}_j)} \cdot \mathbf{C}_{i^*-1,\widetilde{x}_{i-1}}^{(j,1-\mathsf{tag}_j)} + \mathbf{s}_{i^*}^{(j,1-\mathsf{tag}_j)} \cdot \mathbf{B}_{i^*,\widetilde{x}_i}^{(j,1-\mathsf{tag}_j)} + \mathbf{e}_{i^*}^{(j,1-\mathsf{tag}_j)}$, while it switches these terms to random in Game 6.i^*.

The proof of this lemma uses (standard) LWE, similar to the proof of Lemma 8.23. One minor difference between this lemma and Lemma 8.23 is that in the previous lemma, the challenger switches both $\mathbf{t}_1^{(j,1-\mathsf{tag}_j)}$ and $\mathbf{t}_2^{(j,1-\mathsf{tag}_j)}$ (this is because the vector $\mathbf{s}_1^{(j,1-\mathsf{tag}_j)}$ is used for computing both these components). However, in this lemma, the reduction algorithm sets the LWE challenge's public vectors as the rows of $\mathbf{C}_{i^*,0}^{(j,1-\mathsf{tag}_j)}$ and $\mathbf{C}_{i^*,1}^{(j,1-\mathsf{tag}_j)}$, and the LWE challenge for setting $\mathbf{t}_{i^*}^{(j,1-\mathsf{tag}_j)}$. ■

Lemma 8.25. For any adversary $\mathcal{A}$ and any $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},6.(\ell+1)}(\lambda) = \mathsf{Adv}_{\mathcal{A},7.1}(\lambda)$.

Proof. The only difference between $\mathsf{Game}\ 6.(\ell+1)$ and $\mathsf{Game}\ 7.1$ is with respect to the $\{\mathbf{t}_1^{(j,\mathsf{tag}_j)}\}_{j\in\mathsf{comm}}$ components in key queries. In $\mathsf{Game}\ 6$, for each key query x, the challenger chooses $\left\{\mathbf{y}^{(j)}\right\}_{j\neq j^*}$ and sets $\mathbf{t}_1^{(j,\mathsf{tag}_j)} = \mathbf{s}_1^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\mathsf{tag}_j)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\mathsf{tag}_j)}$ for each $j \in \mathsf{comm}$. In $\mathsf{Game}\ 7.1$, the $\{\mathbf{t}_1^{(j,\mathsf{tag}_j)}\}_{j\in\mathsf{comm}}$ vectors are set to be uniformly random vectors.

Note that in $\mathsf{Game}\ 6.(\ell+1)$, the $\mathbf{y}^{(j)}$ terms are chosen afresh for each key, and $\mathbf{y}^{(j)}$ is only used in constructing $\mathbf{t}_1^{(j,\mathsf{tag}_j)}$ (recall $\mathbf{t}_1^{(j,1-\mathsf{tag}_j)}$ is uniformly random). As a result, the components $\{\mathbf{t}_1^{(j,\mathsf{tag}_j)}\}_{j\in\mathsf{comm}}$ are uniformly random vectors and therefore the secret keys in the two games are identically distributed. ∎

Lemma 8.26. Assuming the $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ assumption, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $i^* \in \{2,\ldots,\ell\}$, $\mathsf{Adv}_{\mathcal{A},7.i^*-1}(\lambda) - \mathsf{Adv}_{\mathcal{A},7.i^*}(\lambda) \leq \text{negl}(\lambda)$.

Proof. The only difference between $\mathsf{Game}\ 7.(i^*-1)$ and $\mathsf{Game}\ 7.i^*$ is with respect to the $\{\mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)}\}_{j\in\mathsf{comm}}$ components in key queries. In $\mathsf{Game}\ 7.(i^*-1)$, for each key query x, the challenger sets $\mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)} = -\mathbf{s}_{i^*-1}^{(j,\mathsf{tag}_j)} \cdot \mathbf{C}_{i^*,\widetilde{x}_{i^*-1}}^{(j,\mathsf{tag}_j)} + \mathbf{s}_{i^*}^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\mathsf{tag}_j)} + \mathbf{e}_{i^*}^{(j,\mathsf{tag}_j)}$ for each $j \in \mathsf{comm}$. In $\mathsf{Game}\ 7.i^*$, the $\{\mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)}\}_{j\in\mathsf{comm}}$ vectors are set to be uniformly random vectors. We will show that these two games are computationally indistinguishable via a hybrid argument. First, we will define $q_{\mathsf{keys}} \cdot \lambda$ hybrid experiments.

Hybrid $H_{o,\hat{j},0}$ for $o \in \{0,1,\ldots,q_{\mathsf{keys}}\}$, $\hat{j} \in [\lambda]$: In this hybrid, for the first o keys, for $j \in \mathsf{comm} \cap [\hat{j}]$, the $\mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)}$ components are sampled uniformly at random, while the remaining components are sampled as in $\mathsf{Game}\ 7.i^*$.

Hybrid $H_{o,\hat{j},1}$ for $o \in \{0,1,\ldots,q_{\mathsf{keys}}\}$, $\hat{j} \in [\lambda]$: This hybrid is similar to the previous one, except that for $j = \hat{j}+1$, it adds an additional χ_{lwe} noise to $\mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)}$.

Clearly, $H_{0,0,0}$ corresponds to $\mathsf{Game}\ 7.(i^*-1)$, $H_{q_{\mathsf{keys}},\lambda,1}$ is identical to $\mathsf{Game}\ 7.i^*$ and $H_{o-1,\lambda,0} \equiv H_{o,0,0}$. Let $a_{\mathcal{A},o,\hat{j},b}(\lambda)$ denote the advantage of $\mathcal{A}$ in $H_{o,\hat{j},b}$.

Claim 8.11. For every PPT adversary $\mathcal{A}$ making q_{keys} queries, there exists a $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $o \in [q_{\mathsf{keys}}]$ and $j \in [\lambda]$, $a_{\mathcal{A},o,\hat{j},0} - a_{\mathcal{A},o,\hat{j},1} \leq \text{negl}(\lambda)$.

The proof of this claim follows via the smudging lemma (Lemma 2.1).

Claim 8.12. Assuming the $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ assumption, for any PPT adversary $\mathcal{A}$ making $q_{\mathsf{keys}}(\cdot)$ key queries, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ and all indices $o \in [q_{\mathsf{keys}}]$ and $\hat{j} \in [\lambda]$, $a_{\mathcal{A},o,\hat{j}-1,1} - a_{\mathcal{A},o,\hat{j},0} \leq \text{negl}(\lambda)$.

Proof. The proof of this claim follows from the LWE assumption, where the reduction algorithm sets the LWE public vectors to be columns of $\mathbf{C}_{i^*,0}^{(\hat{j},\mathsf{tag}_{\hat{j}})}, \mathbf{C}_{i^*,1}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$, and the LWE challenge is used to set $\mathbf{t}_{i^*}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$. Note that the vector $\mathbf{s}_{i^*-1}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$ is used only for defining $\mathbf{t}_{i^*}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$. This is because this vector is chosen afresh for each key query, and in hybrids $H_{o,\hat{j}-1,1}/H_{o,\hat{j},0}$, the key component $\mathbf{t}_{i^*-1}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$ is already random. [30] ∎

∎

[30] If $\mathbf{t}_{i^*-1}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$ was not already switched to random, then $\mathbf{s}_{i^*-1}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$ would have been used to define it.

Lemma 8.27. Assuming $\mathsf{LT}_{\mathsf{en}}$ satisfies the $(q, \chi_{\mathsf{appr}}, \sigma_{\mathsf{pre}})$-target switching property and $(q, \sigma_{\mathsf{pre}})$-well sampledness of preimage, for any PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $i^* \in [\ell]$, $\mathsf{Adv}_{\mathcal{A},8.(i^*-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A},8.i^*}(\lambda) \leq \text{negl}(\lambda)$.

Lemma 8.28. First, let us discuss the differences between $\mathsf{Game}\ 8.(i^*-1)$ and $\mathsf{Game}\ 8.i^*$. In $\mathsf{Game}\ 8.(i^*-1)$, the challenge ciphertext components $\{\mathbf{U}^*_{i,b}\}_{i \geq i^*, b \in \{0,1\}}$ and query ciphertext components $\{\mathbf{U}_{i,b}\}_{i \geq i^*, b \in \{0,1\}}$ are computed using $\mathsf{Mixed\text{-}SubEnc}$, while the remaining are chosen from Gaussian distribution with parameter χ_{pre}. $\mathsf{Game}\ 8.i^*$ is similar to $\mathsf{Game}\ 8.(i^*-1)$, except for the challenge ciphertext components $\mathbf{U}^*_{i^*,0}, \mathbf{U}^*_{1,1}$ and the query ciphertext components $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$ are chosen from the Gaussian distribution with parameter σ_{pre}. To show that these games are indistinguishable, we will define a hybrid experiments H.

Hybrid H This hybrid is similar to $\mathsf{Game}\ 8.(i^*-1)$, except that the challenger computes $\{\mathbf{U}^*_{i^*,b}, \mathbf{U}_{i^*,b}\}_{b \in \{0,1\}}$ such that they map $\mathbf{M}_{i^*}$ to uniformly random matrices.

Let $\mathsf{Adv}_{\mathcal{A},H}$ denote the advantage of adversary $\mathcal{A}$ in hybrid H.

Claim 8.13. Assuming $\mathsf{LT}_{\mathsf{en}}$ satisfies the $(q, \chi_{\mathsf{appr}}, \sigma_{\mathsf{pre}})$-target switching property, for any PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $i^* \in [\ell]$, $\mathsf{Adv}_{\mathcal{A},8.(i^*-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A},H}(\lambda) \leq$ negl(λ).

Proof. The proof of this claim is similar to the proof of Lemma 8.10. First, let us discuss the reasons why the target switching property is applicable here. Let $S^{(i^*)}$ be defined as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$, $\mathsf{BP}^* = \{\pi^*_{i,b}\}_{(i,b) \in [\ell] \times \{0,1\}}$ and $\mathsf{BP} = \{\pi_{i,b}\}_{(i,b) \in [\ell] \times \{0,1\}}$ the challenge/query programs.

1. In both $\mathsf{Game}\ 8.(i^*-1)$ and hybrid H, the components $\left\{\mathbf{U}^*_{i,b}, \mathbf{U}_{i,b}\right\}_{i \in [i^*-1], b \in \{0,1\}}$ are chosen from a Gaussian distribution, and therefore these terms do not contain any information about the $\{\mathbf{P}_{i^*,v}\}_{v \in [w]}$ or $\left\{\mathbf{B}^{(j,\beta)}_{i^*,b}\right\}_{(j,\beta,b) \in S^{(i^*)}}$ matrices.

2. The components $\left\{\mathbf{U}^*_{i,b}, \mathbf{U}_{i,b}\right\}_{i > i^*, b \in \{0,1\}}$ do not contain any information about the $\{\mathbf{P}_{i^*,v}\}_{v \in [w]}$ or $\left\{\mathbf{B}^{(j,\beta)}_{i^*,b}\right\}_{(j,\beta,b) \in S^{(i^*)}}$ matrices (this follows from the construction).

3. The keys are all either random vectors, or computed in terms of the challenge/query ciphertext components, and therefore do not explicitly require $\{\mathbf{P}_{i^*,v}\}_{v \in [w]}$ or $\left\{\mathbf{B}^{(j,\beta)}_{i^*,b}\right\}_{(j,\beta,b) \in S^{(i^*)}}$ matrices.

Consider matrices $\left\{\mathbf{Z}^*_{0,b}, \mathbf{Z}^*_{1,b}, \mathbf{Z}_{0,b}, \mathbf{Z}_{1,b}\right\}_{b \in \{0,1\}}$ defined as follows:

$$\mathbf{Z}^*_{0,b} = \begin{bmatrix} \left\{\mathbf{C}^{(j,\beta)}_{i^*,b}\right\}_{(j,\beta,b) \in S^{(i^*)}} \\ \left\{\mathbf{P}_{i^*,\pi^*_{i^*,b}(v)}\right\}_{v \in [w]} \end{bmatrix} \mathbf{Z}^*_{1,b} = \left[\leftarrow \mathbb{Z}_q^{\widetilde{n}_i \times m}\right]$$

$$\mathbf{Z}_{0,b} = \begin{bmatrix} \left\{\mathbf{C}^{(j,\beta)}_{i^*,b}\right\}_{(j,\beta,b) \in S^{(i^*)}} \\ \left\{\mathbf{P}_{i^*,\pi_{i^*,b}(v)}\right\}_{v \in [w]} \end{bmatrix} \mathbf{Z}_{1,b} = \left[\leftarrow \mathbb{Z}_q^{\widetilde{n}_i \times m}\right]$$

The reduction algorithm sends $\left(1^{\widetilde{n}_i}, 1^m, \emptyset\right)$ to the target switching property challenger [31]. It does not receive any matrix from the challenger (since the challenge set is empty). Next, it chooses $(M_i, T_i) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q)$ for all $i > i^*$, and parses $\mathbf{M}_i$ as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$ to obtain $\left\{\mathbf{B}^{(j,\beta)}_{i,b}\right\}_{(j,\beta,b) \in S^{(i)}}$ and $\{\mathbf{P}_{i,v}\}_{v \in [w]}$ for all $i > i^*$. It receives BP^* as the challenge query from the adversary. The reduction algorithm sends $\mathbf{Z}^*_{0,b}, \mathbf{Z}^*_{1,b}$ to the target switching property challenger, and receives $\mathbf{U}_{i^*,b}$ in response. It

[31]Note that the set specified by the adversary in the target switching property game can be empty.

chooses the remaining components as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$ and sends the challenge ciphertext to the adversary.

Next, it receives the ciphertext query BP. It sends $\mathbf{Z}_{0,b}, \mathbf{Z}_{1,b}$ to the target switching property challenger, and receives $\mathbf{U}_{i^*,b}$. The remaining ciphertext components are chosen as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$ and sends the challenge ciphertext to the adversary. Finally, the adversary makes key queries. For each key query, the reduction algorithm sets $\left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}$ as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$ and sends them to the adversary. The adversary sends its guess, which the reduction algorithm forwards to the challenger.

Therefore, if there exists a PPT adversary $\mathcal{A}$ and a non-negligible function η such that $\mathsf{Adv}_{\mathcal{A},8.(i^*-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A},H}(\lambda) \geq \eta(\lambda)$ for all λ, then there exists a PPT algorithm $\mathcal{B}$ that breaks the target switching property. ■

Claim 8.14. Assuming $\mathsf{LT}_{\mathsf{en}}$ satisfies the $(q, \sigma_{\mathsf{pre}})$-well sampledness of pre-image, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},H}(\lambda) - \mathsf{Adv}_{\mathcal{A},8.i^*}(\lambda) \leq \mathrm{negl}(\lambda)$.

Proof. This proof follows directly from the (q, σ)-well sampledness of pre-image property. Suppose there exists a PPT adversary $\mathcal{A}$ and a reduction algorithm $\eta(\cdot)$ such that $\mathsf{Adv}_{\mathcal{A},H}(\lambda) - \mathsf{Adv}_{\mathcal{A},8.i^*}(\lambda) \geq \eta(\lambda)$ for all $\lambda \in \mathbb{N}$. Then there exists a reduction algorithm that breaks the (q, σ)-well sampledness of pre-image property.

The reduction algorithm sends $1^{\widetilde{n}_i}, 1^m, 1^{4m}$ to the challenger. Note that $m > \widetilde{n}_i \log q + \lambda$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda$ as required. It receives a matrix $\mathbf{U} \in Z_q^{m\times 4m}$, which it parses as $\mathbf{U} = \left[\mathbf{U}^*_{i^*,0} \mid \mathbf{U}^*_{i^*,1} \mid \mathbf{U}_{i^*,0} \mid \mathbf{U}_{i^*,1}\right]$. The reduction algorithm also chooses $(\mathbf{M}_i, T_i) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q)$ for all $i \neq i^*$.

On receiving the challenge ciphertext, it chooses the remaining ciphertext components as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$, and sends them to the adversary. Similarly, it handles the ciphertext query. Finally, for the key queries, the reduction algorithm handles them as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$. ■

Using these two claims, it follows that $\mathsf{Adv}_{\mathcal{A},8.(i^*-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A},8.i^*}(\lambda)$ is bounded by a negligible function.

Lemma 8.29. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},8.\ell}(\lambda) - \mathsf{Adv}_{\mathcal{A},9}(\lambda) \leq \mathrm{negl}(\lambda)$.

Proof. The only difference between these two hybrids is with respect to the key components $\left\{\mathbf{t}_\ell^{(j,\beta)}\right\}_{j\in\widehat{\mathsf{diff}},\beta\in\{0,1\}}$. In $\mathsf{Game}\ 8.\ell$, these vectors are computed as $\sum_{\alpha=1}^{\ell}\left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta}\right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \mathbf{e}^{(j,\beta)}_{\ell+1}$, while in $\mathsf{Game}\ 9$, this vector is set to be $\sum_{\alpha=1}^{\ell}\left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta}\right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \widetilde{\mathbf{e}}^{(j,\beta)}_{\ell+1} \cdot \prod_{\delta=2}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \mathbf{e}^{(j,\beta)}_{\ell+1}$, where $\widetilde{\mathbf{e}}^{(j,\beta)}_{\ell+1} \leftarrow \chi^m_{\mathsf{lwe}}$. Note that the $\mathbf{U}_{i,b}$ matrices are all drawn from the Gaussian distribution with parameter σ_{pre}, and therefore, with all but negligible probability, $\left\|\widetilde{\mathbf{e}}^{(j,\beta)}_{\ell+1} \cdot \prod_{\delta=2}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta}\right\| \leq m\sigma_{\mathsf{lwe}} \cdot (m\sigma_{\mathsf{pre}})^{\ell-1}$. Since $\sigma_{\mathsf{last}}/m\sigma_{\mathsf{lwe}} \cdot (m\sigma_{\mathsf{pre}})^{\ell-1} \geq 2^\lambda$, we can use the smudging lemma to argue that the statistical distance between these two games is at most $m \cdot q_{\mathsf{keys}} \cdot \mathrm{negl}_{\mathsf{smud}}(\lambda)$. ■

Lemma 8.30. Assuming the $\mathsf{LWE\text{-}sp}_{d,q,\sigma_{\mathsf{pre}},\chi_{\mathsf{lwe}}}$ assumption (see Assumption 3) holds (where $d(\lambda) = 6\lambda \cdot q$), for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},9}(\lambda) - \mathsf{Adv}_{\mathcal{A},10}(\lambda) \leq \mathrm{negl}(\lambda)$.

Proof. The only difference between $\mathsf{Game}\ 9$ and $\mathsf{Game}\ 10$ is with respect to the key components $\left\{\mathbf{t}_{\ell+1}^{(j,\beta)}\right\}_{j\in\widehat{\mathsf{diff}},\beta\in\{0,1\}}$. In $\mathsf{Game}\ 9$, for each key, these are computed as $\mathbf{t}^{(j,\beta)}_{\ell+1} = \sum_{\alpha=1}^{\ell}\left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta}\right) + \left(\mathbf{y}^{(j)} \cdot \mathbf{U}^{(\beta)}_{1,\widetilde{x}_1} + \widetilde{\mathbf{e}}^{(j,\beta)}_{\ell+1}\right) \prod_{\delta=2}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \mathbf{e}^{(j,\beta)}_{\ell+1}$, while in $\mathsf{Game}\ 10$, these vectors are uniformly random. To prove this claim, it suffices to switch $\left(\mathbf{y}^{(j)} \cdot \mathbf{U}^{(\beta)}_{1,\widetilde{x}_1} + \widetilde{\mathbf{e}}^{(j,\beta)}_{\ell+1}\right)$ to a uniformly random vector. Since $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$, $\mathbf{U}^{(\beta)}_{1,\widetilde{x}_1} \leftarrow \mathcal{D}^{m\times m}_{\mathbb{Z},\sigma_{\mathsf{pre}}}$ and $\widetilde{\mathbf{e}}^{(j,\beta)}_{\ell+1} \leftarrow \chi^m_{\mathsf{lwe}}$, we can use $\mathsf{LWE\text{-}sp}$ assumption as in the proof of Theorem 7.4. ■

8.4.5 Proving 1-Bounded Restricted Accept Indistinguishability

First, note that using the lemmas provided in Section 8.4.4, we can conclude that our construction satisfies 1-bounded complete accept indistinguishability security. Concretely, algorithms $\mathsf{SK\text{-}Enc}^*$ and KeyGen^*, that take as input the public parameters, are defined as follows: $\mathsf{SK\text{-}Enc}^*$ is same as the standard encryption algorithm Enc (that is, it outputs random Gaussian matrices), and KeyGen^* on input x outputs secret key as $\left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$, where all $\mathbf{t}_i^{(j,\beta)}$ are sampled uniformly at random from $\mathbb{Z}_q^m$. Since in Game 10 the challenger is already using $\mathsf{SK\text{-}Enc}^*$ and KeyGen^* to answer corresponding queries, therefore using lemmas in Section 8.4.4 we can argue 1-bounded complete accept indistinguishability security.

Lastly, to finish the proof we only need to argue that the probability adversary outputs 1 in the 1-bounded *restricted* accept indistinguishability security game, when the challenger computes challenge ciphertext as a normal FE ciphertext instead of encrypting BP^*, is negligibly close to the probability adversary outputs 1 in Game 10. Note that this again follows from the lemmas in Section 8.4.4. Or in other words, it follows from the fact that our construction satisfies 1-bounded complete accept indistinguishability security. This is because if an adversary can distinguish Game 10 from the scenario described above, then we could come up with a reduction algorithm that breaks 1-bounded complete accept indistinguishability security of our construction.

The idea is straightforward. The reduction algorithm will simply forwards messages (back and forth) between the attacker and complete accept indistinguishability challenger, except the following changes:

- The reduction algorithm does not forward the adversary's challenge program BP^* as its challenge query to complete accept indistinguishability challenger. Instead it runs the normal encryption algorithm Enc and sends the output back to adversary as its challenge ciphertext.
- Also, the reduction algorithm sends the adversary's post-challenge encryption query (if any) to the challenger as its challenge query and forwards challenger's response to the adversary.
- And, the reduction algorithm does not make any post-challenge encryption query.
- Finally, it outputs whatever the adversary outputs.

Clearly the reduction algorithm perfectly simulates the indistinguishability experiment (between Game 10 and the scenario described above), thus if adversary's advantage is non-negligible, then the reduction algorithm also breaks 1-bounded complete accept indistinguishability security with non-negligible probability. This completes the proof.

Acknowledgements

We thank Chris Peikert for useful discussions about the Micciancio-Peikert lattice trapdoors [MP12].

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical ibe. In *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO'10, pages 98–115, Berlin, Heidelberg, 2010. Springer-Verlag.

[ABP+17] Shweta Agrawal, Sanjay Bhattacherjee, Duong Hieu Phan, Damien Stehlé, and Shota Yamada. Efficient public trace and revoke from standard assumptions: Extended abstract. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2277–2293, 2017.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, pages 595–618, 2009.

[ADGM16] Daniel Apon, Nico Döttling, Sanjam Garg, and Pratyay Mukherjee. Cryptanalysis of indistinguishability obfuscations of circuits over ggh13. Cryptology ePrint Archive, Report 2016/1003, 2016.

[ADM$^+$07] Michel Abdalla, Alexander W. Dent, John Malone-Lee, Gregory Neven, Duong Hieu Phan, and Nigel P. Smart. Identity-based traitor tracing. In *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, pages 361–376, 2007.

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 500–518, 2013.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, pages 1–9, 1999.

[AJW11] Gilad Asharov, Abhishek Jain, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. Cryptology ePrint Archive, Report 2011/613, 2011. http://eprint.iacr.org/2011/613.

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In *Annual Cryptology Conference*, 2016.

[Bar86] D A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, 1986.

[BDFP86] Allan Borodin, Danny Dolev, Faith E. Fich, and Wolfgang J. Paul. Bounds for width two branching programs. *SIAM J. Comput.*, 15(2):549–560, 1986.

[BF99] Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 338–353, 1999.

[BF11] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *International Workshop on Public Key Cryptography*, 2011.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 533–556, 2014.

[BGH$^+$15] Zvika Brakerski, Craig Gentry, Shai Halevi, Tancrède Lepoint, Amit Sahai, and Mehdi Tibouchi. Cryptanalysis of the quadratic zero-testing of GGH. *IACR Cryptology ePrint Archive*, 2015.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 410–428, 2013.

[BLP$^+$13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 575–584, 2013.

[BLSV17] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous ibe, leakage resilience and circular security from new assumptions. Cryptology ePrint Archive, Report 2017/967, 2017. http://eprint.iacr.org/2017/967.

[BN08] Dan Boneh and Moni Naor. Traitor tracing with constant size ciphertext. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 501–510, 2008.

[BP08] Olivier Billet and Duong Hieu Phan. Efficient traitor tracing from collusion secure codes. In *Information Theoretic Security, Third International Conference, ICITS 2008, Calgary, Canada, August 10-13, 2008, Proceedings*, pages 171–182, 2008.

[BS15] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 306–324, 2015.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, pages 573–592, 2006.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 171–190, 2015.

[BVWW16] Zvika Brakerski, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Obfuscating conjunctions under entropic ring lwe. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, 2016.

[BW06] Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 211–220, 2006.

[BWZ14] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. Cryptology ePrint Archive, Report 2014/930, 2014.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 480–499, 2014.

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained prfs for nc1 from lwe. In *EUROCRYPT*, 2017.

[CFL$^+$16] Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new clt multilinear map over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2016.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, pages 257–270, 1994.

[CFNP00] Benny Chor, Amos Fiat, Moni Naor, and Benny Pinkas. Tracing traitors. *IEEE Trans. Information Theory*, 46(3):893–910, 2000.

[CGH$^+$15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, 2015.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 523–552, 2010.

[CHL$^+$15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 3–12, 2015.

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low-level encoding of zero. *LMS Journal of Computation and Mathematics*, 2016.

[CLLT16] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, 2016.

[CLLT17] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over CLT13. In *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*, 2017.

[CLT14] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of two candidate fixes of multilinear maps over the integers. Cryptology ePrint Archive, Report 2014/975, 2014.

[CPP05] Hervé Chabanne, Duong Hieu Phan, and David Pointcheval. Public traceability in traitor tracing schemes. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 542–558, 2005.

[DG17a] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 537–569, 2017.

[DG17b] Nico Dttling and Sanjam Garg. From selective ibe to full ibe and selective hibe. TCC, 2017.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, pages 265–284, 2006.

[DNR$^+$09] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil Vadhan. On the complexity of differentially private data release: Efficient algorithms and hardness results. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 381–390, New York, NY, USA, 2009. ACM.

[FNP07] Nelly Fazio, Antonio Nicolosi, and Duong Hieu Phan. Traitor tracing with optimal transmission rate. In *Information Security, 10th International Conference, ISC 2007, Valparaíso, Chile, October 9-12, 2007, Proceedings*, pages 71–88, 2007.

[Fre10] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 44–61, Berlin, Heidelberg, 2010. Springer-Verlag.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indstinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *TCC*, 2015.

[GKRW17] Rishab Goyal, Venkata Koppula, Andrew Russell, and Brent Waters. Risky traitor tracing and new differential privacy negative results. Cryptology ePrint Archive, Report 2017/1117, 2017.

[GKSW10] Sanjam Garg, Abishek Kumarasubramanian, Amit Sahai, and Brent Waters. Building efficient fully collusion-resilient traitor tracing and revocation schemes. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 121–130, New York, NY, USA, 2010. ACM.

[GKW17a] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 612–621, 2017.

[GKW17b] Rishab Goyal, Venkata Koppula, and Brent Waters. Separating semantic and circular security for symmetric-key bit encryption from the learning with errors assumption. In *EUROCRYPT*, 2017.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, 2012.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

[Hal15] Shai Halevi. Graded encoding, variations on a scheme. Cryptology ePrint Archive, Report 2015/866, 2015.

[HJ16] Yupu Hu and Huiwen Jia. Cryptanalysis of ggh map. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2016.

[KD98] Kaoru Kurosawa and Yvo Desmedt. Optimum traitor tracing and asymmetric schemes. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, pages 145–157, 1998.

[KMUW17] Lucas Kowalczyk, Tal Malkin, Jonathan Ullman, and Daniel Wichs. Hardness of non-interactive differential privacy from one-way functions. Cryptology ePrint Archive, Report 2017/1107, 2017.

[KY02a] Aggelos Kiayias and Moti Yung. Traitor tracing with constant transmission rate. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, pages 450–465, 2002.

[KY02b] Kaoru Kurosawa and Takuya Yoshida. Linear code implies public-key traitor tracing. In *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*, pages 172–187, 2002.

[LPSS14] San Ling, Duong Hieu Phan, Damien Stehlé, and Ron Steinfeld. Hardness of k-lwe and applications in traitor tracing. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 315–334, 2014.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 700–718, 2012.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, April 2007.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over ggh13. In *Annual Cryptology Conference*, 2016.

[NP98] Moni Naor and Benny Pinkas. Threshold traitor tracing. In *Advances in CryptologyCRYPTO'98*, pages 502–517. Springer, 1998.

[NWZ16] Ryo Nishimaki, Daniel Wichs, and Mark Zhandry. Anonymous traitor tracing: How to embed arbitrary information in a key. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 388–419, 2016.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 333–342, 2009.

[PST06] Duong Hieu Phan, Reihaneh Safavi-Naini, and Dongvu Tonien. Generic construction of hybrid public key traitor tracing with full-public-traceability. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, pages 264–275, 2006.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93, 2005.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 463–472, New York, NY, USA, 2010. ACM.

[SSW01] Jessica Staddon, Douglas R. Stinson, and Ruizhong Wei. Combinatorial properties of frameproof and traceability codes. *IEEE Trans. Information Theory*, 47(3):1042–1049, 2001.

[SW98] Douglas R. Stinson and Ruizhong Wei. Combinatorial properties and constructions of traceability schemes and frameproof codes. *SIAM J. Discrete Math.*, 11(1):41–53, 1998.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 600–611, 2017.

[Yao86] Andrew Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167, 1986.

A Background: Attribute-Based Encryption

A.1 Key-Policy Attribute Based Encryption

A key-policy attribute based encryption (KP-ABE) scheme $\mathcal{ABE}$, for set of attribute spaces $\mathcal{X} = \{\mathcal{X}_\kappa\}_\kappa$, predicate classes $\mathcal{C} = \{\mathcal{C}_\kappa\}_\kappa$ and message spaces $\mathcal{M} = \{\mathcal{M}_\kappa\}_\kappa$, consists of four polytime algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{pp}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ and a functionality index κ, and outputs the public parameters pp and the master secret key msk.
- $\mathsf{Enc}(\mathsf{pp}, x, m) \to \mathsf{ct}$. The encryption algorithm takes as input public parameters pp, an attribute $x \in \mathcal{X}_\kappa$ and a message $m \in \mathcal{M}_\kappa$. It outputs a ciphertext ct.
- $\mathsf{KeyGen}(\mathsf{msk}, C) \to \mathsf{sk}_C$. The key generation algorithm takes as input master secret key msk and a predicate $C \in \mathcal{C}_\kappa$. It outputs a secret key sk_C.
- $\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) \to m$ or $\perp$. The decryption algorithm takes as input a secret key sk_C and a ciphertext ct. It outputs either a message $m \in \mathcal{M}_\kappa$ or a special symbol $\perp$.

Correctness. A key-policy attribute based encryption scheme is said to be correct if there exists negligible functions $\mathrm{negl}_1(\cdot), \mathrm{negl}_2(\cdot)$ such that for all $\lambda, \kappa \in \mathbb{N}$, for all $x \in \mathcal{X}_\kappa$, $C \in \mathcal{C}_\kappa$, $m \in \mathcal{M}_\kappa$, the following holds

$$C(x) = 1 \Rightarrow \Pr\left[\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa); \\ \mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, x, m) \end{array}\right] \geq 1 - \mathrm{negl}_1(\lambda),$$

$$C(x) = 0 \Rightarrow \Pr\left[\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) = \perp \; \begin{array}{c} (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa); \\ \mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, x, m) \end{array}\right] \geq 1 - \mathrm{negl}_2(\lambda).$$

Security. The standard notion of security for a KP-ABE scheme is that of full or adaptive security. It is formally defined as follows.

Definition A.1. A key-policy attribute based encryption scheme $\mathcal{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be fully secure if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\left| \Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b : \begin{array}{c} 1^\kappa \leftarrow \mathcal{A}(1^\lambda); (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa) \\ ((m_0, m_1), x) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pp}) \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, x, m_b) \end{array}\right] - \frac{1}{2} \right| \leq \mathrm{negl}(\lambda)$$

where every predicate query C, made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle, must satisfy the condition that $C(x) = 0$.

In this work, we only require the scheme to achieve selective security, which is formally defined as follows.

Definition A.2. A key-policy attribute based encryption scheme $\mathcal{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be selectively secure if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\left| \Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b : \begin{array}{c} (1^\kappa, x) \leftarrow \mathcal{A}(1^\lambda); (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa) \\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pp}) \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, x, m_b) \end{array}\right] - \frac{1}{2} \right| \leq \mathrm{negl}(\lambda)$$

where every predicate query C, made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle, must satisfy the condition that $C(x) = 0$.

B ABE and Mixed FE to PLBE: Preserving Perfect Correctness

In this section, we give an alternate construction for constructing PLBE such that if the underlying ABE scheme achieves *perfect* correctness, then so does the PLBE scheme even if the mixed FE scheme is *not* perfectly correct. Since existing ABE schemes [GVW13, BGG$^+$14] can be made perfectly correct by appropriately truncating noise distributions used, thus this gives a pathway to get perfect correctness under LWE. Note that in the construction described in Section 6.1 only achieves perfect correctness when both underlying ABE and mixed FE scheme are perfectly correct. This is because the policy circuit is the mixed FE decryption circuit, and thus in order to guarantee perfect correctness, we need the minimum requirement that the mixed FE normal ciphertexts always decrypt to 1. Below we give the main idea to obtain perfect correctness.

Outline. At a very high level, the idea is to encrypt the message m under two independent ABE systems such that at least one of ciphertext components can always be decrypted to obtain the underlying message. To this end, during setup we sample two ABE key pairs $(\mathsf{abe.pp}_b, \mathsf{abe.msk}_b)$ (for $b \in \{0,1\}$) and a mixed FE key pair $(\mathsf{mixed.pp}, \mathsf{mixed.msk})$. To generate the secret key for i^{th} user, we generate a mixed FE secret key $\mathsf{mixed.sk}_i$ for message i, and later compute two ABE keys $\mathsf{abe.sk}_{i,0}, \mathsf{abe.sk}_{i,1}$ for predicates $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$ and $\overline{\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)}$ using $\mathsf{abe.msk}_0, \mathsf{abe.msk}_1$, respectively. Here $\overline{\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)}$ denotes the circuit that first decrypts the input using key $\mathsf{mixed.sk}_i$ and later applies a "not" gate (i.e., outputs the complement). Now the PLBE ciphertexts will consist of two parts, one for each ABE sub-system. For PLBE normal encryption, one computes two ciphertexts ct_b (for $b \in \{0,1\}$) as encryptions of message m under attributes $\mathsf{ct}_{\mathsf{attr}}$ using parameters $\mathsf{abe.pp}_b$, where $\mathsf{ct}_{\mathsf{attr}}$ is computed as before. Now for encrypting a message to index i, the encryption algorithm behaves differently in that it computes ct_0 as before, but ct_1 will now be an encryption of message 0 under the same attributes. The reason for not encrypting the message m in the second component of the index ciphertext becomes clear while proving security.

Now for arguing perfect correctness, we observe that it should be the case that either $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \mathsf{ct}_{\mathsf{attr}}) = 0$ or $= 1$. Thus, at least one of the PLBE normal ciphertext components could be correctly decrypted. Note that for such an argument we only require that ABE is perfectly correct. Next, the security proof is similar to that in Section 6.3, except for arguing normal hiding security of our construction we need to rely on both the ABE security as well as the weak accept indistinguishability property of mixed FE scheme. The main idea is that by correctness of FE scheme, we can say that with all but negligible probability the attribute used in the second component of challenge ciphertext is not satisfied by any of the ABE keys queried. Thus, as our first hybrid argument, we could use ABE security to switch second challenge ciphertext component to an encryption of 0 instead of message m. The remaining is identical to as before with the only modification being that the reduction algorithm needs to generate the ABE keys for the second component on its own during the entire reduction. Below we describe our construction $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ for messages spaces $\{\mathcal{M}_\kappa\}_\kappa$ in detail.

B.1 Construction

Let $\mathcal{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.Enc}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Dec})$ be a key-policy attribute based encryption scheme for set of attribute spaces $\{\mathcal{X}_\kappa\}_\kappa$, predicate classes $\{\mathcal{C}_\kappa\}_\kappa$ and message spaces $\{\mathcal{M}_\kappa\}_\kappa$, and $\mathsf{Mixed\text{-}FE} = (\mathsf{Mixed.Setup}, \mathsf{Mixed.Enc}, \mathsf{Mixed.SK\text{-}Enc}, \mathsf{Mixed.KeyGen}, \mathsf{Mixed.Dec})$ be a mixed functional encryption scheme, for function classes $\{\mathcal{F}_\kappa\}_\kappa$ and message space $\{\mathcal{I}_\kappa\}_\kappa$, with ciphertexts of length $\ell(\lambda, \kappa)$. For every n, let $\kappa = \kappa(n)$ be the lexicographically smallest functionality index such that every string of length $\log(n)$ can be uniquely represented in message space $\mathcal{I}_\kappa$ (i.e., $\{0,1\}^{\log(n)} \subseteq \mathcal{I}_\kappa$), and function class $\mathcal{F}_\kappa$ contains the "comparison" ($>$) operator. Also, let $\widetilde{\kappa} = \widetilde{\kappa}(\lambda, \kappa)$ be the lexicographically smallest functionality index such that every string of length $\ell(\lambda, \kappa)$ can be uniquely represented in attribute class $\mathcal{X}_{\widetilde{\kappa}}$ (i.e., $\{0,1\}^{\ell(\lambda,\kappa)} \subseteq \mathcal{X}_{\widetilde{\kappa}}$), and $\mathcal{C}_{\widetilde{\kappa}}$ contains mixed FE decryption circuit (as well its complement circuit) corresponding to functionality index κ. Below we describe our construction.

- $\mathsf{Setup}(1^\lambda, 1^n) \to \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i \leq n}\right)$. The setup algorithm runs $\mathsf{ABE.Setup}$ and $\mathsf{Mixed.Setup}$ to generate ABE and mixed FE public parameters and master secret key as $(\mathsf{abe.pp}, \mathsf{abe.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\widetilde{\kappa}})$

and $(\mathsf{mixed.pp}, \mathsf{mixed.msk}) \leftarrow \mathsf{Mixed.Setup}(1^\lambda, 1^\kappa)$. Next, it runs $\mathsf{Mixed.KeyGen}$ to generate n mixed secret keys $\mathsf{mixed.sk}_i$ as

$$\forall\ i \leq n, \quad \mathsf{mixed.sk}_i \leftarrow \mathsf{Mixed.KeyGen}(\mathsf{mixed.msk}, i).$$

Let $C^0_{\mathsf{mixed.sk}_i}$ denote the circuit $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$, and $C^1_{\mathsf{mixed.sk}_i}$ denote the circuit $\overline{\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)}$, i.e. $C^1_{\mathsf{mixed.sk}_i}$ is the $\mathsf{Mixed\text{-}FE}$ decryption circuit with key $\mathsf{mixed.sk}_i$ hardwired and a "not" gate applied on the output of decryption. Next, it computes $2n$ ABE secret keys $\mathsf{abe.sk}_{i,b}$ as

$$\forall\ i \leq n, b \in \{0,1\}, \quad \mathsf{abe.sk}_{i,b} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}_b, C^b_{\mathsf{mixed.sk}_i}).$$

Finally, it sets $\mathsf{pp} = (\mathsf{abe.pp}_0, \mathsf{abe.pp}_1, \mathsf{mixed.pp})$, $\mathsf{msk} = (\mathsf{abe.msk}_0, \mathsf{abe.msk}_1, \mathsf{mixed.msk})$ and $\mathsf{sk}_i = (\mathsf{abe.sk}_{i,0}, \mathsf{abe.sk}_{i,1})$ for $i \leq n$.

- $\mathsf{Enc}(\mathsf{pp}, m) \rightarrow \mathsf{ct}$. Let $\mathsf{pp} = (\mathsf{abe.pp}_0, \mathsf{abe.pp}_1, \mathsf{mixed.pp})$. The encryption algorithm first computes $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.Enc}(\mathsf{mixed.pp})$. Next, it encrypts message m as $\mathsf{ct}_b \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}_b, \mathsf{ct}_{\mathsf{attr}}, m)$ for $b \in \{0,1\}$, and outputs ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$.
- $\mathsf{Enc\text{-}index}(\mathsf{msk}, m, i) \rightarrow \mathsf{ct}$. Let $\mathsf{msk} = (\mathsf{abe.msk}_0, \mathsf{abe.msk}_1, \mathsf{mixed.msk})$ and comp_i denote the comparison function $\overset{?}{>} i$, i.e. $\mathsf{comp}_i(x) = 1$ iff $x > i$. The encryption algorithm first computes $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.SK\text{-}Enc}(\mathsf{mixed.msk}, \mathsf{comp}_i)$. Next, it encrypts message m as $\mathsf{ct}_0 \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}_0, \mathsf{ct}_{\mathsf{attr}}, m)$ and $\mathsf{ct}_1 \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}_1, \mathsf{ct}_{\mathsf{attr}}, 0)$, and outputs ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \rightarrow m$ or $\perp$. Let $\mathsf{sk} = (\mathsf{sk}_0, \mathsf{sk}_1)$ and $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$. The decryption algorithm runs $\mathsf{ABE.Dec}$ on ciphertexts ct_b using key sk_b as $y_b = \mathsf{ABE.Dec}(\mathsf{sk}_b, \mathsf{ct}_b)$ for $b \in \{0,1\}$. If $y_0 \neq \perp$, it outputs y_0. Otherwise, it sets y_1 as the output of decryption.

B.2 Correctness

For all $\lambda, n \in \mathbb{N}$, message $m \in \mathcal{M}_\lambda$, public parameters and master secret keys $(\mathsf{abe.pp}_b, \mathsf{abe.msk}_b) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\widetilde{\kappa}})$ (for $b \in \{0,1\}$), $(\mathsf{mixed.pp}, \mathsf{mixed.msk}) \leftarrow \mathsf{Mixed.Setup}(1^\lambda, 1^\kappa)$, the secret keys $\mathsf{sk}_{i,b}$ for $i \leq n, b \in \{0,1\}$ are simply the ABE keys $\mathsf{abe.sk}_{i,b} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}_b, C^b_{\mathsf{mixed.sk}_i})$. For any index $i \leq n$, consider the following two cases:

1. **Normal encryption.** For any ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$ computed as $\mathsf{ct}_b \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}_b, \mathsf{ct}_{\mathsf{attr}}, m)$ (for $b \in \{0,1\}$), where $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.Enc}(\mathsf{mixed.pp})$, we know that either $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \mathsf{ct}_{\mathsf{attr}}) = 1$ or $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \mathsf{ct}_{\mathsf{attr}}) = 0$. In other words, either $C^b_{\mathsf{mixed.sk}_i}(\mathsf{ct}_{\mathsf{attr}}) = 1$ for some bit $b \in \{0,1\}$. Therefore, by perfect correctness of ABE scheme, we have that $\mathsf{ABE.Dec}(\mathsf{abe.sk}_{i,b}, \mathsf{ct}_b) = m$ for some bit $b \in \{0,1\}$. Therefore, the PLBE decryption algorithm always decrypts the normal ciphertexts correctly.
2. **Index encryption.** This is identical to the argument provided in Section 6.2. Note that perfect correctness for PLBE only requires perfect decryption in the case of normal encryption. Thus, it is sufficient to prove statistical correctness in the case of index encryption.

Therefore, PLBE scheme is perfectly correct.

B.3 Security

The proof of security is almost identical to that provided in Section 6.3, except to argue normal hiding security of our construction, we first need to use ABE security of the auxiliary sub-system (for $b = 1$) and statistical correctness property of underlying Mixed FE scheme simultaneously to switch the encryption of challenge message m^* to 0. Rest of the proof is identical.

C Hybrids

C.1 Detailed Hybrid Experiments for Theorem 7.2

Hybrid H_0 : This corresponds to the original game (as per Definition 7.1, with the single row removal restriction) with $b = 0$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $(\mathbf{B}_1, T_{\mathbf{B}_1}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lceil m/2 \rceil}, q)$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.
   (b) It also chooses a vector $\mathbf{p} \leftarrow \mathbb{Z}_q^m$, and sets matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ as $\mathbf{A} = \mathsf{Arrange}(\mathbf{B}, \mathbf{p}, [n] \setminus \{i\})$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1) \times t}$. The challenger responds to each query as follows.
   (a) It chooses $\mathbf{W} \leftarrow \mathbb{Z}_q^{(n-1) \times t}$ and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{B}_1, T_{\mathbf{B}_1}, \sigma, \mathbf{W})$.
   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ (which is equal to $\mathbf{C} - \mathbf{W}$), and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.
   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.
3. The adversary outputs a bit b'.

Hybrid H_1 : In this experiment, the challenger chooses $\mathbf{U}_1$ to be a random Gaussian matrix with parameter σ for each query.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $(\mathbf{B}_1, T_{\mathbf{B}_1}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lceil m/2 \rceil}, q)$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.
   (b) It also chooses a vector $\mathbf{p} \leftarrow \mathbb{Z}_q^m$, and sets matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ as $\mathbf{A} = \mathsf{Arrange}(\mathbf{B}, \mathbf{p}, [n] \setminus \{i\})$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1) \times t}$. The challenger responds to each query as follows.
   (a) It samples $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.
   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$, and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.
   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.
3. The adversary outputs a bit b'.

Hybrid H_2 : In this hybrid, the challenger chooses $\mathbf{B}_1$ uniformly at random, instead of choosing it using $\mathsf{TrapGen}$. At this point, note that the left half of $\mathbf{A}$ is a uniformly random matrix.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $\mathbf{B}_1 \leftarrow \mathbb{Z}_q^{(n-1) \times \lceil m/2 \rceil}$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.
   (b) It also chooses a vector $\mathbf{p} \leftarrow \mathbb{Z}_q^m$, and sets matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ as $\mathbf{A} = \mathsf{Arrange}(\mathbf{B}, \mathbf{p}, [n] \setminus \{i\})$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.
   (a) It samples $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2\rceil \times t}$.
   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$, and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.
   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{bmatrix}$ to $\mathcal{A}$.
3. The adversary outputs a bit b'.

Hybrid H_3 : This hybrid involves syntactic changes. The challenger chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n\times\lceil m/2\rceil}$, and derives $\mathbf{B}_1$ by removing the i^{th} row of $\mathbf{A}_1$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n\times\lceil m/2\rceil}$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2\rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n]\setminus\{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.
   (b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2\rfloor}$, and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n]\setminus\{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.
   (a) It samples $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2\rceil \times t}$.
   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$, and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.
   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{bmatrix}$ to $\mathcal{A}$.
3. The adversary outputs a bit b'.

Hybrid H_4 : In this hybrid, the challenger chooses the left half of $\mathbf{A}$ using $\mathsf{TrapGen}$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2\rceil}, q\right)$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2\rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n]\setminus\{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.
   (b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2\rfloor}$, and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n]\setminus\{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.
   (a) It samples $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2\rceil \times t}$.
   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$, and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.
   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{bmatrix}$ to $\mathcal{A}$.
3. The adversary outputs a bit b'.

Hybrid H_5 : In this hybrid, the challenger chooses $\mathbf{U}_1$ using $\mathsf{SamplePre}$ for each query.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.
   (b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2 \rfloor}$, and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n] \setminus \{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.
   (a) It chooses $\mathbf{W}' \leftarrow \mathbb{Z}_q^{n\times t}$, sets $\mathbf{W} = \mathsf{Restrict}(\mathbf{W}', [n]\setminus\{i\})$, and samples $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.
   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$, and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.
   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.
3. The adversary outputs a bit b'.

Hybrid H_6 : This hybrid represents a syntactic change, in which the challenger, for each query, chooses $\mathbf{Y}$ as a uniformly random matrix, and set $\mathbf{W} = \mathbf{C} - \mathbf{Y} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.
   (b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2 \rfloor}$, and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n] \setminus \{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.
   (a) It chooses $\mathbf{Y} \leftarrow \mathbb{Z}_q^{(n-1)\times t}$, and samples $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.
   (b) Next, it sets $\mathbf{W} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$ (which is equal to $\mathbf{C} - \mathbf{Y}$), chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{W}' = \mathsf{Arrange}(\mathbf{W}, \mathbf{w}, [n] \setminus \{i\})$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.
   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.
3. The adversary outputs a bit b'.

Hybrid H_7 : In this hybrid experiment, the challenger chooses $\mathbf{U}_2$ from a Gaussian distribution with parameter σ.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.
   (b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2 \rfloor}$, and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n] \setminus \{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.

   (a) It samples $\mathbf{U}_2 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lfloor m/2 \rfloor \times t}$.

   (b) Next, it sets $\mathbf{W} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$, chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{W}' = \mathsf{Arrange}(\mathbf{W}, \mathbf{w}, [n] \setminus \{i\})$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid H_8 : In this hybrid, the challenger chooses matrix $\mathbf{B}_2$ uniformly at random. Note that this means $\mathbf{A}_2$ is uniformly random in this hybrid.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.

   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $\mathbf{B}_2 \leftarrow \mathbb{Z}_q^{(n-1)\times \lfloor m/2 \rfloor}$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

   (b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2 \rfloor}$, and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n] \setminus \{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.

   (a) It samples $\mathbf{U}_2 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lfloor m/2 \rfloor \times t}$.

   (b) Next, it sets $\mathbf{W} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$, chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{W}' = \mathsf{Arrange}(\mathbf{W}, \mathbf{w}, [n] \setminus \{i\})$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid H_9 : In this hybrid, the matrix $\mathbf{A}_2$ is chosen using $\mathsf{TrapGen}$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.

   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $\mathbf{A}_2 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lfloor m/2 \rfloor}, q\right)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

   (b) It sets $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.

   (a) It samples $\mathbf{U}_2 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lfloor m/2 \rfloor \times t}$.

   (b) Next, it sets $\mathbf{W} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$, chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{W}' = \mathsf{Arrange}(\mathbf{W}, \mathbf{w}, [n] \setminus \{i\})$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid H_{10} : In this hybrid, the challenger chooses $\mathbf{U}_2$ using $\mathsf{SamplePre}$ for each query.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $\mathbf{A}_2 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lfloor m/2 \rfloor}, q\right)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.
   (b) It sets $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.
   (a) It chooses $\mathbf{Y}' \leftarrow \mathbb{Z}_q^{n \times t}$ and samples $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y}')$.
   (b) Next, it sets $\mathbf{W} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$, chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{W}' = \mathsf{Arrange}(\mathbf{W}, \mathbf{w}, [n] \setminus \{i\})$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.
   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.
3. The adversary outputs a bit b'.

Hybrid H_{11} : This hybrid represents a syntactic change in which the i^{th} row of matrix $\mathbf{W}'$ is set as a difference of random vector $\mathbf{c}$ and i^{th} row of $\mathbf{A}_2 \cdot \mathbf{U}_2$ instead of being sampled uniformly at random directly.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $\mathbf{A}_2 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lfloor m/2 \rfloor}, q\right)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.
   (b) It sets $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.
   (a) It chooses $\mathbf{Y}' \leftarrow \mathbb{Z}_q^{n \times t}$ and samples $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y}')$.
   (b) Next, it chooses a uniformly random vector $\mathbf{c} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{C}' = \mathsf{Arrange}(\mathbf{C}, \mathbf{c}, [n] \setminus \{i\})$, sets $\mathbf{W}' = \mathbf{C}' - \mathbf{A}_2 \cdot \mathbf{U}_2$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.
   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.
3. The adversary outputs a bit b'.

Hybrid H_{12} : This hybrid represents a syntactic change. It corresponds to the security game in Definition 7.1 with $b = 1$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $\mathbf{A}_2 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lfloor m/2 \rfloor}, q\right)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.
   (b) It sets $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.
   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$ where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows.

   (a) It chooses $\mathbf{W}' \leftarrow \mathbb{Z}_q^{n\times t}$ and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W})$.

   (b) Next, it chooses a uniformly random vector $\mathbf{c} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{C}' = \mathsf{Arrange}(\mathbf{C}, \mathbf{c}, [n] \setminus \{i\})$, sets $\mathbf{Y}' = \mathbf{C}' - \mathbf{A}_1 \cdot \mathbf{U}_1$ (which is equal to $\mathbf{C}' - \mathbf{W}'$), and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y}')$.

   (c) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

C.2 Detailed Hybrid Experiments for Theorem 7.4

Hybrid H_0 : This corresponds to the single target switching security game.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.

   (a) It chooses $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lceil m/2 \rceil}, q)$ and $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0, 1\}$.

   (b) Next, it sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and sends $[\mathbf{B}_1 \,|\, \mathbf{B}_2]$ to $\mathcal{A}$.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$ where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n\times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0, [n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows.

   (a) It chooses $\mathbf{W} \leftarrow \mathbb{Z}_q^{n\times t}$, computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W})$.

   (b) It also samples vector $\mathbf{e} \leftarrow \chi^t$, and sets $\mathbf{E} = \mathsf{Arrange}(\mathbf{0}^{(n-1)\times t}, \mathbf{e}, [n] \setminus \{i\})$.

   (c) Next, it sets $\mathbf{Y} = \mathbf{Z}_b - \mathbf{A}_1 \cdot \mathbf{U}_1 + \mathbf{E}$ (which is equal to $\mathbf{Z}_b - \mathbf{W} + \mathbf{E}$), and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.

   (d) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.

3. $\mathcal{A}$ outputs its guess b'.

Hybrid H_1 : In this hybrid experiment, the challenger sets $\mathbf{U}_1$ to be a Gaussian matrix for each query.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.

   (a) It chooses $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lceil m/2 \rceil}, q)$ and $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0, 1\}$.

   (b) Next, it sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and sends $[\mathbf{B}_1 \,|\, \mathbf{B}_2]$ to $\mathcal{A}$.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$ where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n\times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0, [n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows.

   (a) It computes $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.

   (b) It also samples vector $\mathbf{e} \leftarrow \chi^t$, and sets $\mathbf{E} = \mathsf{Arrange}(\mathbf{0}^{(n-1)\times t}, \mathbf{e}, [n] \setminus \{i\})$.

   (c) Next, it sets $\mathbf{Y} = \mathbf{Z}_b - \mathbf{A}_1 \cdot \mathbf{U}_1 + \mathbf{E}$, and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.

   (d) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.

3. $\mathcal{A}$ outputs its guess b'.

Hybrid H_2 : In this hybrid experiment, the challenger sets $\mathbf{A}_1$ to be a uniformly random matrix (that is, sampled without a trapdoor).

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times \lceil m/2 \rceil}$ and $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0,1\}$.
   (b) Next, it sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and sends $[\mathbf{B}_1 \mid \mathbf{B}_2]$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$ where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0, [n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows.
   (a) It computes $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.
   (b) It also samples vector $\mathbf{e} \leftarrow \chi^t$, and sets $\mathbf{E} = \mathsf{Arrange}(\mathbf{0}^{(n-1) \times t}, \mathbf{e}, [n] \setminus \{i\})$.
   (c) Next, it sets $\mathbf{Y} = \mathbf{Z}_b - \mathbf{A}_1 \cdot \mathbf{U}_1 + \mathbf{E}$, and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.
   (d) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.
3. $\mathcal{A}$ outputs its guess b'.

Hybrid H_3 : This hybrid is a syntactic change. Here, we express $\mathbf{Y}$ in terms of $\mathbf{B}_1$ and the i^{th} row of $\mathbf{A}_1$. Note that the i^{th} row of $\mathbf{A}_1$ is used only for computing the i^{th} row of $\mathbf{Y}$.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times \lceil m/2 \rceil}$ and $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0,1\}$.
   (b) Next, it sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and sends $[\mathbf{B}_1 \mid \mathbf{B}_2]$ to $\mathcal{A}$.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$ where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0, [n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows.
   (a) It computes $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.
   (b) It also samples vector $\mathbf{e} \leftarrow \chi^t$, and sets $\mathbf{Z}'_b = \mathsf{Restrict}(\mathbf{Z}_b, [n] \setminus \{i\})$.
   (c) Next, it sets $\mathbf{Y}' = \mathbf{Z}'_b - \mathbf{B}_1 \cdot \mathbf{U}_1$, $\mathbf{y} = \mathbf{Z}_b[i] - \mathbf{A}_1[i] \cdot \mathbf{U}_1 + \mathbf{e}$, and $\mathbf{Y} = \mathsf{Arrange}(\mathbf{Y}', \mathbf{y}, [n] \setminus \{i\})$. It then computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.
   (d) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.
3. $\mathcal{A}$ outputs its guess b'.

Hybrid H_4 : In this hybrid experiment, the challenger sets the i^{th} row of $\mathbf{Y}$ to be a uniformly random vector.

1. **Setup Phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows.
   (a) It chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times \lceil m/2 \rceil}$ and $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0,1\}$.
   (b) Next, it sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and sends $[\mathbf{B}_1 \mid \mathbf{B}_2]$ to $\mathcal{A}$.

2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$ where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0, [n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows.

   (a) It computes $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.

   (b) It sets $\mathbf{Z}'_b = \mathsf{Restrict}(\mathbf{Z}_b, [n] \setminus \{i\})$.

   (c) Next, it sets $\mathbf{Y}' = \mathbf{Z}'_b - \mathbf{B}_1 \cdot \mathbf{U}_1$, $\mathbf{y} \leftarrow \mathbb{Z}_q^t$, and $\mathbf{Y} = \mathsf{Arrange}(\mathbf{Y}', \mathbf{y}, [n] \setminus \{i\})$. It then computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.

   (d) Finally, it sends $\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}$ to $\mathcal{A}$.

3. $\mathcal{A}$ outputs its guess b'.