Adaptive Multiparty NIKE

Venkata Koppula
IIT Delhi
kvenkata@cse.iitd.ac.in

Brent Waters
NTT Research & UT Austin
bwaters@cs.utexas.edu

Mark Zhandry
NTT Research & Princeton University
mzhandry@gmail.com

Abstract

We construct adaptively secure multiparty non-interactive key exchange (NIKE) from polynomially hard indistinguishability obfuscation and other standard assumptions. This improves on all prior such protocols, which required sub-exponential hardness. Along the way, we establish several compilers which simplify the task of constructing new multiparty NIKE protocols, and also establish a close connection with a particular type of constrained PRF.

1 Introduction

Non-interactive key exchange (NIKE) is a fundamental application in public key cryptography. In a G-party NIKE protocol, a group of G users simultaneously publish individual public keys to a bulletin board, keeping individual secret keys to themselves. Then just by reading the bulletin board and using their individual private keys but no further interaction, the G users can arrive at a common key hidden to anyone outside the group.

In this work, we build multiparty NIKE attaining *adaptive* security under polynomially-hard non-interactive assumptions. Our assumptions are indistinguishability obfuscation (iO) and standard assumptions on cryptographic groups[1]. The main restriction is that we must bound the number of users that can be adaptively corrupted. That is, the number of honest users, and even the number of adversarially generated users, can be unbounded; only the number of users that were initially honest and later corrupted must be bounded. This improves on prior standard-model adaptively secure schemes [BZ14, Rao14], which all bound the *total* number of users, and also required either *interactive* or *sub-exponential* assumptions. Along the way, we give a toolkit for designing iO-based multiparty NIKE, with several compilers to simplify the design process. We also explore adaptive security for constrained PRFs, giving a new construction for what we call "one symbol fixing" constraints, and show a close connection to multiparty NIKE.

1.1 Prior Work and Motivation

NIKE has a long history, with the 2-party case dating back to the foundational work of Diffie and Hellman [DH76], and the multiparty case already referred to as "a long-standing open problem"

[1]We note that there are two uses of the term "group": the group of users establishing a shared key, and the cryptographic group used as a tool in our constructions. Which use of the term should always be clear from context.

in 2002 [BS02]. Joux gave a 3-party protocol from pairings [Jou00]. The first protocol for $G \geq 4$ used multilinear maps [GGH13a], though the only protocols directly based on multilinear maps that have not been attacked are limited to a *constant* number of users [MZ18]. Currently, the only known solutions for a super-constant number of users are built from indistinguishability obfuscation (iO). The first such construction for polynomially-many users was due to Boneh and Zhandry [BZ14] (using punctured programming techniques [SW14]), with a number of follow-up works [Rao14, KRS15, HJK+16, MZ17, GPSZ17, BGK+18].

Multiparty NIKE remains a fascinating object: the central feature of *non-interactive* key exchange (as opposed to protocols requiring multiple interaction rounds) is that public keys can be re-used across many groups, simplifying key management and significantly reducing communication. This feature makes NIKE an important tool with many applications. Multiparty NIKE in particular is a useful tool for group key management [STW96] and broadcast encryption with small parameters [BZ14]. Multiparty NIKE is also interesting from a foundational perspective, being perhaps the simplest cryptographic object which currently is *only* known via obfuscation[2].

Adaptive Security. The re-use of public keys in a NIKE protocol, on the other hand, opens the door to various *active* attacks. For example, if a shared key for one group is accidentally leaked, it should not compromise the shared key of other groups, including those that may intersect. Worse, an adversary may participate in certain groups using maliciously generated public keys, or may be able to corrupt certain users. Finally, decisions about which groups' shared keys to compromise, how the adversary devises its own malicious public keys, which users to corrupt, and even which set of users to ultimately attack, can all potentially be made *adaptively*.

Adaptive security is an important goal in cryptography generally, being the focus of hundreds if not thousands of papers. Numerous works have considered adaptive NIKE. In the 2-party case, adaptive security can often be obtained *generically* by guessing the group that the adversary will attack. If there are a total of N users in the system, the reduction loss is N^2, a polynomial. The focus of works in the 2-party case (e.g. [CKS08, FHKP13, BJLS16, HHK18, HHKL21]) has therefore been *tight* reductions, which still remains unresolved.

The situation becomes more critical in the multiparty case, where the generic guessing reduction looses a factor of $\binom{N}{G} \approx N^G$, which is exponential for polynomial group size G. In order to make this generic reduction work, one must assume the (sub)exponential hardness of the underlying building blocks and scale up the security parameter appropriately. This therefore results in qualitatively stronger underlying computational assumptions. A couple works have attempted to improve on this reduction, achieving security in the random oracle model [HJK+16], or under *interactive* assumptions [BZ14, Rao14] [3]. In fact, Rao [Rao14] argues that an exponential loss or interactive assumption is likely necessary, by giving a black box impossibility of a polynomial reduction to non-interactive assumptions. This impossibility will be discussed in more depth momentarily. We also note that existing standard-model schemes with adaptive security all limit the *total* number of users, including both honest *and* dishonest users, to an a priori polynomial bound.

Constrained PRFs. A constrained PRF is a pseudorandom function which allows the key holder to produce *constrained* keys k_C corresponding to functions C. The key k_C should allow for evaluating

[2] Multiparty NIKE can also be built via functional encryption [GPSZ17], which is equivalent to iO [BV15a, AJ15] under sub-exponential reductions.

[3] Note that multiparty NIKE security itself can already be phrased as an interactive assumption.

the PRF on any input x where $C(x) = 1$, but the output should remain pseudorandom if $C(x) = 0$. First proposed in three concurrent works [BW13, KPTZ13, BGI14], constrained PRFs have become a fundamental concept in cryptography, with many follow-up works (e.g. [BV15b, BFP+15, CRV16, DKW16, CC17, BTVW17, AMN+18]). A particularly interesting class of constrained PRFs are those for *bit-fixing* constraints, where C takes the form of a vector $v \in \{0, 1, ?\}^n$, and accepts any string x such that $v_i = x_i$ or $v_i =?$ for all i. Bit-fixing constrained PRFs give secret key broadcast encryption [BW13], for example.

Adaptive security is of particular interest for constrained PRFs [Hof14, FKPR14, HKW15, HKKW14, DKN+20]. For example, Boneh and Zhandry [BZ14] build "semi-statically" secure NIKE from adaptively secure constrained PRFs. Unfortunately, with one exception, all known adaptively secure constrained PRFs require random oracles, super-polynomial hardness, or a constant collusion resistance bound. The one exception is [HKW15] for simple puncturing constraints, where C contains a list of polynomially-many points, and accepts all inputs not in the list. Even with such simple constraints, the construction requires iO, algebraic tools, and a non-trivial proof. Fuchsbauer et al. [FKPR14] show that the bit-fixing construction of Boneh and Waters [BW13] *inherently* loses a superpolynomial factor in any reduction to non-interactive assumptions.

1.2 Technical Challenges

Rao's impossibility. Rao [Rao14] proves that multiparty NIKE protocols with standard model proofs relative to non-interactive assumptions (including iO) must incur an exponential loss. The proof follows a meta-reduction, which runs the reduction until the reduction receives the challenge from the underlying non-interactive assumption. At this point, Rao argues that the adversary need not commit to the group it will attack. Now, we split the reduction into two branches:

- In the first branch, choose and corrupt an arbitrary honest user i, obtaining secret key sk_i. Then abort the branch.
- In the second branch, choose the group S to attack such that (1) S contains only honest users for this branch, and (2) $i \in S$. User i is honest in this branch since it was never corrupted here, despite being corrupted in the other branch. Use sk_i to compute the shared group key.

From the view of the reduction, the second branch appears to be a valid adversary. Hence, by the guarantees of the reduction, it must break the underlying hard problem, a contradiction. Hence, no such reduction could exist.

Rao's proof is quite general, and handles reductions that may rewind the adversary or run it many times concurrently. It also works in the more restricted setting where there is an upper bound on the total number of users in the system.

There is *one* way in which Rao's result does not completely rule out a construction relative to polynomial hardness: in order to guarantee that the second branch is successful, one needs that the shared key derived from sk_i must match the shared key in the second branch. This would seem to follow from correctness, as i is a member of the group S. However, correctness only holds with respect to honestly generated public and secret keys. The reduction may, however, give out malformed public or secret keys that are indistinguishable from the honest keys. In this case, it may be that sk_i actually computes the wrong shared key, causing the meta-reduction to fail.

Rao therefore only considers reductions where, roughly, the public keys of the users outputted by the reduction, even if not computed honestly, uniquely determine the shared key. Rao calls these

"admissible reductions." Analogous lower bounds have been shown for tight reductions in the 2-party setting [BJLS16, HHK18, HHKL21], making similar restrictions on the reduction referred to as "committing reductions."

All existing reductions for multiparty NIKE from iO are admissible. A closer look reveals that all such schemes derive the shared key from some constrained PRF applied to the public values of the users. While the secret key is used to compute this value, the value itself does not depend on the secret key, only the public key. Therefore, Rao's impossibility captures all the existing techniques, and new ideas are required to achieve adaptive security from static polynomial assumptions.

Dual system methodology? The situation is reminiscent of HIBE and ABE, where Lewko and Waters [LW14] showed that adaptive security cannot be proved under polynomially hard non-interactive assumptions, using reductions that always output secret keys which decrypt consistently. Solutions overcoming this barrier were already known, say based on dual system encryption [Wat09, LOS+10]. The point of [LW14] was to explain necessary features of those proofs.

The multiparty NIKE setting appears much more challenging. HIBE and ABE benefit from a central authority which issues keys. In the proof, the reduction provides the adversary with all of the keys, which will have a special structure that allows for decrypting some ciphertexts and not others. In the NIKE setting, the adversary is allowed to introduce *his own* users. This presents many challenges as we cannot enforce any dual system structure on such users. It also gives the adversary a lot more power to distinguish the *reduction's* keys from honestly generated keys, as the adversary can request the shared keys of groups containing both honest and malicious users.

Very recently, Hesse et al.[HHKL21] show how to circumvent the above barriers in the 2-party setting. However there is no obvious analog to the multiparty setting.

Another barrier: adaptive constrained PRFs. Looking ahead, we will show that adaptive multiparty NIKE implies adaptive constrained PRFs for a limited "one symbol fixing" functionality. Here, the inputs are words over a polynomial-sized alphabet Σ, and constrains have the form $(?, ?, \cdots, ?, s, ?, \dots)$, constraining only a single position to some character. The resulting PRFs are fully collusion resistant. One-symbol-fixing constrained PRFs can be seen as a special case of bit-fixing PRFs, where only a single contiguous block of bits can be fixed. Adaptive constrained PRFs for even very simple functionalities have remained a very challenging open question. In particular, no prior standard-model construction from polynomial hardness achieves functionalities that have a superpolynomial number of both accepting and rejecting inputs. Any adaptive multiparty NIKE construction would along the way imply such a functionality, representing another barrier.

1.3 Result Summary

- We give several compilers, allowing us to simplify the process of designing multiparty NIKE schemes. One compiler shows how to generically remove a common setup from multiparty NIKE (assuming iO). We note that many iO-based solutions could be tweaked to remove setup, but the solutions were ad hoc and in the adaptive setting often required significant effort; we accomplish this generically.

  Another compiler shows that it suffices to ignore the case where the adversary can compromise the security of shared keys for a different groups of users. That is, we show how to generically

compile any scheme that is secure against adversaries that *cannot* compromise shared keys into one that is secure even if the adversary *can*.

- We show a close connection between multiparty NIKE and one-symbol-fixing PRFs:
  - Adaptively secure multiparty NIKE—even for a bounded number of users, and no maliciously generated users—implies adaptively secure one-symbol-fixing PRF.
  - One-symbol-fixing PRFs, together with iO, imply a multiparty NIKE protocol with a bounded number of honest users (and hence also corruption queries) and group size, but an unbounded number of malicious users. This result starts by constructing a weaker NIKE protocol, and then applying our compilers.
- We construct adaptively secure one-symbol-fixing PRFs from iO and DDH. We thus obtain multiparty NIKE from the same assumptions with a bounded number of honest users.
- We finally give a direct construction of multiparty NIKE from iO and standard assumptions on groups, which allows for an unbounded number of honest users. The construction roughly follows the path above, but opens up the layers of abstraction and makes crucial modifications to attain the stronger security notion. The main limitation is that there is still a bound on the number of users that the adversary can adaptive corrupt, as well as on the group size.

1.4 On Polynomially-Hard iO

Indistinguishability obfuscation can be thought of as an exponential-sized family of assumptions: for every pair of equivalent circuits C_0, C_1, iO assumes that the obfuscations of C_0, C_1 are computationally indistinguishable. This puts iO on a different footing that typical assumptions such as DDH, which are just single fixed assumptions.

iO can be constructed from a constant-sized family of assumptions, as shown by Jain, Lin, and Sahai [JLS21]. However, the underlying assumptions must be sub-exponentially hard, and there is evidence that sub-exponential hardness is necessary when reducing from a fixed number of assumptions [GGSW13]. Under sub-exponential hardness the resulting iO construction achieves sub-exponential hardness as well. In this case, one can achieve adaptive multi-party NIKE under the same sub-exponential assumptions by starting with a suitable selective scheme (say, Boneh-Zhandry [BZ14]), and then applying the generic reduction between selective and adaptive security.

Nevertheless, it is still interesting to consider achieving adaptive security from polynomial iO. Here we list several reasons:

- Achieving security under an exponential number of polynomially hard assumptions (as is the case for polynomially-hard iO) offers a different trade-off than security under a constant number of sub-exponentially hard assumptions. For example, it is conceivable that sub-exponential hardness does not exist in NP, and yet iO (and other crypto) exist. Thus, while sub-exponential hardness might be inherent to proving security relative to a polynomial number of assumptions, sub-exponential hardness is not inherent in iO itself. In such a world, the security proof of iO from sub-exponentially hard assumptions would be vacuous. In contrast, by proving security relative to polynomial hardness, we guarantee security in such a world.
- [JLS21] requires constant-locality PRGs, which have not been well-studied in the sub-exponential hardness regime. We also note that [JLS21] relies on "pre-quantum" assumptions,

and has no security guarantees against quantum attackers. As a result of these considerations, there are numerous constructions (e.g. [GGH+13b, GMM+16, MZ18, BGMZ18, BIJ+20, BDGM20, GP21, WW21]) of iO based on multilinear maps or other objects that could plausibly be secure, even if the assumptions underlying [JLS21] turn out to be insecure or do not achieve the required subexponential hardness.

- If iO ever becomes truly practical, the exponential security loss required to base security on a polynomial of assumptions would likely result in too-high an overhead. As such, we expect any future practical iO construction to not have a meaningful security proof under such assumptions, and instead only have conjectured security. In this setting, polynomially-hard iO is a milder assumption, and a security proof with a polynomial loss would give a much more efficient construction. This is consistent with the current state-of-affairs, where the the JLS construction achieving security under a few assumptions is far more inefficient than "direct" constructions.

- Ideally, future work would remove all bounds, including the bound on the group size. We note that with an unbounded group size, it is no longer impossible to brute force your way to adaptive security, as the security loss $\binom{N}{G} \approx N^G$ is an unbounded exponential. Any such result using polynomial iO would have to, as a special case, also achieve security for a bounded group size. Thus, our work represents a necessary step in this direction. Even in our more limited setting, there are still significant barriers to achieving adaptive security under polynomial hardness, as evidenced by Rao's impossibility.

- Ideally, future work would also base security on a polynomial number of polynomial assumptions. Currently, this only seems possible by basing security on functional encryption (perhaps following [GPS16, GS16, GPSZ17, LZ17]), which is implied by iO under polynomial reductions. Thus, any result along these lines would have to in particular also achieve security under polynomial iO. Again, this means our work represents a necessary step in this direction.

1.5 Technical Overview

We first briefly recall the types of queries an adversary can make:

- **Corrupt User.** The adversary selects an honest user's public key, and learns the secret key.

- **Shared Key.** The adversary selects a list of public keys, which may contain both honest users adversarially-generated users, and learns the shared key for the group of users. Since the adversary's public keys may be malformed, different users may actually arrive at different shared keys. So the query specifies which of the users' version of the shared key is revealed.

- **Challenge.** Here, the adversary selects a list of honest public keys, and tries to distinguish the shared key from random.

Upgrading NIKE. In addition to providing the first iO-based NIKE, Boneh and Zhandry [BZ14] also construct the first NIKE without a trusted setup, or crs. Their basic idea is to first design an iO-based protocol *with* a crs, but where the resulting crs is only needed to generate the shared keys, but not the individual public keys. Then they just have every user generate their own crs; when it

comes time to compute the shared key for a group, the group arbitrarily selects a "leader" and uses the leader's crs.

The above works in the selective setting. However, in the adaptive setting, problems arise. The crs contains an obfuscated program that is run on the user's secret key. The adversary could therefore submit a Shared Key query on an adversarial public key containing a malicious crs. If that malicious user is selected as the leader for the group, honest users' secret keys will be fed into the malicious program, the output being revealed to the adversary, leading to simple attacks. Worse, in Rao's basic scheme with setup, the users need to know the crs in order to generate their public key. So in the setup-less scheme, each user would need to wait until the leader outputs their crs before they can publish their public key, resulting in an interactive protocol. Boneh and Zhandry and later Rao [Rao14] therefore devised more sophistocated techniques to remove the trusted setup.

Our first result sidesteps the above difficulties, by considering the setting where Shared Key queries are not allowed. In this setting, we can make the above strategy of having each party run their own trusted setup fully generic. To accommodate the case where the public keys may depend on the trusted setup, we actually have each user produce an obfuscation of a program that takes as input the crs, and samples a public key. In order to prove security, we also have the secret key for a user be an obfuscated program, which is analogous to the public key program except that is samples the corresponding secret key. In the reduction, this allows us to adaptively embed information in the secret key, which is needed to get the proof to work. See Section 3.2 for details.

Then we show how to generically lift any NIKE scheme that does not support Shared Key queries into one that does support them, without any additional assumptions. Combined with the previous compiler, we therefore eliminate the crs *and* add Shared Key queries to any scheme. The high-level idea is to give the reduction a random subset of the secret keys for honest users. The hope is that these keys will be enough to answer all Shared Key queries, while *not* allowing the reduction to answer the Challenge query. This requires care, as this will not be possible if some of the Shared Key queries have too much overlap with the Challenge query. Our solution is to have have each user actually have many public keys, a subset of which will be used to compute shared keys. By carefully choosing subsets using error correcting codes, we can ensure that Shared Key queries are sufficiently far from the Challenge query, allowing the above idea to work. See Section 3.3 for details.

Connection to Constrained PRFs. Multi-party NIKE already had a clear connection to constrained PRFs, with all iO-based NIKE crucially using constrained PRFs. In Section 4, we make this precise, showing that one symbol fixing (1-SF) PRFs are *equivalent* to NIKE, assuming iO.

One direction is straightforward: to build a 1-SF PRF from multiparty NIKE, create $n \times |\Sigma|$ users, which are arranged in an $|\Sigma| \times n$ grid. Each input in Σ^n then selects a single user from each column, and the value of the PRF is the shared key for the resulting set of n users. To constrain the ith symbol to be σ, simply reveal the secret key for user σ in column i.

The other direction is more complicated, and requires additionally assuming iO. The high-level idea is that the shared key for a group of users will be a PRF evaluated on the list of the users' public keys. If we pretend for the moment that user public keys come from a polynomial-sized set Σ, we could imagine using a 1-SF PRF for this purpose.

Following most iO-based NIKE protocols, we will then have a crs be an obfuscated program which takes as input the list of public keys, together with one of the users secret keys, and evaluates the PRF if the secret key is valid. Our novelty is how we structure the proof to attain adaptive security. Observe that user σ's secret key allows them to evaluate the PRF on any input that

contains at least one σ. This is the union of the inputs that can be computed by keys that constrain symbol i to σ, as i ranges over all input positions.

We therefore switch to a hybrid where user σ has the aforementioned constrained keys covertly embedded in their secret key. In this hybrid, we crucially allow the reduction to generate the user's public key without knowing the constrained keys, and only later when the adversary makes a corruption query will it query for the constrained keys and construct the user's secret key. This strategy is our first step to overcoming Rao's impossibility result: the shared key is no longer information-theoretically determined by the public keys, and is only determined once the secret key with the embedded constrained key is specified. We note, however, that a version of Rao's impossibility still applies to the underlying adaptively secure constrained PRFs, which we will have to overcome later when constructing our PRF.

Moving to this hybrid is accomplished using a simplified version of delayed backdoor programming [HJK+16]. After switching the secret keys for each user, we switch the crs program to use the embedded constrained keys to evaluate the PRF, rather than the master key. At this point, adaptive NIKE security follows directly from adaptive 1-SF PRF security.

Of course, NIKE protocols cannot have public keys in a polynomial-sized set. Our actual protocol first generically compiles a 1-SF PRF into a more sophisticated constrained PRF where now Σ is exponentially large. By adapting the above sketch to this special kind of constrained PRF, we obtain the full proof. See Section 4 for details.

Constructing 1-SF PRFs. We turn to constructing a 1-SF PRF. As mentioned above, a version of Rao's impossibility result still applies even to constrained PRFs. Namely, an "admissible" reduction would commit at the beginning of the experiment to the PRF functionality it provides to the adversary. Such an admissible reduction cannot be used to prove adaptive security for constrained PRFs, for almost identical reasons as with Rao's impossibility. This means our reduction must actually have the PRF seen by the adversary be specified dynamically, where its outputs are actually dependent on prior queries made by the adversary.

One may be tempted to simply obfuscate a puncturable PRF. Boneh and Zhandry [BZ14] show that this gives a constrained PRF for any constraint, though only with selective security. Unfortunately, it appears challenging to to get adaptively secure constrained PRFs with this strategy. In particular, the punctured PRF specifies the value of the PRF at all points but one, which is problematic given that we need to dynamically determine the PRF function in order to circumvent Rao's impossibility.

We will instead use algebraic tools to achieve an adaptively secure construction. Our PRF will be Naor-Reingold [NR97], but adapted from a binary alphabet to a polynomial-sized alphabet. The secret key contains $n \times |\Sigma|$ random values $e_{j,\sigma}$, and the PRF on input $(x_1, \ldots, x_n) \in \Sigma^n$ outputs

$$\mathsf{F}(k, x) = h^{\prod_{i=1}^{n} e_{i,x_i}} \ ,$$

where h is a random generator of a cryptographic group. Without using any computational assumptions, F is already readily seen to be a 1-SF constrained PRF for *a single constrained key*. To constrain position i to σ, simply give out $e_{i,\sigma}$ and $e_{j,x}$ for all $x \in \Sigma$ and all $j \neq i$.

However, we immediately run into trouble even for two constrained keys, since constrained keys for two different i immediately yield the entire secret key. Instead, we constrain keys in this way, except that we embed the constrained keys in an obfuscated program. While this is the natural approach to achieve many-key security, it is a priori unclear how to actually prove security.

We show that obfuscating the constrained keys does in fact upgrade the single-key security of the plain scheme to many-time security. The proof is quite delicate. Essentially, we move to a hybrid where each constrained key uses its own independent h. But here we have a problem: since multiple keys will be able to compute the PRF at the same point, we need to ensure consistency between the keys. Maintaining this consistency is the main challenge in the many-key setting. To maintain such consistency, a constrained key only uses its particular h for inputs that cannot be computed by previous constrained keys. For outputs that can be computed by previous keys, the new constrained key will have to use the h for those keys.

Interestingly, this means that keys in this hybrid must actually contain the h's of all previous constrained keys, and the evaluation of the PRF will actually depend on the order constrained keys are queried. The salient point is that, when the ith constrained key query is made, we only commit to the structure of the PRF on the points that can be evaluated by the first i queries, but the PRF on the remaining part of the domain is unspecified. Structuring the proof in this way is the main insight that allows us to circumvent Rao's impossibility and prove adaptive security.

By careful iO arguments, we show that we are able to move to such a setting where the h for different pieces are random independent bases. The challenge query is guaranteed to be in its own piece, using a different h than all the constrained keys. Therefore, once we move to this setting the constrained keys do not help evaluate the challenge, and security follows. See Section 5 for details. By combining with our compilers, we obtain the following:

Theorem 1 (Informal)**.** *Assuming polynomial iO and DDH, there exist an adaptively secure multiparty NIKE where the number of honestly generated users is a priori bounded, but where the number of maliciously generated users is unbounded.*

In addition to improving to only polynomial hardness, the above improves on existing works by enhancing the security definition to allow an unbounded number of malicious users. Note that the adversary can always create some of its unbounded malicious users in an honest way. In some ways, such users behave as challenger-generated honest users, in that they can be corrupted trivially since the adversary already knows their secret key. But on the other hand, they can *never* be a part of the challenge set of users. Therefore, another way of phrasing security is that the adversary must commit to a bounded-sized set of initially-honest users T, such that the challenge set is a subset of T. Other than the bound on T, the number of honest and malicious users is unbounded. Such a security notion may be useful in settings where the adversary has some idea of which group it wants to attack, but may have some flexibility on exactly which users it includes in the group.

Our Final Construction. Finally, we give another NIKE construction which further improves on the security attained in Theorem 1, at the cost of a slightly stronger group-based assumption:

Theorem 2 (Informal)**.** *Assuming polynomial iO and the DDH-powers assumption, there exist an adaptively secure multiparty NIKE where the group size and number of corruptions is bounded, but otherwise the number of honest and malicious users unbounded.*

We note that bounding the number of corruptions is very natural, and has arisen in many cryptographic settings under the name "bounded collusions." Examples include traitor tracing [CFN94], Broadcast encryption [FN94], identity-based encryption [DKXY02] and its generalizations to functional encryption [GVW12], to name a few. Bounded collusions are often seen as a reasonable relaxation, and in many cases are stepping-stones to achieving full security. We view bounded

collusion security for NIKE similarly, except that in some ways, bounded corruptions for NIKE is even stronger than bounded collusions, in that we allow the NIKE adversary to control an *unbounded* number of users, only limiting the number of users that can be corrupted adaptively.

In our construction, we no longer go through 1-SF-PRFs explicitly, but instead open up the layers of abstraction that gave Theorem 1 and make several crucial modifications to the overall protocol. The main technical challenge is that, in our proof of security for 1-SF-PRFs, we must hard-code all prior queries into each secret key. In the obtained NIKE scheme, this means hard-coding all the keys of users generated by the challenger. But as the number of hard-coded users can never be more than the bit-length of the secret key, this limits the number of honest users.

In our solution, we no longer explicitly hardcode the challenger-generated users, but switch to a hybrid where they are generated with a trigger. Only the obfuscated programs can detect this trigger so that they look like honestly generated users, and it moreover is impossible for the adversary to generate users with the trigger. By a delicate hybrid argument, we are able to mimic the security proof above using these triggers instead of the explicitly hardcoded public keys. See Section 6 for details.

Note that the DDH-powers assumption is a q-type assumption, but this can be proved from a single assumption in the composite order setting, assuming appropriate subgroup decision assumptions [CM14].

2 Preliminaries

2.1 Multiparty NIKE

Here, we define the version of NIKE that we will be considering.

Definition 1 (Multiparty NIKE, Syntax)**.** *A multiparty NIKE scheme with bounded honest users is a pair* $(\mathsf{Pub}, \mathsf{KeyGen})$ *with the following syntax:*

- $\mathsf{Pub}(1^\lambda, 1^\ell, 1^n, 1^c)$ *takes as input the security parameter* λ*, an upper bound* n *on the number of honest users, an upper bound* ℓ *on the number of users in a set, and an upper bound* c *on the number of corruptions. It outputs a public key* pk *and secret key* sk*.*

- $\mathsf{KeyGen}(U, \mathsf{sk})$ *takes as input a list* U *of* $t \leq \ell$ *public keys, plus the secret key for one of the public keys. It outputs a shared key. We have the following correctness guarantee: for any* $\ell, n, c > 0, t \in [\ell]$ *and any* $i, j \in [t]$*,*

$$\Pr[\mathsf{KeyGen}(\{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}, \mathsf{sk}_i) = \mathsf{KeyGen}(\{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}, \mathsf{sk}_j)] \geq 1 - \mathsf{negl}$$

  where the probability is over $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda, 1^\ell, 1^n, 1^c)$ *for* $i = 1, \ldots, t$*.*

Enhanced correctness notions. As a technical part of our compilers, we will also consider stronger variants of correctness. The first is *perfect correctness*, where the probability above is exactly 1. The second notion is *adversarial correctness*, which is defined via the following experiment with an adversary $\mathcal{A}$:

- On input 1^λ, $\mathcal{A}$ computes $1^\ell, 1^n, 1^c$.

- The challenger runs $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{Pub}(1^\lambda, 1^\ell, 1^n, 1^c)$ for $b = 0, 1$, and sends $\mathsf{pk}_0, \mathsf{pk}_1$ to $\mathcal{A}$

- $\mathcal{A}$ then computes a set U of public keys such that $|U| \leq \ell$ and $\mathsf{pk}_0, \mathsf{pk}_1 \in U$.
- The challenger computes $k_b = \mathsf{KeyGen}(U, \mathsf{sk}_b)$ for $b = 0, 1$. $\mathcal{A}$ wins if and only if $k_0 \neq k_1$.

A multi-party NIKE scheme is adversarially correct if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function ϵ such that the $\mathcal{A}$ wins with probability at most ϵ.

Security. We now define security.

Definition 2 (Multiparty NIKE, Adaptive Security)**.** *Consider the following experiment with an adversary $\mathcal{A}$:*

- *The challenger initializes empty tables T and U. T will contain records of the form $(\mathsf{pk}, \mathsf{sk}, b)$ where pk, sk are the public key and secret key for a user, and b is a flag bit indicating if the user is honest (0) or corrupted (1). We will maintain the invariant that if the flag bit is 0, then $\mathsf{sk} \neq \perp$. U will contain unordered sets of public keys. The challenger also keeps track of an unordered set S^*, initially set to $\perp$.*
- *$\mathcal{A}$ receives 1^λ, and replies with $1^\ell, 1^n, 1^c$. It can now make several kinds of queries:*
  - **Register Honest User.** *Here, $\mathcal{A}$ sends nothing. The challenger runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Pub}(1^\lambda, 1^\ell, 1^n, 1^c)$. If there is a record containing pk in T, the challenger replies with $\perp$. Otherwise, it adds $(\mathsf{pk}, \mathsf{sk}, 0)$ to T, and sends pk to $\mathcal{A}$. The total number of such queries is not allowed to exceed n.*
  - **Corrupt User.** *Here, $\mathcal{A}$ sends an pk. The challenger finds a record $(\mathsf{pk}, \mathsf{sk}, 0)$ in the table T. If no such record is found, or if a record is found but with flag bit set to 1, the challenger replies with $\perp$. Otherwise it replies with sk. It then updates the record in T to $(\mathsf{pk}, \mathsf{sk}, 1)$. The total number of such queries is not allowed to exceed c.*
  - **Register Malicious User.** *Here, $\mathcal{A}$ sends a public key pk. If there is no record in T containing pk, the challenger adds to T the record $(\mathsf{pk}, \perp, 1)$. There is no limit to the number of such queries.*
  - **Shared Key.** *Here, the adversary sends an unordered set $S = (\mathsf{pk}_1, \ldots, \mathsf{pk}_t)$ of up to $t \leq \ell$ distinct public keys, as well as an index $i \in [t]$. If $S^* \neq \perp$ and $S = S^*$, then the challenger replies with $\perp$.*

    Otherwise, the challenger checks for each $j \in [t]$ if there is a record $(\mathsf{pk}_j, \mathsf{sk}_j, b_j)$ in T. Moreover, it checks that $\mathsf{sk}_i \neq \perp$. If any of the checks fail, the challenger replies with $\perp$. If all the checks pass, the challenger replies with $\mathsf{KeyGen}(S, \mathsf{sk}_i)$. It adds the list S to U. There is no limit to the number of such queries.
  - **Challenge.** *The adversary makes a single challenge query on an unordered list $S = (\mathsf{pk}_1^*, \ldots, \mathsf{pk}_t^*)$ of up to $t \leq \ell$ distinct public keys. The challenger sets $S^* = S$. The challenger then checks for each $j \in [t]$ that there is a record $(\mathsf{pk}_j^*, \mathsf{sk}_j^*, b_j^*)$ in T such that $b_j^* = 0$. The challenger also checks that S^* is not in U. If any of the checks fails, the challenger immediately aborts and outputs a random bit.*

    If the checks pass, the challenger chooses a random bit $b^* \in \{0, 1\}$ and replies with k_{b^*} where $k_0 \leftarrow \mathsf{KeyGen}(S^*, \mathsf{sk}_1)$ and k_1 is uniformly random.
- *Finally, $\mathcal{A}$ produces a guess b' for b^*. The challenger outputs 1 if $b' = b^*$ and 0 otherwise.*

A Multiparty NIKE is adaptively secure if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function ϵ such that the challenger outputs 1 with probability at most $\frac{1}{2} + \epsilon$.

Other security notions. We can also consider multiparty NIKE with *unbounded honest users*, where the input 1^n is ignored in Pub, and there is no limit to the number of Register Honest User. We can similarly consider multiparty NIKE with *unbounded corruptions* where there is no limit to the number of Corrupt User queries, and *unbounded set size*, where there is no limit to the set size t that can be inputted to KeyGen or queried in Shared Key or Challenger queries.

We can also consider NIKE that is "secure with out X queries", which means that security holds against all adversaries that do not make any queries of type X.

Common Reference String. We can also consider a crs model, where there is a setup algorithm $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^n, 1^c)$. Then Pub is changes to have the syntax $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Pub}(\mathsf{crs})$. In the adaptive security experiment, we have the challenger run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^n, 1^c)$ and give crs to $\mathcal{A}$. It then uses the updated Pub algorithm when registering honest users.

2.2 Constrained PRFs

A special case of bit-fixing PRFs. Here, we define a type of bit-fixing PRF.

Definition 3 (1-Symbol-Fixing PRF, Syntax)**.** *1-SF-PRF is a tuple* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Constr}, \mathsf{EvalC})$ *with the following syntax:*

- $\mathsf{Gen}(1^\lambda, 1^{|\Sigma|}, 1^\ell)$ *takes as input a security parameter λ, an alphabet size $|\Sigma|$, and an input length ℓ, all represented in unary. It outputs a key k.*
- $\mathsf{Eval}(k, x)$ *is the main evaluation algorithm, which is deterministic and takes as input a key k and $x \in \Sigma^\ell$, and outputs a string.*
- $\mathsf{Constr}(k, i, z)$ *is a potentially randomized constraining algorithm that takes as input a key k, an index $i \in [\ell]$, and symbol $z \in \Sigma$. It outputs a constrained key $k_{i,z}$.*
- $\mathsf{EvalC}(k_{i,z}, x)$ *takes as input a constrained key $k_{i,z}$ for an index/symbol pair (i, z), and an input x. It outputs a string. We have the correctness guarantee that:*

$$\mathsf{EvalC}(k_{i,z}, x) = \begin{cases} \bot & \text{if } x_i \neq z \\ \mathsf{Eval}(k, x) & \text{if } x_i = z \end{cases}$$

Definition 4 (1-SF-PRF, Adaptive Security)**.** *Consider the following experiment with an adversary $\mathcal{A}$:*

- *$\mathcal{A}$ on input 1^λ, produces $1^{|\Sigma|}, 1^\ell$. The challenger runs $k \leftarrow \mathsf{Gen}(1^\lambda, 1^{|\Sigma|}, 1^\ell)$. It returns nothing to $\mathcal{A}$.*
- *Then $\mathcal{A}$ can adaptively make the following types of queries:*
  - **Constrain.** *$\mathcal{A}$ sends i, z, and receives $k_{i,z} \leftarrow \mathsf{Constr}(k, i, z)$. The challenger records each (i, z) in a table C. There is no limit to the number of constrain queries.*

– **Eval.** *$\mathcal{A}$ sends an input x, and receives $\mathsf{Eval}(k, x)$. The challenger records each x in a table E. There is no limit to the number of Eval queries.*

– **Challenge.** *$\mathcal{A}$ can make a* single *challenge query on an input $x^* \in \Sigma^\ell$. The challenger flips a random bit $b \in \{0, 1\}$ and replies with $y^* = y_b$ where $y_0 = \mathsf{Eval}(k, x)$ and y_1 is sampled uniformly and independently.*

If at any time, $x_i^* = z$ for some $(i, z) \in C$ or $x^* \in E$, the challenger immediately aborts and outputs a random bit.

- *The adversary outputs bit b'. The challenger outputs 1 if $b = b'$, 0 otherwise.*

A 1-SF-PRF is adaptively secure *if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function ϵ such that the challenger outputs 1 with probability at most $\frac{1}{2} + \epsilon$. It is* adaptively secure without Eval queries *if this holds for all $\mathcal{A}$ that make no Eval queries.*

A 1-SF-PRF scheme is said to be adaptively secure against unique-query adversaries *if the above holds for any adversary $\mathcal{A}$ that makes unique constrained key queries to the challenger.*

2.3 Puncturable Pseudorandom Deterministic Encryption

Below, we present the notion of puncturable pseudorandom deterministic encryption (PPDE) introduced by [KPW17]. In a PPDE scheme, we have a symmetric key deterministic encryption algorithm, and a decryption algorithm. Additionally, the private key can be punctured at any point. Given a key punctured at m, the encryption of m is indistinguishable from a uniformly random string. The following syntax and definitions are taken from [KPW17].

Let $\mathcal{M}$ be the message space. A *pseudorandom puncturable deterministic encryption scheme* (or *PPDE scheme*) for $\mathcal{M}$ and ciphertext space $\mathcal{CT} \subseteq \{0, 1\}^\ell$ (for some polynomial ℓ), is defined to be a collection of four algorithms.

- $\mathsf{PPDE.Setup}(1^\lambda)$ takes the security parameter and generates a key K in keyspace $\mathcal{K}$. This algorithm is randomized.
- $\mathsf{DetEnc}(K, m)$ takes a key $K \in \mathcal{K}$ and message $m \in \mathcal{M}$ and produces a ciphertext $\mathsf{CT} \in \mathcal{CT}$. This algorithm is deterministic.
- $\mathsf{Dec}(K, \mathsf{CT})$ takes a key $K \in \mathcal{K}$ and ciphertext $\mathsf{CT} \in \mathcal{CT}$ and outputs $m \in \mathcal{M} \cup \{\perp\}$. This algorithm is deterministic.
- $\mathsf{Puncture}_{\mathrm{PPDE}}(K, m)$ takes a key $K \in \mathcal{K}$ and message $m \in \mathcal{M}$ and produces a *punctured key* $K\{m\} \in \mathcal{K}$ and $y \in \{0, 1\}^\ell$. This algorithm may be randomized.

Correctness A PPDE scheme is correct if it satisfies the following conditions.

1. **Correct Decryption** For all messages m and keys $K \leftarrow \mathcal{K}$, we require
$$\mathsf{Dec}(K, \mathsf{DetEnc}(K, m)) = m.$$
2. **Correct Decryption Using Punctured Key** For all distinct messages m, for all keys $K \leftarrow \mathcal{K}$,

$$\Pr\left[\begin{array}{c} \#\{\mathsf{CT} : \mathsf{Dec}(K\{m\}, \mathsf{CT}) \neq \mathsf{Dec}(K, \mathsf{CT})\} > 1 \\ (K\{m\}, y) \leftarrow \mathsf{Puncture}_{\mathrm{PPDE}}(K, m) \end{array}\right]$$

is less than $\mathsf{negl}(\lambda)$, where all probabilities are taken over the coins of $\mathsf{Puncture}_{\mathrm{PPDE}}$.

3. For all messages $m^* \in \mathcal{M}$ and keys $K \leftarrow \mathcal{K}$,

$$\left\{y \;\middle|\; (K\{m^*\}, y) \leftarrow \mathsf{Puncture}_{\mathrm{PPDE}}(K, m^*)\right\} \approx U_\ell$$

where U_ℓ denotes the uniform distribution over $\{0,1\}^\ell$.

Definition 5. *A PPDE scheme is* selectively secure *if no PPT algorithm $\mathcal{A}$ can determine the bit b in the following game except with probability negligibly close to $\frac{1}{2}$:*

1. *$\mathcal{A}$ chooses a message m^* to send to the challenger.*
2. *The challenger chooses $K \leftarrow \mathsf{PPDE.Setup}(1^\lambda)$ and $(K\{m^*\}, y) \leftarrow \mathsf{Puncture}_{\mathrm{PPDE}}(K, m^*)$ and $\mathsf{CT} = \mathsf{DetEnc}(K, m^*)$. Next, it chooses $b \leftarrow \{0,1\}$. If $b = 0$, it sends $(K\{m^*\}, (\mathsf{CT}, y))$; otherwise it sends $(K\{m^*\}, (y, \mathsf{CT}))$.*
3. *$\mathcal{A}$ outputs a guess b' for b.*

2.4 DDH-Powers Assumption

Definition 6. *Let $\mathbb{G}$ be a group of order p. The k-DDH-powers assumption states that the following distributions are computationally indistinguishable:*

$$\mathcal{D}_1 \equiv \{(g, g^a, g^{a^2}, g^{a^k}, g^{a^{k+1}}) : g \leftarrow \mathbb{G}, a \leftarrow \mathbb{Z}_p\} \quad \mathcal{D}_2 \equiv \{(g, g^a, g^{a^2}, g^{a^k}, T) : g \leftarrow \mathbb{G}, a \leftarrow \mathbb{Z}_p, T \leftarrow \mathbb{G}\}$$

3 Enhancing Multi-party NIKE

In this section, we give some compilers for multi-party NIKE, which allow for simplifying the task of designing new NIKE protocols built from iO. Our ultimate goal is to show that one can safely ignore Shared Key and Register Malicious User queries, and also employ a trusted setup. Our compilers then show how to lift such a scheme into one secure under all types of queries and without a trusted setup.

3.1 Achieving Adversarial Correctness

First, we show how to convert any NIKE that is perfectly correct into one with adversarial correctness. While adversarial correctness is not a particular design goal in multiparty NIKE, this step will be needed in order to apply our later compilers.

Theorem 3. *Assume there exists a multi-party NIKE with perfect correctness, potentially in the crs model. Assume additionally there exists a NIZK. Then there exists a multi-party NIKE with both perfect and adversarial correctness in the crs model. If the perfectly correct scheme has unbounded honest users, corruptions, and/or set size, then so does the resulting adversarially correctscheme.*

Theorem 3 follows from a standard application of NIZKs, and is similar to a theorem used in the context of two-party NIKE by [HHK18]. The proof is given in Appendix B.

3.2 Removing the CRS

Next, we use iO to remove the common reference string (crs) from any multi-party NIKE. A side-effect of this transformation, however, is that we only achieve security without Register Malicious User queries.

Theorem 4. *Assuming there exists iO an adaptively secure multi-party NIKE in the common reference string (crs) model, then there also exists adaptively multi-party NIKE in the plain model that is secure without Register Malicious User queries. If the crs scheme has unbounded honest users, corruptions, and/or set size, or has perfect and/or adversarial correctness, or only has secure without X queries for some X, then the same is true of the resulting plain model scheme.*

Theorem 4 formalizes the ad hoc techniques for removing the CRS in iO-based constructions starting from Boneh and Zhandry [BZ14]. The technique works by having each user separately run the trusted setup. Then each group selects (deterministically) a distinguished user, whose trusted setup is used to actually derive the shared key. The main limitation of this technique is that it does not preserve security under Register Malicious User queries, which will be fixed in the compiler in the next subsection. We now give the proof of Theorem 4.

Proof. The proofs of the various bounded/unbounded cases and perfect/adversarial correctness cases are essentially the same, so we focus on the case where everything is bounded. We will let $(\mathsf{Setup}, \mathsf{Pub}', \mathsf{KeyGen}')$ be a multi-party NIKE with setup. The idea is to replace a user's public key (which would have potentially needed the crs) with an obfuscated program that takes as input the crs and samples a public/secret key pair from Pub', outputting the public key. The user's secret key is the same program, except that it outputs the sampled secret key. We now give the construction and proof in more detail.

Let F be a puncturable PRF. F can be constructed from any one-way function, which are in turn implied by any NIKE scheme. We construct a new mutliparty NIKE $(\mathsf{Pub}, \mathsf{KeyGen})$ without setup as follows:

- $\mathsf{Pub}(1^\lambda, 1^\ell, 1^n, 1^c)$: Sample a random PRF key k for F. Also run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^n, 1^c)$. Let $\mathsf{PKey}_k, \mathsf{SKey}_k$ be the programs in Figures 1 and 2, and let $\widehat{\mathsf{PKey}} = \mathsf{iO}(\mathsf{PKey}_k), \widehat{\mathsf{SKey}} = \mathsf{iO}(\mathsf{SKey}_k)$. $\mathsf{pk} = (\mathsf{crs}, \widehat{\mathsf{PKey}})$ and $\mathsf{sk} = \widehat{\mathsf{SKey}}$.

- $\mathsf{KeyGen}(S, \mathsf{sk}_i)$: Let $\mathsf{pk}^* \in S$ be the minimal $\mathsf{pk} \in S$ according to some ordering; we will call pk^* the distinguished public key.

  Write $\mathsf{pk}^* = (\mathsf{crs}^*, \widehat{\mathsf{PKey}}^*)$. Let S' be derived from S, where for each $\mathsf{pk} = (\mathsf{crs}, \widehat{\mathsf{PKey}}) \in S$, we include $\mathsf{pk}' = \widehat{\mathsf{PKey}}(\mathsf{crs}^*)$ in S'. Also let $\mathsf{sk}_i = \widehat{\mathsf{SKey}}_i$, and run $\mathsf{sk}'_i = \widehat{\mathsf{SKey}}_i(\mathsf{crs}^*)$. Then run and output $\mathsf{KeyGen}'(\mathsf{crs}, S', \mathsf{sk}'_i)$.

Correctness: Correctness follows from the correctness of the underlying scheme:

$$\begin{aligned}\mathsf{KeyGen}(S, \mathsf{sk}_i) &= \mathsf{KeyGen}'(\mathsf{crs}, S', \mathsf{sk}'_i)\\ &= \mathsf{KeyGen}'(\mathsf{crs}, S', \mathsf{sk}'_j)\\ &= \mathsf{KeyGen}(S, \mathsf{sk}_j)\end{aligned}$$

Figure 1: The program PKey_k.

Inputs: crs
Constants: k

1. $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Pub}'(\mathsf{crs};\ \mathsf{F}(k, \mathsf{crs}))$
2. Output pk'

Figure 2: The program SKey_k.

Inputs: crs
Constants: k

1. $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Pub}'(\mathsf{crs};\ \mathsf{F}(k, \mathsf{crs}))$
2. Output sk'

Security: We now prove the security of the above construction. Let $\mathcal{A}$ be an adversary that wins with probability $\frac{1}{2} + \epsilon_0$, where we assume towards contradiction that ϵ_0 is non-negligible. Let ℓ be an upper bound on the number of Register Honest User queries made by $\mathcal{A}$. Consider the following experiments:

- H_0: This is the standard adaptive NIKE experiment. By assumption, we have that the adversary wins in H_0 with probability $p_0 = \frac{1}{2} + \epsilon_0$.
- H_1: This is the same as H_0, except for the following changes. At the beginning of the experiment, a random $j^* \in [\ell]$ is chosen. Then if the distinguished public key for the Challenge query is *different* from the public key from the j^*-th Register Honest User query, immediately abort and output a random bit. We will call this an "H_1 abort." Let $p_1 = \frac{1}{2} + \epsilon_1$ be the probability $\mathcal{A}$ wins in experiment H_1.

  Let pk_{j^*} be the public key for the j^*-th honest user; note that we can sample pk_{j^*} at the beginning of the experiment. Also note that if no abort happens, it must be that pk_{j^*} is never corrupted.
- H_2: This is the same as H_1, except that we change how Register Honest User queries are answered. For each such query, we sample $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{Pub}'(\mathsf{crs}^*)$. We then run $\mathsf{Pub}(1^\lambda)$, except that we set $\widehat{\mathsf{PKey}} = \mathsf{iO}(\mathsf{PKey}'_{k_{\mathsf{crs}^*}, \mathsf{crs}^*, \mathsf{pk}''})$ and $\widehat{\mathsf{SKey}} = \mathsf{iO}(\mathsf{SKey}'_{k_{\mathsf{crs}^*}, \mathsf{crs}^*, \mathsf{sk}''})$ where $\mathsf{crs}^* = \mathsf{crs}_{j^*}$, k_{crs^*} is the result of puncturing k at crs^*, and $\mathsf{PKey}'_{k_{\mathsf{crs}^*}, \mathsf{crs}^*, \mathsf{pk}''}, \mathsf{SKey}'_{k_{\mathsf{crs}^*}, \mathsf{crs}^*, \mathsf{sk}''}$ are the programs given in Figures 3 and 4. The differences between $\mathsf{PKey}'_{k_{\mathsf{crs}^*}, \mathsf{crs}^*, \mathsf{pk}''}, \mathsf{SKey}'_{k_{\mathsf{crs}^*}, \mathsf{crs}^*, \mathsf{sk}''}$ and $\mathsf{PKey}_k, \mathsf{SKey}_k$ are highlighted in yellow.

  Let $p_2 = \frac{1}{2} + \epsilon_2$ be the probability $\mathcal{A}$ wins in H_2.

Lemma 1. $\epsilon_1 = \epsilon_0 / \ell$.

Lemma 2. *Under the assumed security of* F *and* iO*, there exists a negligible function* negl *such that* $|\epsilon_2 - \epsilon_1| < \mathsf{negl}(\lambda)$.

Lemma 3. *Under the assumed security of* $(\mathsf{Setup}, \mathsf{Pub}', \mathsf{KeyGen}')$*, there exists a negligible function* negl *such that* $\epsilon_2 < \mathsf{negl}(\lambda)$.

Combining the above four lemmas shows that ϵ_0 is negligible, a contradiction. It therefore remains to justify the four lemmas.

Figure 3: The program $\mathsf{PKey}'_{k_{\mathsf{crs}^*},\mathsf{crs}^*,\mathsf{pk}''}$.

Inputs: crs, x

Constants: $k_{\mathsf{crs}^*}, \mathsf{crs}^*, \mathsf{pk}''$

1. $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Pub}'(\mathsf{crs};\ \mathsf{F}(k_{\mathsf{crs}^*}, \mathsf{crs}))$
2. If $\mathsf{crs}^* = \mathsf{crs}$, output pk''. Otherwise output pk'

Figure 4: The program $\mathsf{SKey}'_{k_{z^*},z^*,\mathsf{sk}''}$.

Inputs: crs, x

Constants: $k_{\mathsf{crs}^*}, \mathsf{crs}^*, \mathsf{pk}''$

1. $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Pub}'(\mathsf{crs};\ \mathsf{F}(k_{\mathsf{crs}^*}, \mathsf{crs}))$
2. If $\mathsf{crs}^* = \mathsf{crs}$, output sk''. Otherwise output sk'

Proof of Lemma 1. The only difference between H_0 and H_1 is the H_1 abort. The distinguished public key for the Challenge query must be one of the users registered as honest. Until the H_1 abort, j^* is independent of the adversary's view, and j^* has a probability of $1/\ell$ of being the correct guess. Therefore, from the adversary's view, H_1 is identical to H_0 except for a $1 - 1/\ell$ probability of abort. Hence, $\epsilon_1 = \epsilon_0/\ell$.

Proof of Lemma 2. By a simple hybrid over Register Honest User queries, it suffices to show that $(\mathsf{crs}^*, \mathsf{iO}(\mathsf{PKey}_k), \mathsf{iO}(\mathsf{SKey}_k))$ is indistinguishable from $(\mathsf{crs}^*, \mathsf{iO}(\mathsf{PKey}'_{k_{z^*},z^*,\mathsf{pk}''}), \mathsf{iO}(\mathsf{SKey}'_{k_{z^*},z^*,\mathsf{sk}''}))$, where

- $\mathsf{crs}^* \leftarrow \mathsf{Setup}'(1^\lambda, 1^\ell, 1^n, 1^c)$.
- k is a random key for F.
- k_{crs^*} is the result of puncturing k at crs^*.
- $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{Pub}'(\mathsf{crs}^*)$.

This can be proven using the following sub-hybrids:

- H_a: Output $(\mathsf{crs}^*, \mathsf{iO}(\mathsf{PKey}_k), \mathsf{iO}(\mathsf{SKey}_k))$.
- H_b: Output $(\mathsf{crs}^*, \mathsf{iO}(\mathsf{PKey}'_{k_{\mathsf{crs}^*},\mathsf{crs}^*,\mathsf{pk}'}), \mathsf{iO}(\mathsf{SKey}'_{k_{\mathsf{crs}^*},\mathsf{crs}^*,\mathsf{sk}'}))$ where $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Pub}'(\mathsf{crs}^*; \mathsf{F}(k, \mathsf{crs}^*))$.
- H_c: Output $(\mathsf{pk}^*, \mathsf{iO}(\mathsf{PKey}'_{k_{\mathsf{crs}^*},\mathsf{crs}^*,\mathsf{pk}''}), \mathsf{iO}(\mathsf{SKey}'_{k_{\mathsf{crs}^*},\mathsf{crs}^*,\mathsf{sk}''}))$

The programs PKey_k and $\mathsf{PKey}'_{k_{\mathsf{crs}^*},\mathsf{crs}^*,\mathsf{pk}'}$ are identical, as are SKey_k and $\mathsf{SKey}'_{k_{\mathsf{crs}^*},\mathsf{crs}^*,\mathsf{sk}'}$. Hence H_a and H_b are indistinguishable by iO. Meanwhile, the only difference between $(\mathsf{pk}', \mathsf{sk}')$ and $(\mathsf{pk}'', \mathsf{sk}'')$ is that the random coins used to generate them switch from being $\mathsf{F}(k, \mathsf{crs}^*)$ to uniform. But the rest of the experiment can be simulated using the punctured key k_{crs^*}. Hence indistinguishability between H_b and H_c follow from punctured PRF security.

Proof of Lemma 3. We now describe an adversary $\mathcal{A}'$ for $(\mathsf{Setup}, \mathsf{Pub}', \mathsf{KeyGen}')$, which will run $\mathcal{A}$ as a sub-routine. $\mathcal{A}$ works as follows:

- Let crs^* be the common reference string provided to $\mathcal{A}'$ by its challenger..
- $\mathcal{A}'$ chooses a random j^*. $\mathcal{A}'$ then simulates $\mathcal{A}$, playing the role of challenger to $\mathcal{A}$, answering queries as follows:
  - **Register Honest User.**: $\mathcal{A}'$ makes a Register Honest User query to its own challenger, receiving pk''. It samples a random PRF key k for F, and punctures k at crs^* to obtain k_{crs^*}. It computes $\widehat{\mathsf{PKey}} = \mathsf{iO}(\mathsf{PKey}'_{k_{\mathsf{crs}^*},\mathsf{crs}^*,\mathsf{pk}''})$.

    If this is the j^*-th Register Honest User query, then $\mathcal{A}'$ sets $\mathsf{crs} = \mathsf{crs}^*$. Otherwise, it samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^n, 1^c)$.

    Finally, it returns $(\mathsf{crs}, \widehat{\mathsf{PKey}})$ to $\mathcal{A}$.
  - **Corrupt User.** Upon receiving a public key $\mathsf{pk} = (\mathsf{crs}, \widehat{\mathsf{PKey}})$ from $\mathcal{A}$, $\mathcal{A}'$ checks to see if pk was indeed the result of a previous Register Honest User query that has not previously been corrupted. If the check fails, $\mathcal{A}'$ replies with $\perp$.

    If the check passes, $\mathcal{A}'$ then procures the pk'' it received from its challenge when answering the prior Register Honest User query. It makes a Corrupt User query to its own challenger on pk'', receiving sk''. Then it replies with $\widehat{\mathsf{SKey}} = \mathsf{iO}(\mathsf{SKey}'_{k_{\mathsf{crs}^*},\mathsf{crs}^*,\mathsf{sk}''})$.
  - **Shared Key.** Upon receiving a list of public keys S and index i from $\mathcal{A}$, $\mathcal{A}'$ constructs a list S', where for each $\mathsf{pk} = (\mathsf{crs}, \widehat{\mathsf{PKey}}) \in S$, $\mathcal{A}'$ adds $\widehat{\mathsf{PKey}}(\mathsf{crs}^*)$ to S'. Then $\mathcal{A}'$ makes a Shared Key query on S', i to its own challenger, receiving k, which it forwards to $\mathcal{A}$.
  - **Challenge.** Upon receiving a list of public keys S, $\mathcal{A}'$ constructs S' as in a Shared Key query. Then it makes a Challenge query on S' to its own challenger, receiving k, which it forwards to $\mathcal{A}$.

It is not hard to see that $\mathcal{A}'$ perfectly simulates the view of $\mathcal{A}$ in H_2, and therefore has success probability exactly p_2. By the assumed security of $(\mathsf{Setup}, \mathsf{Pub}', \mathsf{KeyGen}')$, p_2 must be negligible. This completes the proof of Lemma 3, and therefore the proof of Theorem 4. □

3.3 Adding Shared Key Queries

The final compiler generically convert a NIKE scheme whose security does *not* support shared key queries into one that does.

Theorem 5. *Assume there exists a multi-party NIKE with adversarial correctness and adaptive security without Shared Key or Register Malicious User queries. Then there exists a multi-party NIKE with adversarial correctness and adaptive security (*with *Shared Key and Register Malicious*

User queries). If the original scheme is also perfectly correct, then so is the resulting scheme. If the original scheme has unbounded honest users, corruptions, and/or set size, then so does the resulting scheme. The resulting scheme is in the CRS model if and only if the original scheme is.

Note the requirement that the underlying NIKE protocol have adversarial correctness. The proof of Theorem 5 exploits the structure of multiparty NIKE, together with combinatorial tricks, to ensure that the reduction can answer all Shared Key queries (even on sets involving malicious users) while not being able to answer the challenge query. We now give the proof of Theorem 5.

Proof. The rough idea is to randomly give the reduction some of the secret keys for users. We give the reduction enough secret keys so that with non-negligible probability it will be able to answer all shared key queries, while simultaneously being *unable* to answer the challenge query.

There are several difficulties getting this to work. First, when the adversary makes an shared key query on a set of users that includes a maliciously-generated user, correctness is no longer guaranteed. This means different users may output different shared keys, even for the same set. This in turn means the extra secret keys we gave the reduction may not answer correctly. We fix this by assuming the underlying NIKE is adversarially correct.

A more important difficulty is that is that the shared key queries can be highly correlated with the challenge query, potentially differing on only a single user. In order to be able to answer the shared key query but not the challenge query, we must give out the secret key for exactly the differing user, which we do not know in advance. We could of course guess the differing user and incur a polynomial loss in the security reduction, but this will compound over all the shared key queries, resulting in an exponential loss.

Our solution leverages the functionality of multi-party NIKE. Every user will actually contain many sub-users. When computing the shared key for a group of users, a collection of each user's sub-users will be selected, and the key will be computed for the union of all sub-users. The collections of sub-users will be chosen so that each group will yield collections that are "far" from each other. This gives us many possibilities for giving out secret keys without compromising the challenge.

We now give the construction and proof of security in more detail. All the proofs are essentially identical, so we prove the bounded case without a crs. Let $(\mathsf{Pub}', \mathsf{KeyGen}')$ be a multiparty NIKE scheme. Let H be a collision resistant hash function, with hashing key space $\mathcal{H}$ and range $\mathcal{Y}$. Let $C \subseteq \Sigma^m$ be a code of minimum distance d, such that Σ is a polynomial-sized alphabet, and $|C| \geq |\mathcal{H}| \times |\mathcal{Y}|$. Let $\mathsf{Map} : \mathcal{H} \times \mathcal{Y} \to C$ be an arbitrary efficiently computable injective mapping.

Our new multi-party NIKE scheme $(\mathsf{Pub}, \mathsf{KeyGen})$ is given as follows:

- $\mathsf{Pub}(1^\lambda, 1^\ell, 1^n, 1^c)$: Sample a random hashing key $\mathsf{hk} \in \mathcal{H}$. Let $n' = n \times m, \ell' = \ell \times |\Sigma| \times m, c' = c \times \Sigma$. For each $i \in [m], z \in \Sigma$, run $(\mathsf{pk}_{i,z}, \mathsf{sk}_{i,z}) \leftarrow \mathsf{Pub}'(1^\lambda, 1^{\ell'}, 1^{n'}, 1^{c'})$. Output $\mathsf{pk} = (\mathsf{hk}, (\mathsf{pk}_{i,z})_{i,z}), \mathsf{sk} = (\mathsf{sk}_{1,z})_{z \in \Sigma}$.
- $\mathsf{KeyGen}(U, \mathsf{sk})$: Write $U = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}$ and write $\mathsf{pk}_j = (\mathsf{hk}_j, (\mathsf{pk}_{j,i,z})_{i,z})$. Let j^* be such that pk_{j^*} is the minimal element in U for some ordering of public keys. Let $y \leftarrow H(\mathsf{hk}_{j^*}, U)$ and $c = \mathsf{Map}(\mathsf{hk}_{j^*}, y)$. Let $U_j = (\mathsf{pk}_{j,i,c_i})_{i \in [m]}$. Output $\mathsf{KeyGen}'(\cup_{j \in [t]} U_j, \mathsf{sk}_{c_1})$.

Correctness: Correctness follows straightforwardly from the correctness of $(\mathsf{Pub}', \mathsf{KeyGen}')$:

$$\begin{aligned}\mathsf{KeyGen}(U, \mathsf{sk}_{j_1}) &= \mathsf{KeyGen}'(\cup_{j\in[t]} U_j, \mathsf{sk}_{j_1,c_1}) \\ &= \mathsf{KeyGen}'(\cup_{j\in[t],i\in[m]} \mathsf{pk}_{j,i,c_i}, \mathsf{sk}_{j_1,1,c_1}) \\ &= \mathsf{KeyGen}'(\cup_{j\in[t],i\in[m]} \mathsf{pk}_{j,i,c_i}, \mathsf{sk}_{j_2,1,c_1}) \\ &= \mathsf{KeyGen}(U, \mathsf{sk}_{j_2})\end{aligned}$$

Perfect and adversarial correctness follow from similar arguments.

Security: We now prove security. Let $\mathcal{A}$ be an adversary that wins with probability $\frac{1}{2} + \epsilon_0$, where we assume towards contradiction that ϵ_0 is non-negligible. Let q be a polynomial upper bound on the number of Shared Key queries made by $\mathcal{A}$. We prove security through a sequence of hybrids.

- H_0: This is the standard adaptive NIKE experiment. By assumption, we have that the adversary wins in H_0 with probability $p_0 = \frac{1}{2} + \epsilon_0$.
- H_1: Here, we add two abort conditions:
  - **Inconsistency** For every Shared Key query on set $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_t)$, we let $U_j = (\mathsf{pk}_{j,i,c_i})_{i\in[m]}$ be as in KeyGen. Let sk_{j,i,c_i} be the associated secret keys amongst the pk_{j,i,c_i} belonging to honest users.

    The challenger checks that $\mathsf{KeyGen}'(\cup_j U_j, \mathsf{sk}_{j_1,i_1,c_{i_1}}) = \mathsf{KeyGen}'(\cup_j U_j, \mathsf{sk}_{j_2,i_2,c_{i_2}})$ for each $(j_1, i_1), (j_2, i_2) \in [t] \times [m]$ beloning to honest users. If any of the checks fail, then the challenger outputs a random bit and aborts.
  - **Collisions** For the challenge query S^*, let hk^* be the hashing key selected during KeyGen. For any Shared Key query on a set S (occurring before or after the challenge query), the challenger checks that $H(\mathsf{hk}^*, S) \neq H(\mathsf{hk}^*, S^*)$.

  If any of the checks fail, the challenger aborts an outputs a random bit.

  Let $\frac{1}{2} + \epsilon_1$ be the probability the adversary wins.
- H_2: Here, we add a new abort condition. Let $u \in \mathbb{Z}, r \in [0, 1]$ be parameters to be chosen later. At the beginning of the experiment, do the following, for $k = 1, \ldots, u$:
  - Choose a random $i_k \in [m]$.
  - Select a random subset $S_k \subset \Sigma$ of size $r|\Sigma|$. Here, we assume $r|\Sigma|$ is an integer.

  Now, during any Shared Key query on a set S, we compute c as in KeyGen. Then we find a k such that $c_{i_k} \in S_k$, if it exists. If such a k does *not* exist, then we immediately abort and output a random bit.

  On the other hand, during the challenge query on a set S^*, we compute c^* as in KeyGen. Then we find a k such that $c^*_{i_k} \in S_k$, if it exists. If such a k *does* exist, then we immediately abort and output a random bit.

  If an abort happens as above, we call it a **Simulation** abort.

  Finally, at the very end of the experiment, we do the following. Let $L = c_1, \ldots, c_q$ be the list of c's from the various Shared Key queries, and c^* the c from the Challenge query. Note that, by

the **Collision** abort condition, $c^* \notin L$. The probability of a **Simulation** abort, over the choice of i_k, S_k, only depends on L, c^*, u, r. We denote this probability by $p_{L,c^*}(u,r)$. Let $p_{\min}(u,r)$ be a lower bound on $p_{L,c^*}(u,r)$ over the choice of L, c^*. We will discuss the computation of $p_{L,c^*}, p_{\min}$ later. At the end of the experiment, assuming no abort has happened yet, we **Artificially abort** with probability $p_{\min}(u,r)/p_{L,c^*}(u,r)$, which is in $[0,1]$ since $p_{\min}$ lower bounds p_{L,c^*}.

Let $1/2 + \epsilon_2$ be the probability the adversary wins.

- H_3: This is identical to H_2, except that we answer Shared Key queries on a set S and index $j \in [t]$ using $\mathsf{sk}_{j,i_k,c_{i_k}}$. Let $1/2 + \epsilon_3$ be the probability the adversary wins.
- H_4: This is identical to H_3, except that we remove the **Inconsistency** abort condition. Let $1/2 + \epsilon_4$ be the probability the adversary wins.

We now have the following lemmas:

Lemma 4. *Under the assumed adversarial correctness of* $(\mathsf{Pub}', \mathsf{KeyGen}')$ *and collision resistance of* H*, there exists a negligible* negl *such that* $|\epsilon_0 - \epsilon_1| < \mathsf{negl}(\lambda)$.

Lemma 4 is straightforward: any collision or inconsistency immediately yields an attack on the collision resistance of H or the adversarial correctness of $(\mathsf{Pub}', \mathsf{KeyGen}')$.

Lemma 5. $\epsilon_2 = p_{\min}\epsilon_1$.

Lemma 5 is also straightforward: the overall abort probability introduced in H_2 is just $p_{\min}$, independent of the adversary's view.

Lemma 6. $\epsilon_3 = \epsilon_2$

Lemma 6 is also straightforward: by the **Inconsistency** abort condition, any of the sk's will give the same result, so switching to a different sk is identical from the adverary's view.

Lemma 7. *Under the assumed collision resistance of* H*, there exists a negligible* negl *such that* $|\epsilon_4 - \epsilon_3| < \mathsf{negl}(\lambda)$.

Lemma 7 is essentially identical to Lemma 4. Finally, we have:

Lemma 8. *Under the assumed adaptive security of* $(\mathsf{Pub}', \mathsf{KeyGen}')$ *without Shared Key queries,* ϵ_4 *is negligible.*

To prove Lemma 8, we devise an adversary $\mathcal{A}'$ against the adaptive security of $(\mathsf{Pub}', \mathsf{KeyGen}')$ without Shared Key queries, which success probability $1/2 + \epsilon_4$. $\mathcal{A}'$ first chooses random $i_k \in [m]$ for $k = 1, \ldots, u$, and then selects random subsets $S_k \subset \Sigma$ of size $r|\Sigma|$. Then it simulates $\mathcal{A}$, and does the following:

- On a Register Honest User query, $\mathcal{A}'$, for each $i \in [m]$, it lets $T_i = \cup_{k:i_k=i} S_k$, the set of symbols that are included in any of the S_k corresponding to that i. For each i and each symbol $z \in T_i$, it runs $(\mathsf{pk}_{i,z}, \mathsf{sk}_{i,z}) \leftarrow \mathsf{Pub}'(1^\lambda, 1^{\ell'}, 1^{n'}, 1^{c'})$. For all other (i,z) pairs, it makes a Register Honest User query to its own challenger, receiving $\mathsf{pk}_{i,z}$. It then samples a random hashing key $\mathsf{hk} \leftarrow \mathcal{H}$, and outputs $\mathsf{pk} = (\mathsf{hk}, (\mathsf{pk}_{i,z})_{i,z})$.
- On a Corrupt User query on $\mathsf{pk} = (\mathsf{hk}, (\mathsf{pk}_{i,z})_{i,z})$, $\mathcal{A}'$ makes Corrupt User queries for each $z \notin T_1$, receiving $\mathsf{sk}_{1,z}$. For all other z, $\mathcal{A}'$ already has $\mathsf{sk}_{1,z}$. $\mathcal{A}'$ outputs $(\mathsf{sk}_{1,z})_{z \in \Sigma}$

- On any Register Malicious User query, $\mathcal{A}'$ simply registers all the component public keys, assuming they have not been registered before.
- On a Shared Key query on $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_t)$, $\mathcal{A}'$ computes y, c, U_j as in KeyGen. It then outputs $\mathsf{KeyGen}'(\cup_{j\in[t]} U_j, \mathsf{sk}_{j,i_k,c_{i_k}})$, assuming it has the value $\mathsf{sk}_{j,i_k,c_{i_k}}$. Otherwise this corresponds to a **Simulation** abort, in which case $\mathcal{A}'$ outputs a random bit and aborts.
- On the Challenge Query on a set $U = (\mathsf{pk}_1^*, \ldots, \mathsf{pk}_t^*)$, $\mathcal{A}'$ computes y^*, c^*, U_j^* as in KeyGen. If $\cup_{j\in[t]} U_j$ is not entirely comprised of un-corrupted Honest User queries made by $\mathcal{A}'$, then this corresponds to a **Simulation** abort, and $\mathcal{A}'$ immediately aborts and outputs a random bit. Otherwise, it makes a challenge query on $\cup_{j\in[t]} U_j$, forwarding the result to $\mathcal{A}$.
- At the end of the experiment, $\mathcal{A}'$ checks for a **Collision** abort, and also simulates an **Artificial** abort with the necessary probability. If no abort happens, $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs.

From inspection, it can seen that $\mathcal{A}'$ perfectly simulates the view of $\mathcal{A}$ in H_4, and thus has the desired success probability $1/2 + \epsilon_4$. Thus ϵ_4 must be negligible.

In order to now conclude that ϵ_0 is negligible, we must show that there are u, r such that $p_{\min}$ is inverse-polynomial.

Lemma 9. *Assume $(1 - rd/2m)^u q \leq 1$. For any L of size at most q and any c^*, $p_{L,c^*}(u, r) \geq [(1 - r)rd/2m]^u$.*

We can therefore set $p_{\min} = [(1 - r)rd/2m]^u$.Fix a constant $r \in [0, 1]$ and constant rate d/m. Then for a polynomial q, we can set $u = \lceil -\log(q)/\log(1 - rd/2m) \rceil$ to satisfy the conditions of Lemma 9. In this case, we have $p_{\min} = q^{\Omega(1)}$ as desired. We now prove Lemma 9.

Proof. We need to bound the probability both of the following conditions are met:

1. For every k, $c^*_{i_k} \notin S_j$, and
2. For each $c \in L$, there exists a k such that $c_{i_k} \in S_k$.

For each k, the probability that $c^*_{i_k} \notin S_k$ is exactly $(1 - r)$. Over all k, the probability that Condition 1 holds is $(1 - r)^u$.

We will now condition on Condition 1 holding. In this case, each S_k is a random subset of $\Sigma \setminus c^*_{i_k}$. For each i and for each $c \in L$, there are two cases:

- $c_i = c^*_i$. In this case, if $i_k = i$, then $c_{i_k} \notin S_k$ with probability 1.
- $c_i \neq c^*_i$. In this case, if $i_k = i$, then $c_{i_k} \in S_k$ with probability $r|\Sigma|/(|\Sigma| - 1) > r$.

Since the code C has minimum distance d, for each $c \in L$, there are at least d different i such that $c_i \neq c^*_i$. For these i, we have $\Pr[c_i \in S_k] > r$ from above. Thus, for each k, if we average over a uniform choice of i_k, we have that $\Pr[c_{i_k} \in S_k] > rd/m$.

For $k = 0, \ldots, u$, let M_k be the set of $c \in L$ for which Condition 2 is not yet satisfied by the kth step. In other words, L_k is the set of c such that there does not exist a $k' \leq k$ such that $c_{i'_k} \in S_{k'}$.

$L_0 = L$. Clearly $L_{k+1} \subseteq L_k$. For each $c \in L_k$, $\Pr[c \in L_{k+1}] < 1 - rd/m$. By linearity of expectation, once we've fixed L_k, we have that $\mathbb{E}[|L_{k+1}|] < (1 - rd/m)|L_k|$. Therefore, $\Pr[|L_{k+1}| < (1 - rd/2m)|L_k|] \geq rd/2m$. Over all u trials, we therefore have that $\Pr[|L_u| < (1 - rd/2m)^u |L|] \geq$

$(rd/2m)^u$. If we choose $(1 - rd/2m)^u q \leq 1$, then $|L_t| < (1 - rd/2m)^u |L|$ can only be true if L_u is empty, meaning Condition 2 is met. Putting it all together, we have that the process outputs 1 with probability at least $[(1 - r)rd/2m]^u$. This completes the proof of Lemma 9. □

One final piece remains: actually computing $p_{L,c^*}(u, r)$. Unfortunately, it is not necessarily true that the probabilities can be exactly computed efficiently. This is analogous to the artificial abort of Waters [Wat05]. As in [Wat05], we instead have $\mathcal{A}'$ *estimate* p_{L,c^*} to within an error much less than $p_{\min}\epsilon$ by simply sunning $\mathsf{poly}(1/p_{\min}\epsilon, \lambda)$ trials of the process defining p_{L,c^*}. This will introduce an error $\ll p_{\min}\epsilon$ into the simulation, still resulting in a non-negligible success probability. This completes the proof. □

3.4 Putting It All Together

We can combine Theorems 3, 4, and 5 together, to get the following corollary:

Corollary 1. *Assume there exists iO and perfectly correct multi-party NIKE in the crs model with adaptive security without Shared Key or Register Malicious User queries. Then there exists perfectly correct (and also adversarially correct) multi-party NIKE in the plain model with adaptive security (under both Shared Key and Register Malicious User queries). If the original scheme has unbounded honest users, corruptions, and/or set size, then so does the resulting scheme.*

Corollary 1 shows that, for multiparty NIKE from iO, it suffices to work in the CRS model and ignore Shared Key and Register Malicious User queries.

4 The Equivalence of Multiparty NIKE and 1-SF-PRF

In this section, we show that NIKE is equivalent to a 1-SF-PRF.

4.1 From Multiparty NIKE to 1-SF-PRF

Let $(\mathsf{Setup}, \mathsf{Pub}, \mathsf{KeyGen})$ be a multiparty NIKE. We construct a 1-SF-PRF $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Constr}, \mathsf{EvalC})$ as follows:

- $\mathsf{Gen}(1^\lambda, 1^{|\Sigma|}, 1^\ell)$: let $c = n := \ell * |\Sigma|$. Run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^n, 1^c)$. In other words, we set the maximum number of users in a group to be ℓ, and the total number of users (and allowed corruptions) to be $n \times |\Sigma|$. For $z \in \Sigma, i \in [\ell]$, run $(\mathsf{pk}_{i,z}, \mathsf{sk}_{i,z}) \leftarrow \mathsf{Pub}(\mathsf{crs})$. Set $k = \{(\mathsf{pk}_{i,z}, \mathsf{sk}_{i,z})\}_{i \in [\ell], z \in \Sigma}$.
- $\mathsf{Eval}(k, x)$: run $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{pk}_{1,x_1}, \ldots, \mathsf{pk}_{\ell,x_\ell}, 1, \mathsf{sk}_{1,x_1})$
- $\mathsf{Constr}(k, i, z)$: output $k_{i,z} = (\{\mathsf{pk}_{i',z'}\}_{i' \in [\ell], z' \in \Sigma}, \mathsf{sk}_{i,z})$.
- $\mathsf{EvalC}(k_{i,z}, x)$: Output $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{pk}_{1,x_1}, \ldots, \mathsf{pk}_{\ell,x_\ell}, i, \mathsf{sk}_{i,z})$

Theorem 6. *If* $(\mathsf{SetupPub}, \mathsf{KeyGen})$ *is an adaptively secure multiparty NIKE without Register Malicious User queries in the CRS model, then* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Constr}, \mathsf{EvalC})$ *is an adaptively secure 1-SF-PRF. If* $(\mathsf{SetupPub}, \mathsf{KeyGen})$ *has security without Shared Key queries, then* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Constr}, \mathsf{EvalC})$ *has security without Eval queries.*

The proof is straightforward, and we only sketch it here. The $(\mathsf{SetupPub}, \mathsf{KeyGen})$ adversary simply runs the supposed adversary for $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Constr}, \mathsf{EvalC})$, making Corrupt User queries to answer any Constrain query in the obvious way, and making a Shared Key query to answer any Eval query.

4.2 Adding Eval Queries to a 1-SF-PRF

In Appendix C, we explain how to generically lift a 1-SF-PRF that is insecure under Eval queries into one that is secure. The idea follows from similar arguments as our NIKE compiler for adding Shared Key queries. We note that upgrading the case of 1-SF-PRFs is not strictly needed for our results, since the Eval queries ultimately correspond to Shared Key queries, which are handled by our compiler. Nevertheless, we include this result for 1-SF-PRFs for completeness.

4.3 From 1-SF-PRF to Special Constrained PRF

Here, we define an intermediate notion of constrained PRF, which enhances a 1-SF-PRF. The idea is that the symbol space Σ is now exponentially large. However, at the beginning a polynomial-sized set S is chosen, and a punctured key is revealed that allows for evaluating the PRF on any point *not* in S. The points in S then behave like the symbol space for a plain 1-SF-PRF, where it is possible to generate keys that fix any given position to some symbol in S.

Looking ahead to our NIKE construction, the set S will correspond to the public keys of the honest users of the system, while the rest of Σ will correspond to maliciously-generated keys. The abstraction of our special constrained PRF in this section is the missing link to formalize the connection between 1-SF-PRFs and NIKE as outlined in Section 1.

Definition 7 (Special Constrained PRF, Syntax)**.** *SC-PRF is a tuple of algorithms* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Punc}, \mathsf{EvalP}, \mathsf{Constr}, \mathsf{EvalC})$ *with the following syntax:*

- $\mathsf{Gen}(1^\lambda, |\Sigma|, 1^\ell, 1^n)$ *takes as input a security parameter* λ*, an alphabet size* $|\Sigma|$*, an input length* ℓ*, and a maximal set size* n*. Here,* $|\Sigma|$ *is represented in binary (thus allowing exponential-sized* Σ*), but everything else in unary.*
- $\mathsf{Eval}(k, x)$ *is the main evaluation algorithm, which is deterministic and takes as input a key* k *and* $x \in \Sigma^\ell$*, and outputs a string.*
- $\mathsf{Punc}(k, S)$ *is a randomized puncturing algorithm that takes as input a key* k *and set* $S \subseteq \Sigma$ *of size at most* n*. It outputs a punctured key* k_S*.*
- $\mathsf{EvalP}(k_S, x)$ *takes as input an* $x \in \Sigma^\ell$*, and outputs a value such that*

$$\mathsf{EvalP}(k_S, x) = \begin{cases} \bot & \text{if } x \in S^n \\ \mathsf{Eval}(k, x) & \text{if } x \notin S^n \end{cases}$$

- $\mathsf{Constr}(k, S, i, z)$ *is a potentially randomized constraining algorithm that takes as input a set* S*, a key* k*, an index* $i \in [\ell]$*, and symbol* $z \in S$*. It outputs a constrained key* $k_{S,i,z}$*.*

- $\mathsf{EvalC}(k_{S,i,z}, x)$ *takes as input a constrained key* $k_{S,i,z}$ *for an set/index/symbol triple* (S, i, z)*, and input* x*. It outputs a string. The correctness guarantee is:*

$$\mathsf{EvalC}(k_{S,i,z}, x) = \begin{cases} \bot & \text{if } x_i \neq z \\ \mathsf{Eval}(k, x) & \text{if } x_i = z \end{cases}$$

Definition 8 (Special Constrained PRF, Adaptive Security)**.** *Consider the following experiment with an adversary* $\mathcal{A}$*:*

- $\mathcal{A}$ *on input* 1^λ*, outputs* $|\Sigma|, 1^\ell, 1^n$*, and set* S *of size at most* n*. The challenger runs* $k \leftarrow \mathsf{Gen}(1^\lambda, |\Sigma|, 1^\ell, 1^n)$ *and* $k_S \leftarrow \mathsf{Punc}(k, S)$*. It sends* k_S *to* $\mathcal{A}$*.*
- *Then* $\mathcal{A}$ *can adaptively make the following types of queries:*
  - **Constrain.** $\mathcal{A}$ *sends* i, z*, and receives* $k_{S,i,z} \leftarrow \mathsf{Constr}(k, S, i, z)$*. The challenger records each* (i, z) *in a table* C*.*
  - **Eval.** $\mathcal{A}$ *sends an input* x*, and receives* $\mathsf{Eval}(k, x)$*. The challenger records each* x *in a table* E*. There is no limit to the number of Eval queries.*
  - **Challenge.** $\mathcal{A}$ *can make a* single *challenge query on an input* $x^* \in S^\ell$*. The challenger flips a random bit* $b \in \{0, 1\}$ *and replies with* $y^* = y_b$ *where* $y_0 = \mathsf{Eval}(k, x)$ *and* y_1 *is sampled uniformly and independently.*

  If at any time, $x^*_i = z$ *for some* $(i, z) \in C$ *or* $x^* \in E$*, the challenger immediately aborts and outputs a random bit.*
- *The adversary outputs bit* b'*. The challenger outputs 1 if* $b = b'$*, 0 otherwise.*

A Special Constrained PRF is adaptively secure if, for all PPT adversaries $\mathcal{A}$*, there exists a negligible function* ϵ *such that the challenger outputs 1 with probability at most* $\frac{1}{2} + \epsilon$*.*

Theorem 7. *If 1-SF-PRFs exist, then so do Special Constrained PRFs.*

The proof of Theorem 7 use purely combinatorial techniques. The idea is to set the symbol space Σ for the Special Constrained PRF to be codewords over the symbol space for the 1-SF-PRF, where the code is an error correcting code with certain properties. We defer the details to Appendix C.

4.4 From Special Constrained PRF to Multiparty NIKE with Setup

As a warm up, we construct multiparty NIKE in the common reference string model. We will need the following ingredients:

Definition 9. *A* single-point binding (SPB) signature *is a quadruple* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{GenBind})$ *where* $\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}$ *satisfy the usual syntax of a signature scheme. Additionally, we have the following:*

- $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}(1^\lambda, m)$ *takes as input a message* m*, and produces a verification key* vk *and signature* σ*.*
- *For any messages* $m, m' \neq m$*, with overwhelming probability over the choice of* $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}(1^\lambda, m)$*,* $\mathsf{Ver}(\mathsf{vk}, m', \sigma') = \bot$ *for any* σ'*. That is, there is no message* $m' \neq m$ *where there is a valid signature of* m' *relative to* vk*.*

- *For any* m, $\mathsf{GenBind}(1^\lambda, m)$ *and* $(\mathsf{vk}, \mathsf{Sign}(\mathsf{sk}, m))$ *are indistinguishable, where* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. *This property implies that* $\mathsf{Ver}(\mathsf{vk}, m, \sigma)$ *accepts, when* $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}(1^\lambda, m)$.

Definition 10. *A* multi-point binding (MPB) hash function *is a triple* $(\mathsf{Gen}, H, \mathsf{GenBind})$ *where:*

- $\mathsf{Gen}(1^\lambda, 1^n)$ *takes as input the security parameter* λ*, and an upper bound* n *on the number of inputs to bind. It produces a hashing key* hk.
- $H(\mathsf{hk}, x)$ *deterministically produces a hash* h.
- $\mathsf{GenBind}(1^\lambda, 1^n, S^*)$ *takes as input* λ, n*, and also a set* S^* *of inputs of size at most* n. *It produces a hashing key* hk *with the property that, with overwhelming probability over the choice of* $\mathsf{hk} \leftarrow \mathsf{GenBind}(1^\lambda, 1^n, S^*)$*, for any* $x \in S^*$ *and any* $x' \neq x$ *(which may or may not be in* S^**),* $H(\mathsf{hk}, x) \neq H(\mathsf{hk}, x')$.
- *For any* n *and any set* S^* *of size at most* n, $(S^*, \mathsf{Gen}(1^\lambda, 1^n))$ *is computationally indistinguishable from* $(S^*, \mathsf{GenBind}(1^\lambda, 1^n, S^*))$.

A single-point *binding (SPB) hash function is as above, except we fix* $n = 1$.

We will rely on the following Lemmas from Guan, Wichs, and Zhandry [GWZ21]:

Lemma 10 ([GWZ21])**.** *Assuming one-way functions exist, so do single-point binding signatures.*

Lemma 11 ([GWZ21])**.** *Assuming one-way functions and iO exist, so do single-point binding hash functions.*

We now give an adaptation of Lemma 11 to achieve multi-point binding hashes:

Lemma 12. *Assuming one-way functions and iO exist, then so do multi-point binding hash functions.*

This lemma is proved in Appendix C, following almost identical ideas to the proof of Lemma 11 from [GWZ21].

We use single/multi-point binding hash functions in order to statistically bind to an input m (or set of inputs S^*) with a hash that is much smaller than m. Such hash functions will contain many collisions, but the point binding guarantee means that there is no collision with m or S^*. The SPB signature is used for similar reasons.

Our NIKE Construction. We don't bound collusion queries c (that is, the number of corruption queries), but bound the number of honest users, which implicitly bounds the collusion queries at n.

- $\mathsf{Setup}(1^\lambda, 1^\ell, 1^n)$: Run $\mathsf{hk} \leftarrow \mathsf{Gen}_{Hash}(1^\lambda, 1^n)$. Let $\mathcal{Y}$ be the range of H. Also sample $k \leftarrow \mathsf{Gen}_{PRF}(1^\lambda, |\mathcal{Y}|, 1^\ell, 1^n)$. Let $\mathsf{KGen}_{\mathsf{hk},k}$ be the program given in Figure 5, padded to the maximum size of the programs in Figures 5 and 6, and let $\widehat{\mathsf{KGen}} = \mathsf{iO}(\mathsf{KGen}_{\mathsf{hk},k})$. Output $\mathsf{crs} = \widehat{\mathsf{KGen}}$.
- $\mathsf{Pub}(\mathsf{crs})$: Sample a random message m and run $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}_{Sig}(1^\lambda, m)$. Output $\mathsf{pk} = \mathsf{vk}$ and $\mathsf{sk} = (m, \sigma)$.
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_\ell, i, \mathsf{sk}_i)$: assume the pk_j are sorted in order of increasing pk according to some fixed ordering; if the pk_j are not in order sort them, and change i accordingly. Write $\mathsf{crs} = \widehat{\mathsf{KGen}}$, $\mathsf{pk}_j = \mathsf{vk}_j$ and $\mathsf{sk}_i = (m_i, \sigma_i)$. Then output $\widehat{\mathsf{KGen}}(\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell, i, m_i, \sigma_i)$.

Figure 5: The program $\mathsf{KGen}_{\mathsf{hk},k}$.

Inputs: $\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell, i, m_i, \sigma_i$
Constants: hk, k

1. If the vk_i are not sorted in increasing order, immediately abort and output $\perp$.
2. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
3. For each $t \in [\ell]$, let $u_t = H(\mathsf{hk}, \mathsf{vk}_t)$.
4. Output $\mathsf{Eval}_{PRF}(k, u_1||u_2|| \ldots ||u_\ell)$

Correctness. We must show that for any n and $i, j \in [\ell]$, $\mathsf{KeyGen}(\mathsf{crs}, \{\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell\}, i, \mathsf{sk}_i)$ outputs a value equal to $\mathsf{KeyGen}(\mathsf{crs}, \{\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell\}, j, \mathsf{sk}_j)$ with overwhelming probability. This follows from the correctness of the signature scheme. With overwhelming probability, $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ $=\mathsf{Ver}(\mathsf{vk}_j, m_j, \sigma_j)=1$. Once the signature check passes, the outputs are identical.

4.4.1 Security.

We will prove security via a sequence of hybrid experiments.

- $\mathsf{Game}_{\text{real}}$: This corresponds to the security game.
  - **Setup Phase:**
    The challenger samples $\mathsf{hk} \leftarrow \mathsf{Gen}_{Hash}(1^\lambda, 1^n)$.
    Next, it samples $k \leftarrow \mathsf{Gen}_{PRF}(1^\lambda, |\mathcal{Y}|, 1^\ell, 1^n)$.
    The challenger computes $\widehat{\mathsf{KGen}} = \mathsf{iO}(\mathsf{KGen}_{\mathsf{hk},k})$ and sends $\mathsf{crs} = \widehat{\mathsf{KGen}}$ to the adversary. It also maintains a table T which is initially empty.
  - **Pre-challenge Queries** The adversary makes the following queries:
    - *Honest user registration query*: For the i^{th} registration query, the challenger chooses m_i^*, computes $(\mathsf{vk}_i^*, \sigma_i^*) \leftarrow \mathsf{GenBind}_{Sig}(1^\lambda, m_i^*)$, sets vk_i^* as the public key and (m_i^*, σ_i^*) as the secret key. It adds $(\mathsf{vk}_i^*, (m_i^*, \sigma_i^*), 0)$ to the table T.
    - *Corruption query*: On receiving a corruption query for vk_i^*, the challenger sends (m_i^*, σ_i^*) to the adversary, and updates the i^{th} entry in T to $(\mathsf{vk}_i^*, (m_i^*, \sigma_i^*), 1)$.
    - *Registering Malicious user*: On receiving pk, the challenger adds $(\mathsf{pk}, \perp, 1)$ to T.
  - **Challenge Query** On receiving $(\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$, the challenger checks that the table T contains a $(\mathsf{vk}_i, (m_i, \sigma_i), 0)$ for each $i \in [\ell]$. If so, it chooses a random bit $b \leftarrow \{0, 1\}$. If $b = 0$, it sends $\mathsf{Eval}_{PRF}(k, u_1|| \ldots ||u_\ell)$, where $u_i = H(\mathsf{hk}, \mathsf{vk}_i)$. Else it sends a uniformly random string.
  - **Post-challenge Queries** Same as pre-challenge queries.
  - **Guess** Finally, the adversary sends its guess b', and wins if $b = b'$.
- Game_1: In this experiment, the challenger chooses the ℓ (vk, σ) pairs during setup. These are used to answer registration queries. The distribution of all components is identical to that in the previous experiment.

– **Setup Phase:**

For $j \in [n]$, the challenger chooses m_j^* and $(\mathsf{vk}_j^*, \sigma_j^*) \leftarrow \mathsf{GenBind}_{Sig}(1^\lambda, m_j^*)$.

The challenger samples $\mathsf{hk} \leftarrow \mathsf{Gen}_{Hash}(1^\lambda, 1^n)$.

Next, it samples $k \leftarrow \mathsf{Gen}_{PRF}(1^\lambda, |\mathcal{Y}|, 1^n, 1^\ell)$.

The challenger computes $\widehat{\mathsf{KGen}} = \mathsf{iO}(\mathsf{KGen}_{\mathsf{hk},k})$ and sends $\mathsf{crs} = \widehat{\mathsf{KGen}}$ to the adversary. It also maintains a table T which is initially empty.

- Game_2: In this experiment, the challenger uses the honest users' verification keys to sample a hash key that is binding to all the verification keys.

  – **Setup Phase:**

  For $j \in [n]$, the challenger chooses m_j^* and $(\mathsf{vk}_j^*, \sigma_j^*) \leftarrow \mathsf{GenBind}_{Sig}(1^\lambda, m_j^*)$.

  The challenger samples $\mathsf{hk} \leftarrow \mathsf{GenBind}_{Hash}\left(1^\lambda, \{\mathsf{vk}_i^*\}_{i\in[n]}\right)$.

  Next, it samples $k \leftarrow \mathsf{Gen}_{PRF}(1^\lambda, |\mathcal{Y}|, 1^\ell, 1^n)$.

  The challenger computes $\widehat{\mathsf{KGen}} = \mathsf{iO}(\mathsf{KGen}_{\mathsf{hk},k})$ and sends $\mathsf{crs} = \widehat{\mathsf{KGen}}$ to the adversary. It also maintains a table T which is initially empty.

- Game_3: In this game, the challenger uses a different (but functionally identical) program ($\mathsf{KGenAlt}$, defined in Figure 6) for computing the CRS.

  – **Setup Phase:**

  For $j \in [n]$, the challenger chooses m_j^* and $(\mathsf{vk}_j^*, \sigma_j^*) \leftarrow \mathsf{GenBind}_{Sig}(1^\lambda, m_j^*)$.

  The challenger samples $\mathsf{hk} \leftarrow \mathsf{GenBind}_{Hash}\left(1^\lambda, \{\mathsf{vk}_j^*\}_{j\in[n]}\right)$.

  Next, it samples $k \leftarrow \mathsf{Gen}_{PRF}(1^\lambda, |\mathcal{Y}|, 1^\ell, 1^n)$.

  The challenger then computes $u_j^* = H(\mathsf{hk}, \mathsf{vk}_j^*)$ and sets $S = \{u_j^*\}_{j\in[n]}$.

  It computes $K_S \leftarrow \mathsf{Punc}(k, S)$ and constrained keys

  $K_j^* = \left(\mathsf{Constr}(k, S, t, u_j^*)\right)_{t\in[\ell]}$. It sets $v_j^* = m_j^* \oplus K_j^*$ for each $j \in [n]$.

  The challenger computes $\widehat{\mathsf{KGenAlt}} = \mathsf{iO}\left(\mathsf{KGenAlt}_{\mathsf{hk},\{u_j^*, v_j^*, K_j^*\}, K_S}\right)$ and sends $\mathsf{crs} = \widehat{\mathsf{KGenAlt}}$ to the adversary. It also maintains a table T which is initially empty.

- Game_4: In this experiment, the challenger chooses the verification keys using Gen_{Sig} instead of $\mathsf{GenBind}_{Sig}$.

  – **Setup Phase:**

  For $j \in [n]$, the challenger chooses m_j^*, $(\mathsf{sk}_j^*, \mathsf{vk}_j^*) \leftarrow \mathsf{Gen}_{Sig}(1^\lambda)$ and $\sigma_j^* \leftarrow \mathsf{Sign}(\mathsf{sk}_j^*, m_j^*)$.

  The challenger samples $\mathsf{hk} \leftarrow \mathsf{GenBind}_{Hash}\left(1^\lambda, \{\mathsf{vk}_j^*\}_{j\in[n]}\right)$.

  Next, it samples $k \leftarrow \mathsf{Gen}_{PRF}(1^\lambda, |\mathcal{Y}|, 1^\ell, 1^n)$.

Figure 6: The program $\mathsf{KGenAlt}_{\mathsf{hk},\{u_j^*,v_j^*,K_j^*\},K_S}$.

Inputs: $\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell, i, m_i, \sigma_i$
Constants: Hash key hk
$S = \left\{u_j^*\right\}_{j\in[n]}$
$\left\{v_j^*\right\}_{j\in[n]}$
Punctured key K_S

1. If the vk_i are not sorted in increasing order, immediately abort and output $\perp$.
2. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
3. For each $t \in [\ell]$, let $u_t = H(\mathsf{hk}, \mathsf{vk}_t)$.
4. If $u_i \in \{u_j^*\}_{j\in[n]}$, compute $K_j^* = \left(K_{j,t}^* \right)_{t\in[\ell]} = m_i \oplus v_i^*$,
   then output $\mathsf{EvalC}(K_{j,i}^*, u_1||u_2||\ldots||u_\ell)$. Else output $\mathsf{EvalP}(K_S, u_1||u_2||\ldots||u_\ell)$.

The challenger then computes $u_j^* = H(\mathsf{hk}, \mathsf{vk}_j^*)$ and sets $S = \{u_j^*\}_{j\in[n]}$.
It computes $K_S \leftarrow \mathsf{Punc}(k, S)$ and constrained keys $K_j^* = \left(\mathsf{Constr}(k, S, t, u_j^*) \right)_{t\in[\ell]}$ $\forall j \in [n]$.
It sets $v_j^* = m_j^* \oplus K_j^*$ for each $j \in [n]$.
The challenger computes $\widehat{\mathsf{KGenAlt}} = \mathsf{iO}\left(\mathsf{KGenAlt}_{\mathsf{hk},\{u_j^*,v_j^*,K_j^*\},K_S}\right)$ and sends $\mathsf{crs} = \widehat{\mathsf{KGenAlt}}$ to the adversary. It also maintains a table T which is initially empty.

- Game_5: This game represents a syntactic change. Instead of choosing m_j^* first and then computing v_j^*, the challenger chooses uniformly random v_j^*, and sets $m_j^* = v_j^* \oplus K_j^*$. In terms of the adversary's view, this experiment is identical to the previous one.

  – **Setup Phase:**
  For $j \in [n]$, the challenger chooses $(\mathsf{sk}_j^*, \mathsf{vk}_j^*) \leftarrow \mathsf{Gen}_{Sig}(1^\lambda)$.
  The challenger samples $\mathsf{hk} \leftarrow \mathsf{GenBind}_{Hash}\left(1^\lambda, \left\{\mathsf{vk}_j^*\right\}_{j\in[n]}\right)$.
  Next, it samples $k \leftarrow \mathsf{Gen}_{PRF}(1^\lambda, |\mathcal{Y}|, 1^\ell, 1^n)$.
  The challenger then computes $u_j^* = H(\mathsf{hk}, \mathsf{vk}_j^*)$ and sets $S = \{u_j^*\}_{j\in[n]}$.
  It computes $K_S \leftarrow \mathsf{Punc}(k, S)$.
  The constrained keys are not chosen during setup. It chooses v_j^* for each $j \in [\ell]$.
  The challenger computes $\widehat{\mathsf{KGenAlt}} = \mathsf{iO}\left(\mathsf{KGenAlt}_{\mathsf{hk},\{u_j^*,v_j^*\},K_S}\right)$ and sends $\mathsf{crs} = \widehat{\mathsf{KGenAlt}}$ to the adversary. It also maintains a table T which is initially empty.

– **Pre-Challenge Queries:**

* *Corruption query*: On receiving a corruption query for vk_i^*, the challenger computes $K_i^* = \left(\mathsf{Constr}(k, S, t, u_j^*) \right)_{t \in [\ell]}$. It then computes $m_j^* = v_j^* \oplus K_j^*$ and sends (m_i^*, σ_i^*) to the adversary, and updates the i^{th} entry in T to $(\mathsf{vk}_i^*, (m_i^*, \sigma_i^*), 1)$.

In Appendix C.4, we analyse the adversary's advantage in each of these experiments, and show that these games are computationally indistinguishable.

5 Construction of 1-SF-PRFs

The previous section worked to distill adaptively secure NIKE to the more basic primitive of constrained PRFs for one symbol fixing. While these transformations simplify the problem, the central barriers to proving adaptive security still remain. In this section we address these head on.

Let's review the main issues for adaptivity. Consider an adversary $\mathcal{A}$ that first makes several constrained key queries $(\mathsf{index}_1, \mathsf{sym}_1), \ldots, (\mathsf{index}_Q, \mathsf{sym}_Q)$. Next the $\mathcal{A}$ submits a challenge input x^* such that $x_i^* \neq z$ for any pre-challenge key query (i, z) and receives back the challenge output from the challenger. Before submitting its guess, $\mathcal{A}$ will first perform some consistency checks on the constrained keys it received. For example, it can run the evaluation algorithm on multiple points that are valid for different sets of constrained keys and verify that it receives the same output from each execution. If not, it aborts and refuses to submit its guess.

Dealing with such an attacker is difficult for multiple reasons. First, a reduction algorithm cannot simply guess x^* or which index/symbol pairs will be queried without an exponential loss. Second, it cannot issue constrained keys that are deviate much from each other less this be detected by $\mathcal{A}$'s consistency checks.

We overcome these issues with a proof strategy where the challenger gradually issues constrained keys that deviate from a canonical PRF which is used to evaluate on the challenge input. However, we endeavor to keep all subsequent issued keys consistent with any introduced deviation so that this will avoid being detected.

Diving deeper our construction will use constrained keys which are obfuscated programs. Initially, the obfuscated program will simply check if an input x is consistent with the single symbol fixing of the key. If so, it evaluates the canonical PRF which is a Naor-Reingold style PRF.

The proof will begin by looking at the first key that is issued by the challenger for some query $(\mathsf{index}_1, \mathsf{sym}_1)$. For this key the obfuscated program will branch off and evaluate any inputs x where $x_{\mathsf{index}_1} = \mathsf{sym}_1$ in a different, but functionally equivalent way to the canonical PRF. By the security of iO this will not be detected. Moreover, this alternative evaluation for when $x_{\mathsf{index}_1} = \mathsf{sym}_1$ will be adopted by all further issued keys. Once this alternative pathway is set for all keys, we can change the evaluation on such inputs to be inconsistent with the canonical PRF, but mutually consistent with all issued keys. This follows from the DDH assumption. The proof can then proceed to the transforming the second issued key in a similar way such that there is a separate pathway for all inputs x where $x_{\mathsf{index}_2} = \mathsf{sym}_2$. The one exception is that the second and all future keys will give prioritization to the first established pathway whenever we have an input x where both $x_{\mathsf{index}_1} = \mathsf{sym}_1$ and $x_{\mathsf{index}_2} = \mathsf{sym}_2$.

The proof continues on in this way where each new key issued will establish an alternative evaluation which will be used except when it is pre-empted by an earlier established alternative. In

this manner the constrained keys issued will always be mutually consistent on inputs, even while the gradually deviate from the canonical PRF. Finally, at the end of the proof all issued keys will always used some alternative pathway for *all* evaluations. At this point we can use indistinguishability obfuscation again to remove information about the canonical PRF from the all obfuscated programs since it is never used. With this information removed no attacker can distinguish a canonical PRF output from a random value.

We remark that in order to execute our proof strategy, our initial obfuscated program must be as large as any program used in the proof. In particular, it must be large enough to contain an alternative evaluation programming for all corrupted keys. Thus our constrained PRF keys must grow in size proportional to $\ell \cdot |\Sigma|$ and our resulting NIKE is parameterized for a set number of collusions.

5.1 Construction

- $\mathsf{Gen}(1^\lambda, \Sigma, 1^\ell)$: The key generation algorithm first runs $\mathcal{G}(1^\lambda)$ to compute $(p, \mathbb{G})$. Next, it chooses $v \leftarrow \mathbb{G}$, exponents $e_{j,w} \leftarrow \mathbb{Z}_p$ for each $j \in [\ell]$, $w \in \Sigma$. The PRF key K consists of $(v, \{e_{j,w}\})$.
- $\mathsf{Eval}(\mathsf{K}, x)$: Let $\mathsf{K} = (v, \{e_{j,w}\})$ and $x = (x_1, \ldots, x_\ell) \in \Sigma^\ell$. The PRF evaluation on input x is v^t, where $t = \left(\prod_{j \le n} e_{j,x_j}\right)$.
- $\mathsf{Constr}(\mathsf{K}, i, z)$: The constrained key is an obfuscation of the $\mathsf{ConstrainedKey}_{\mathsf{K},i,z}$ program (defined in Figure 7). The program is sufficiently padded to ensure that its description is of the same size as the programs $\mathsf{ConstrainedKeyAlt}$, $\mathsf{ConstrainedKeyAlt}'$ and $\mathsf{ConstrainedKeyEnd}$ (defined in Figure 8 , 9 and 32 (Appendix D) respectively).

  It outputs $\mathsf{K}_{i,z} \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKey}_{\mathsf{K},i,z})$ as the constrained key.

$\mathsf{ConstrainedKey}_{\mathsf{K},i,z}$

Input: $x = (x_1, \ldots, x_\ell) \in \Sigma^\ell$
Constants: Group element v
Exponents $\{e_{j,w}\}_{j \in [\ell], w \in \Sigma}$
Constraining index/symbol $i \in [\ell], z \in \Sigma$

1. If $x_i \neq z$ output $\perp$.
2. Compute $t = \left(\prod_{j \le \ell} e_{j,x_j}\right)$.
3. Output v^t.

Figure 7: Program ConstrainedKey

- $\mathsf{EvalC}(\mathsf{K}_{i,z}, x)$: The constrained key $\mathsf{K}_{i,z}$ is an obfuscated program. The evaluation algorithm outputs $\mathsf{K}_{i,z}(x)$.

5.2 Security Proof

We will prove that the above construction satisfies security against unique-query adversaries, via a sequence of hybrid games. The first game corresponds to the original security game (security against

unique query adversary). Next, we define Q hybrid games $\{\mathsf{Game}_y\}_{y\in[Q]}$, where Q is a bound on the total number of constrained key queries by the adversary.

- $\mathsf{Game}_{\text{real}}$:
  - **Setup Phase:** The challenger chooses $v \leftarrow \mathbb{G}$, $e_{j,w} \leftarrow \mathbb{Z}_p$ for each $j \in [\ell], w \in \Sigma$. Let $\mathsf{K} = (v, (e_{j,w})_{j,w})$.
    The challenger also maintains an ordered list L of $(\mathsf{index}, \mathsf{sym})$ pairs. This list is initially empty, and for each (new) query, the challenger adds a tuple to L.
  - **Pre-challenge queries:** Next, the challenger receives pre-challenge constrained key queries. Let $(\mathsf{index}_j, \mathsf{sym}_j)$ be the j^{th} constrained key query. The challenger adds $(\mathsf{index}_j, \mathsf{sym}_j)$ to L.
    The challenger computes the constrained key
    $\mathsf{K}_j \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKey}_{\mathsf{K},\mathsf{index}_j,\mathsf{sym}_j})$ and sends K_j to the adversary.
  - **Challenge Phase:** Next, the adversary sends a challenge x^* such that $x^*_i \neq z$ for any pre-challenge key query (i, z). The challenger chooses $b \leftarrow \{0, 1\}$. If $b = 0$, the challenger computes $t = \prod_i e_{i,x^*_i}$ and sends v^t. If $b = 1$, the challenger sends a uniformly random group element in $\mathbb{G}$.
  - **Post-challenge queries:** The post-challenge queries are handled similar to the pre-challenge queries.
  - **Guess:** Finally, the adversary sends the guess b' and wins if $b = b'$.

- Game_y: In this game, the challenger uses an altered program for the first y constrained keys. It computes an obfuscation of $\mathsf{ConstrainedKeyAlt}$ (defined in Figure 8), and it is padded to be of the same size as $\mathsf{ConstrainedKey}$, $\mathsf{ConstrainedKeyAlt}'$ and $\mathsf{ConstrainedKeyEnd}$.
  - **Setup Phase:** The challenger chooses $v \leftarrow \mathbb{G}$, $h_j \leftarrow \mathbb{G}$ for all $j \in [y]$ and $e_{j,w} \leftarrow \mathbb{Z}_p$ for all $j \in [\ell], w \in \Sigma$. Let $H = (h_j)_{j\in[y]}$.
    The challenger also maintains an ordered list L of $(\mathsf{index}, \mathsf{sym})$ pairs which is initially empty.
  - **Pre-challenge queries:** Next, the challenger receives pre-challenge constrained key queries. Let $(\mathsf{index}_j, \mathsf{sym}_j)$ be the j^{th} constrained key query. The challenger adds $(\mathsf{index}_j, \mathsf{sym}_j)$ to L.
    Let $s = \min(y, j)$, and let L_s (resp. H_s) denote the first s entries in L (resp. H). The challenger computes the constrained key
    $\mathsf{K}_j \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKeyAlt}_{s,L_s,H_s,v,(e_{j,w}),\mathsf{index}_j,\mathsf{sym}_j})$ and sends K_j to the adversary.
  - **Challenge Phase:** Next, the adversary sends a challenge x^* such that $x^*_i \neq z$ for any pre-challenge key query (i, z). The challenger chooses $b \leftarrow \{0, 1\}$. If $b = 0$, the challenger computes $t = \prod_i e_{i,x^*_i}$ and sends v^t. If $b = 1$, the challenger sends a uniformly random group element in $\mathbb{G}$.
  - **Post-challenge queries:** The post-challenge queries are handled similar to the pre-challenge queries.
  - **Guess:** Finally, the adversary sends the guess b' and wins if $b = b'$.

$\mathsf{ConstrainedKeyAlt}_{s,L_s,H_s,v,(e_{j,w}),i,z}$

Input: $x = (x_1, \ldots, x_\ell) \in \Sigma^\ell$

Constants: $s \in \ell \cdot |\Sigma|$

$\text{List } L_s = \left(\left(\mathsf{index}_j, \mathsf{sym}_j\right)\right)_{j\in[s]}$

$H_s = (h_j)_{j\in[s]}$

Group element v,

Exponents $(e_{j,w})_{j,w}$,

Constraining index/symbol $i \in [\ell], z \in \Sigma$

1. If $x_i \neq z$ output $\perp$.
2. Compute $t = \left(\prod_{j\leq\ell} e_{j,x_j}\right)$.
3. Find the smallest $j \in [s]$ such that $x_{\mathsf{index}_j} = \mathsf{sym}_j$.
   (a) If such j exists, then output h_j^t.
   (b) Else output v^t.

Figure 8: Program $\mathsf{ConstrainedKeyAlt}$

5.2.1 Analysis

We will now show that $\mathsf{Game}_{\text{real}}$ and Game_y are computationally indistinguishable for all $y \in [Q]$. Finally, we will show that no polynomial time adversary has non-negligible advantage in Game_Q, thereby showing that the scheme is secure against *unique query adversaries*. For any adversary $\mathcal{A}$, let $\mathsf{adv}_{\mathcal{A},\text{real}}$ denote $\mathcal{A}$'s advantage in $\mathsf{Game}_{\text{real}}$, and let $\mathsf{adv}_{\mathcal{A},y}$ denote $\mathcal{A}$'s advantage in Game_y.

Lemma 13. *Assuming* iO *is secure, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ*,* $|\mathsf{adv}_{\mathcal{A},\text{real}} - \mathsf{adv}_{\mathcal{A},0}| \leq \mathsf{negl}(\lambda)$.

Proof. For $y = 0$, the lists L_y and H_y are empty, and as a result, the programs are functionally identical. On any input x, both programs output v^t. Therefore, their obfuscations are computationally indistinguishable. □

Lemma 14. *Fix any* $y \in [Q]$*. Assuming* DDH *and security of* iO*, for any PPT adversary* $\mathcal{A}$ *making at most* Q *queries, there exists a negligible function* negl *such that for all* λ*,* $|\mathsf{adv}_{\mathcal{A},y} - \mathsf{adv}_{\mathcal{A},y+1}| \leq \mathsf{negl}(\lambda)$.

Proof. We will define a few hybrid games to show that Game_y and Game_{y+1} are computationally indistinguishable. The main difference in the two games is with regard to the last $Q - y$ constrained key queries. Note that the first y constrained keys are identical in both experiments. For each of the last $Q - y$ constrained keys, if (i, z) is the constrained key query, then the adversary receives an obfuscation of

- $P_{y,i,z} \equiv \mathsf{ConstrainedKeyAlt}_{y,L_y,H_y,v,(e_{j,w}),i,z}$ in Game_y,
- $P_{y+1,i,z} \equiv \mathsf{ConstrainedKeyAlt}_{y+1,L_{y+1},H_{y+1},v,(e_{j,w}),i,z}$ in Game_{y+1}

Note that the programs $P_{y,i,z}$ and $P_{y+1,i,z}$ only differ on inputs x where $x_i = z$ (in one case the output is v^t, while in the other case the output is h_{y+1}^t). We will prove that these two hybrid games are indistinguishable, using a sequence of sub-hybrid experiments defined below.

- $\mathsf{Game}_{y,a}$: This security game is similar to Game_y, except that the challenger guesses the $(y+1)^{\text{th}}$ query in the setup phase.
  - **Setup Phase:** The challenger chooses $v \leftarrow \mathbb{G}$, $h_j \leftarrow \mathbb{G}$ for all $j \in [y]$ and $e_{j,w} \leftarrow \mathbb{Z}_p$ for all $j \in [\ell], w \in \Sigma$. Let $H_y = (h_j)_{j\in[y]}$.
    The challenger maintains an ordered list L of $(\mathsf{index}, \mathsf{sym})$ pairs which is initially empty.
    The challenger also chooses $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1}) \leftarrow [\ell] \times \Sigma$.
  - **Pre-challenge queries:** Next, the challenger receives pre-challenge constrained key queries. Let $(\mathsf{index}_q, \mathsf{sym}_q)$ be the q^{th} constrained key query. The challenger adds $(\mathsf{index}_q, \mathsf{sym}_q)$ to L.
    If the $(y+1)^{\text{th}}$ query is not $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$, then the challenger aborts. The adversary wins with probability $1/2$.
    Let $s = min(y, q)$, and let L_s denote the first s entries in L. The challenger computes the constrained key
    $\mathsf{K}_q \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKeyAlt}_{s,L_s,H_s,v,(e_{j,w}),\mathsf{index}_q,\mathsf{sym}_q})$ and sends K_q to the adversary.
  - **Challenge Phase:** Next, the adversary sends a challenge x^* such that $x_i^* \neq z$ for any pre-challenge key query (i, z). The challenger chooses $b \leftarrow \{0, 1\}$. If $b = 0$, the challenger computes $t = \prod_i e_{i,x_i^*}$ and sends v^t. If $b = 1$, the challenger sends a uniformly random group element in $\mathbb{G}$.
  - **Post-challenge queries:** The post-challenge queries are handled similar to the pre-challenge queries.
  - **Guess:** Finally, the adversary sends the guess b' and wins if $b = b'$.

- $\mathsf{Game}_{y,b}$: This security game is similar to $\mathsf{Game}_{y,a}$, except that the challenger chooses the h_j constants and one of the $e_{j,w}$ exponents differently. However, the distribution of these components is identical to their distribution in the previous game.
  - **Setup Phase:** The challenger chooses $g \leftarrow \mathbb{G}$, $b \leftarrow \mathbb{Z}_p$, $c_j \leftarrow \mathbb{Z}_p$ for all $j \in [y]$.
    It sets $v = g^b$, $h_j = g^{c_j}$.
    The challenger maintains an ordered list L of $(\mathsf{index}, \mathsf{sym})$ pairs which is initially empty.
    The challenger also chooses $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1}) \leftarrow [\ell] \times \Sigma$.
    It chooses $e_{j,w} \leftarrow \mathbb{Z}_p$ for all $j \in [n], w \in \Sigma$, $(j, w) \neq (\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$.
    It chooses $a \leftarrow \mathbb{Z}_p$ and sets $e_{\mathsf{index}_{y+1},\mathsf{sym}_{y+1}} = a$, $A = g^a$ and $T = v^a$.
    Note that the terms A and T are not used in this experiment; they will be used in some of the following hybrid experiments. Let $H_y = (h_j)_{j\in[y]}$.

- $\mathsf{Game}_{y,c}$: In this security game, the challenger computes the constrained keys differently. Instead of sending an obfuscation of $\mathsf{ConstrainedKeyAlt}$ (with appropriate hardwired constants), the challenger computes an obfuscation of $\mathsf{ConstrainedKeyAlt}'$ (with appropriate hardwired constants). The program $\mathsf{ConstrainedKeyAlt}'$ is defined in Figure 9, and is padded to be of the same size as $\mathsf{ConstrainedKey}$, $\mathsf{ConstrainedKeyAlt}$ and $\mathsf{ConstrainedKeyEnd}$.

  The main difference is that $\mathsf{ConstrainedKeyAlt}'$ does not contain the exponent $e_{\mathsf{index}_{y+1},\mathsf{sym}_{y+1}}$ (recall $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$ is the $(y+1)^{\text{th}}$ constrained key query, and the challenger guesses this query during setup). Instead, the program contains $g^{e_{\mathsf{index}_{j+1},\mathsf{sym}_{j+1}}}$ and $v^{e_{\mathsf{index}_{j+1},\mathsf{sym}_{j+1}}}$. As a result, the final output is computed differently (although the outputs are identical).

  We will show that the two programs are functionally identical, and therefore their obfuscations are computationally indistinguishable.

  - **Pre-challenge queries:** Let $(\mathsf{index}_q, \mathsf{sym}_q)$ be the q^{th} constrained key query. The challenger adds $(\mathsf{index}_q, \mathsf{sym}_q)$ to L. Let L_j denote the first j entries in L.
    If $q \leq y$, the challenger computes
    $\mathsf{K}_q \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKeyAlt}_{q,L_q,H_q,v,(e_{j,w}),\mathsf{index}_q,\mathsf{sym}_q})$ and sends K_q to the adversary.
    If the $(y+1)^{\text{th}}$ query is not $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$,[4] then the challenger aborts. The adversary wins with probability $1/2$.
    If $q > y$, the challenger sends an obfuscation of the following program:
    $\mathsf{ConstrainedKeyAlt}'_{y,L_y,(\mathsf{index}_{y+1},\mathsf{sym}_{y+1}),\{c_y\},g,v,B,T,(e_{j,w})_{(j,w)\neq(\mathsf{index}_{y+1},\mathsf{sym}_{y+1})},\mathsf{index}_q,\mathsf{sym}_q}$

- $\mathsf{Game}_{y,d}$: In this security game, the challenger sets T to be a uniformly random element in $\mathbb{G}$.

  - **Setup Phase:** The challenger chooses $g \leftarrow \mathbb{G}$, $b \leftarrow \mathbb{Z}_p$, $c_j \leftarrow \mathbb{Z}_p$ for all $j \in [y]$.
    It set $v = g^b$, $h_j = g^{c_j}$.
    The challenger maintains an ordered list L of $(\mathsf{index}, \mathsf{sym})$ pairs which is initially empty.
    The challenger also chooses $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1}) \leftarrow [\ell] \times \Sigma$.
    It chooses $e_{j,w} \leftarrow \mathbb{Z}_p$ for all $j \in [\ell], w \in \Sigma$, $(j,w) \neq (\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$.
    It chooses $a \leftarrow \mathbb{Z}_p$ and sets $e_{\mathsf{index}_{y+1},\mathsf{sym}_{y+1}} = a$, $A = g^a$ and $T \leftarrow \mathbb{G}$. Let $H_y = (h_j)_{j\in[y]}$.

- $\mathsf{Game}_{y,e}$: This security game represents a syntactic change. We choose $h_{j+1} \leftarrow \mathbb{G}$ and set $T = h_{j+1}^a$. The group element h_{j+1} is not used anywhere else.

- $\mathsf{Game}_{y,f}$: In this experiment, the challenger uses $\mathsf{ConstrainedKeyAlt}$ for the last $Q - y$ constrained key queries. More formally, on receiving query (i, z), the challenger sends an obfuscation of $\mathsf{ConstrainedKeyAlt}_{y+1,L_{y+1},H_{y+1},v,(e_{k,w}),i,z}$. Here L_{y+1} and H_{y+1} are defined as in $\mathsf{Game}_{y,e}$.

- $\mathsf{Game}_{y,g}$: This security game is identical to $\mathsf{Game}_{y,f}$, and the changes in this game are syntactic. Instead of sampling exponents c_j and setting $h_j = g^{c_j}$, the challenger chooses $h_j \leftarrow \mathbb{G}$. Similarly, the challenger samples $v \leftarrow \mathbb{G}$, and samples all the exponents $e_{j,w} \leftarrow \mathbb{Z}_p$. Note that this experiment is identical to Game_{y+1}, except that the challenger guesses $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$ in the setup phase.

[4] Recall $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$ is chosen during the setup phase.

ConstrainedKeyAlt′

Input: $x = (x_1, \ldots, x_\ell) \in \Sigma^\ell$
Constants: $y \in [\ \ell \cdot |\Sigma|\ ]$
List of first y queries $L_y = \left(\left(\mathsf{index}_j, \mathsf{sym}_j\right)\right)_{j\in[y]}$
$(y+1)^{\text{th}}$ query $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$
exponents for computing $(h_j)_j : (c_j)_{j\in[y]}$
Group elements g, v, A, T
PRF eval exponents $= (e_{j,w})_{(j,w)\neq(\mathsf{index}_{y+1},\mathsf{sym}_{y+1})}$
Constraining index/symbol $i \in [\ell], z \in \Sigma$

1. If $x_i \neq z$ output $\perp$.
2. Compute t as follows:
   (a) If $x_{\mathsf{index}_{y+1}} = \mathsf{sym}_{y+1}$ then set $t = \left(\prod_{j\neq \mathsf{index}_{y+1}} e_{j,x_j}\right)$
   (b) Else $t = \left(\prod_j e_{j,x_j}\right)$
3. Find the smallest $j \in [y]$ such that $x_{\mathsf{index}_j} = \mathsf{sym}_j$.
   (a) If such j exists and $x_{\mathsf{index}_{y+1}} = \mathsf{sym}_{y+1}$ then output $(A)^{t\cdot c_j}$
   (b) If such j exists and $x_{\mathsf{index}_{y+1}} \neq \mathsf{sym}_{y+1}$ then output $\left(g_j^c\right)^t$
   (c) Else if no such j exists and $x_{\mathsf{index}_{y+1}} = \mathsf{sym}_{y+1}$ output $(T)^t$.
   (d) Else if no such j exists and $x_{\mathsf{index}_{y+1}} \neq \mathsf{sym}_{y+1}$ output v^t.

Figure 9: Program ConstrainedKeyAlt′

Claim 1. *For any $y \in [Q]$, and any adversary $\mathcal{A}$ making at most Q constrained key queries, $|\mathsf{adv}_{\mathcal{A},y} - \mathsf{adv}_{\mathcal{A},y+1}| = \frac{1}{\ell\cdot|\Sigma|}\left(|\mathsf{adv}_{\mathcal{A},y,a} - \mathsf{adv}_{\mathcal{A},y,g}|\right)$.*

Proof. Note that the only difference between $\mathsf{Game}_{y,a}$ and Game_y is that the challenger guesses the $(y+1)^{\text{th}}$ constrained key query in the setup phase. Similarly, the only difference between $\mathsf{Game}_{y,g}$ and Game_{y+1} is that the challenger guesses the $(y+1)^{\text{th}}$ constrained key query. This guess is correct with probability $1/(\ell \cdot |\Sigma|)$, and therefore $|\mathsf{adv}_{\mathcal{A},y} - \mathsf{adv}_{\mathcal{A},y+1}| = \frac{1}{\ell\cdot|\Sigma|}\left(|\mathsf{adv}_{\mathcal{A},y,a} - \mathsf{adv}_{\mathcal{A},y,g}|\right)$. □

Therefore, it suffices to show that $\mathsf{Game}_{y,a}, \ldots, \mathsf{Game}_{y,g}$ are computationally indistinguishable. This is proved in Appendix D. Proving the indistinguishability of these hybrids completes the proof of Lemma 14. □

6 NIKE Scheme with Unbounded Honest Users

Construction Overview : In this scheme, our construction supports an unbounded number of honest users, but at most c corruptions. The setup algorithm takes this corruption bound c, together with a bound ℓ on the number of parties that can derive a shared key. It outputs a crs, which is an obfuscated program that will be used for generating the shared key. The program takes as input ℓ public keys and one secret key. Each public key consists of a random string s and a signature scheme's verification key vk. The corresponding secret key is a (random) message m

and a signature σ on m. The crs program, on input $\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell$ and $\mathsf{sk}_i = (m_i, \sigma_i)$, first checks that σ_i is a valid signature on m_i. Next, it computes a hash of each of the ℓ public keys (that is, $u_j = \mathsf{Hash}(\mathsf{pk}, \mathsf{pk}_j)$). The crs program also has a PRF key K_{main} hardwired, using which it computes a pseudorandom integer $t_j = \mathsf{F}(\mathsf{K}_{\text{main}}, u_j)$. It finally computes the product of all these ℓ integers (let t denote the product of $t_1, t_2, \ldots, t_\ell$), and outputs v^t, where v is a random group element hardwired in the program.

The 'publish' algorithm Pub is used to sample a public key and the corresponding secret key. It chooses a random message m, then samples a verification key vk and a signature σ that is binding to m (using the $\mathsf{GenBind}_{sig}$ algorithm). It then chooses a random string s. The public key is (s, vk), and the corresponding secret key is (m, σ).

The key generation algorithm simply takes ℓ public keys, one secret key, and runs the crs program to sample the shared key.

Construction: Let ℓ_{ct} denote the size of ciphertexts output by the PPDE scheme with message space $\{0,1\}^\lambda$. Let ℓ_{hash} denote the output length of the hash function (note that the output length depends only on λ, and does not depend on the message space for the hash function). Let ℓ_{m} denote the size of message space of the signature scheme, and ℓ_{vk} the size of the verification key output by Gen_{Sign}. Finally, let ℓ_{hk} denote the size of the hash key output by Gen_{Hash}.

- $\mathsf{Setup}(1^\lambda, 1^\ell, 1^c)$: The setup algorithm takes as input the security parameter λ, a bound c on the number of corrupt users and a bound ℓ on the number of parties that can derive a shared key.

  Run $\mathsf{hk} \leftarrow \mathsf{Gen}_{Hash}(1^\lambda, 1)$.[5] Let $\mathcal{Y}$ be the range of H. Also sample $\mathsf{K}_{\text{main}} \leftarrow \mathsf{Gen}_{PRF}(1^\lambda, |\mathcal{Y}|)$. Let $\mathsf{KGen}_{\mathsf{hk},\mathsf{K}_{\text{main}},v}$ be the program given in Figure 10, and let $\widehat{\mathsf{KGen}} = \mathsf{iO}(\mathsf{KGen}_{\mathsf{hk},\mathsf{K}_{\text{main}},v})$. Output $\mathsf{crs} = \widehat{\mathsf{KGen}}$.

Figure 10: The program $\mathsf{KGen}_{\mathsf{hk},\mathsf{K}_{\text{main}},v}$.

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: $\mathsf{hk}, \mathsf{K}_{\text{main}}, v$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $j \in [\ell]$, let $u_j = H(\mathsf{hk}, (s_j, \mathsf{vk}_j))$.
3. For each $j \in [\ell]$, let $t_j = \mathsf{F}(\mathsf{K}_{\text{main}}, u_j)$.
4. Compute $t = \prod_j t_j$ and output v^t.

- $\mathsf{Pub}(\mathsf{crs})$: Sample a random message $m \leftarrow \{0,1\}^{\ell_{\text{m}}}$ and run $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}_{Sig}(1^\lambda, m)$. Choose $s \leftarrow \{0,1\}^{\ell_{\text{ct}}}$, and output $\mathsf{pk} = (s, \mathsf{vk})$ and $\mathsf{sk} = (m, \sigma)$.
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_\ell, i, \mathsf{sk}_i)$: let $\mathsf{crs} = \widehat{\mathsf{KGen}}$, $\mathsf{pk}_j = (s_j, \mathsf{vk}_j)$ for all $j \in [\ell]$, and $\mathsf{sk}_i = (m_i, \sigma_i)$. The algorithm outputs $\widehat{\mathsf{KGen}}(\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell, i, m_i, \sigma_i)$.

[5] We only need single-point binding hash function for this construction.

Correctness and Size of Various Components:

Correctness follows immediately from the correctness of iO and the correctness of the signature scheme.

We will now list the sizes of the various components. First, the size of the public key is $\ell_{\mathrm{ct}} + \ell_{\mathrm{vk}}$, which is polynomial in λ, ℓ and c. Note that the message space of the signature scheme is $\{0,1\}^{\ell_{\mathrm{m}}}$, and looking ahead, the size of the message should be equal to the size of the obfuscation of program $\mathsf{EvalProg}$ defined in Figures 13. The size of the obfuscation of $\mathsf{EvalProg}$, and therefore ℓ_{m}, is bounded by $\mathsf{poly}(\lambda, \ell, c)$. [6] Since the message size is of size $\mathsf{poly}(\lambda, \ell, c)$, the verification keys and the signatures are of size $\mathsf{poly}(\lambda, c, \ell)$. The size of the hash key generated by Gen_{hash} depends on ℓ_{ct} (the size of s) and ℓ_{vk} (the size of the verification key). As a result, the hash key has size $\mathsf{poly}(\lambda, c, \ell)$. Finally, the crs is an obfuscation of KGen, and therefore it has size bounded by $\mathsf{poly}(\lambda, c, \ell)$.

Security: The security proof goes via a sequence of hybrid experiments. Let Q denote the number of honest user registration queries made by the adversary (note that Q can be any unbounded polynomial; it is not bounded during setup). First, we have experiments $\mathsf{Game}_{A,1}, \ldots, \mathsf{Game}_{A,Q}$. In the y^{th} experiment $\mathsf{Game}_{A,y}$, the challenger alters the crs program, as well as the first y honest user registrations. For the j^{th} honest user registration, if $j \leq y$, then the string s_j is a pseudorandom deterministic encryption of j, and the message m_j and the verification key vk_j are derived from s_j (m_j and r_j are pseudorandom strings derived from s_j, and $(m_j, \sigma_j) = \mathsf{GenBind}_{sig}(m_j; r_j)$). Let us call these public keys of $\mathsf{type} = 1$ (by default, $\mathsf{type} = 0$).

The crs program, on input ℓ public keys and a secret key, checks how many of these public keys are of $\mathsf{type} = 1$. It has an integer a hardwired, and if there are cnt such public keys, (and assuming the secret key is valid), the program sets $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and outputs v^t. In Section 6.1, we provide an overview of why $\mathsf{Game}_{A,y}$ and $\mathsf{Game}_{A,y+1}$ are indistinguishable.

At the end of these Q hybrid experiments, in $\mathsf{Game}_{A,Q}$, all honestly registered keys have $\mathsf{type} = 1$. As a result, when the attacker sends an ℓ-size subset of these honestly registered, non-corrupted keys as the challenge set, the challenger either outputs $v^{a^\ell \cdot \prod_j t_j}$, or a uniformly random group element (recall, t_j is derived from the public key pk_j by first hashing it, followed by PRF evaluation). Looking ahead, we would like to use the DDH-powers assumption to argue that $v^{a^\ell \cdot \prod_j t_j}$ is indistinguishable from a uniformly random group element. To do that, we need to alter the crs program so that it can evaluate on all inputs using only $v^{a^0}, v^{a^1}, \ldots, v^{a^{\ell-1}}$. We use the next Q hybrids $\{\mathsf{Game}_{B,y}\}_{y \in [Q]}$ to set this up.

In experiment $\mathsf{Game}_{B,y}$, the first y public keys are chosen differently. For the j^{th} public key ($j \leq y$), s_j is still an encryption of j. However, the verification key is chosen in non-binding mode. The challenger computes a pseudorandom string r'_j (using s_j and PRF key K_{C}), and uses r'_j for sampling sigk_j and vk_j. This sets the j^{th} public key, but the corresponding secret key will be chosen only if this key is corrupted. When this key is corrupted, the challenger uses the signing key sigk_j to compute a signature σ_j on a 'random-looking' message m_j. However, unlike in $\mathsf{Game}_{A,Q}$, this message is not a random string - instead, it is an obfuscated program P_j masked by a pseudorandom string. The first y public keys are now of $\mathsf{type} = 2$.

The crs program counts the number of public keys that are of $\mathsf{type} = 1$ or 2 (note - these are

[6] While it is immediate that the size of $\mathsf{EvalProg}$ depends on ℓ, it is not obvious why it also depends on c. This dependency arises because the program $\mathsf{EvalProg}$ is appropriately padded to be of the same size as $\mathsf{EvalProg}$-1, $\mathsf{EvalProg}$-2, $\mathsf{EvalProg}$-3 and $\mathsf{EvalProg}$-4. Some of these programs have c constants hardwired, hence the size of all these programs also depends on c.

the keys that were issued by the challenger). If all the public keys are either $\mathsf{type} = 1$ or 2, and the public key corresponding to the secret key is also $\mathsf{type} = 2$, then the crs program extracts the obfuscated program P from the message m (recall, since this is one of the first y keys, the message is an obfuscated program masked by a pseudorandom string). This program P takes as input the hash strings $\{u_j\}$ as input, computes the integers $\{t_j\}$ using $\mathsf{F}(\mathsf{K}_{\text{main}}, \cdot)$, and outputs $v^{a^\ell \cdot \prod_j t_j}$. Note that this is the same output that the crs program would've produced. In Section 6.2, we provide an overview of the indistinguishability argument.

This brings us to hybrid $\mathsf{Game}_{B,Q}$. In this hybrid, all the public keys are of $\mathsf{type} = 2$. Also, note that the crs program does not require v^{a^ℓ} because if all input public keys are of type 2, then the program uses the message to extract an obfuscated program. If any one of the keys is not of $\mathsf{type} = 2$, then the count cnt is less than ℓ, and therefore, the crs program can compute this using $v^{a^0}, v^{a^1}, \ldots, v^{a^{\ell-1}}$. However, note that the message m_j in the j^{th} secret key contains program P_j, and this program currently computes $v^{a^\ell \cdot \prod_j t_j}$. We will modify the outputs of these programs so that v^{a^ℓ} is not needed to compute the output. While we are altering the outputs of these programs (and not just the implementation), since the t_j integers are pseudorandom, the adversary does not detect this change. More details can be found in the proof overview given in Section 6.3.

Formal description of hybrid experiments: The first experiment $\mathsf{Game}_{\text{real}}$ corresponds to the adaptive security game. Next, we have Q hybrid games $\{\mathsf{Game}_{A,y}\}_{y\in[Q]}$. After $\mathsf{Game}_{A,Q}$, we have Q more hybrid experiments $\{\mathsf{Game}_{B,y}\}_{y\in[Q]}$. First, we show (in Section 6.1) that $\mathsf{Game}_{A,y}$ and $\mathsf{Game}_{A,y+1}$ are computationally indistinguishable (for any $y \in [Q-1]$). Next, we show a similar claim for $\mathsf{Game}_{B,y}$ and $\mathsf{Game}_{B,y+1}$ (in Section 6.2). Finally, (in Section 6.3), we show that any polynomial time adversary has negligible advantage in $\mathsf{Game}_{B,Q}$.

- $\mathsf{Game}_{\text{real}}$: In this experiment, the challenger first performs setup, then receives pre-challenge queries from the attacker. After these queries, the challenger receives the challenge set. It either sends the shared key corresponding to this challenge set, or sends a uniformly random string. The post-challenge queries are handled similar to the pre-challenge queries. The queries can be of the following types:

  - **Setup Phase** The challenger chooses hash key hk, PRF K_{main} and computes an obfuscation of $\mathsf{KGen}_{\mathsf{hk},\mathsf{K}_{\text{main}}}$.
  - **Pre/Post Challenge Queries** The challenger receives the following queries in the pre-challenge/post-challenge phase.
    * honest user registration queries - the challenger chooses s, m uniformly at random, computes $(\sigma, \mathsf{vk}) \leftarrow \mathsf{GenBind}_{sig}(1^\lambda, m)$. It sends $\mathsf{pk} = (s, \mathsf{vk})$ to the adversary, sets $\mathsf{sk} = (m, \sigma)$, and adds $(\mathsf{pk}, \mathsf{sk}, 0)$ to its records.
    * corruption queries (at most c such queries) - let $\mathsf{pk} = (s, \mathsf{vk})$ be the query, and sk the corresponding secret key. The challenger sends sk, and updates its records to indicate that pk is corrupted (that is, the corresponding record is updated to $(\mathsf{pk}, \mathsf{sk}, 1)$).
    * malicious user registration queries - the adversary sends the public key of the malicious user
    * shared key queries - adversary sends a set of public keys, together with an index corresponding to an uncorrupted user. The challenger computes a shared key using the secret key and sends the shared key.

– **Challenge Query** On receiving the set of ℓ public keys for the challenge, the challenger checks that none of these public keys are malicious or corrupted. After performing this check, the challenger either sends the shared key or a uniformly random string.

- $\mathsf{Game}_{A,y}$: Next, we define Q hybrid experiments $\mathsf{Game}_{A,y}$ for each $y \in [Q]$. There are two changes when we go from $\mathsf{Game}_{A,y}$ to $\mathsf{Game}_{A,y+1}$.

  First, in the $(y+1)^{\text{th}}$ honest user registration, the s is now a PDE encryption of $(y+1)$, (m, r) are pseudorandomly derived from s, and the verification key and signature are derived using $\mathsf{GenBind}_{sign}$ with m as the message, and r as the randomness.

  Second, the crs is also altered. Given a public key $\mathsf{pk} = (s, \mathsf{vk})$, it checks if the public key is of this special form. If so, it changes the final PRF output accordingly.

  – **Setup Phase**
    * The challenger chooses hash key hk, PRF K_{main}.
    * It chooses PRF key K_E, PPDE key K_{Det}.
    * It computes an obfuscation of $\mathsf{KGenAlt\text{-}1}_{\mathsf{hk},\mathsf{K}_{\text{main}},\mathsf{K}_{\text{Det}},\mathsf{K}_E,y}$, where $\mathsf{KGenAlt\text{-}1}$ is defined in Figure 11.
  – **Pre/Post Challenge Queries** The challenger receives the following queries in the pre-challenge/post-challenge phase.
    * honest user registration queries - The challenger receives Q honest user registration queries.
      For the j^{th} query, if $j \leq y$, the challenger does the following:
      · The challenger computes $s_j = \mathsf{DetEnc}(\mathsf{K}_{\text{Det}}, j)$.
      · The challenger computes $(m_j, r_j) = \mathsf{F}(\mathsf{K}_E, s_j)$
      · It computes $(\sigma_j, \mathsf{vk}_j) = \mathsf{GenBind}_{sig}(m_j; r_j)$

      If $j > y$, it chooses s_j, m_j uniformly at random, computes $(\sigma_j, \mathsf{vk}_j) \leftarrow \mathsf{GenBind}_{sig}(m_j)$. It sends $\mathsf{pk} = (s, \mathsf{vk})$ to the adversary, sets $\mathsf{sk} = (m, \sigma)$, and adds $(\mathsf{pk}, \mathsf{sk}, 0)$ to its records.
    * corruption queries (at most c such queries) - same as before
    * malicious user registration queries - same as before
    * shared key queries - same as before
  – **Challenge Query** same as before

- $\mathsf{Game}_{B,y}$: After the experiment $\mathsf{Game}_{A,Q}$, we have Q more hybrid experiments $\mathsf{Game}_{B,y}$ for each $y \in [Q]$. In the experiment $\mathsf{Game}_{B,y}$, the challenger alters the crs during setup, as well as the response to the first y honest user registration queries. In particular, the verification key is now generated in non-binding mode. If any of these y keys are corrupted, then the challenger sends a 'special' (message, signature) pair, where the message has a hidden obfuscated program.

  – **Setup Phase**
    * The challenger chooses hash key hk, PRF K_{main}.

Figure 11: The program $\mathsf{KGenAlt\text{-}1}_{y,\mathsf{hk},\mathsf{K}_{\mathrm{main}},\mathsf{K}_E,\mathsf{K}_{\mathrm{Det}},v,a}$.

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: $y, \mathsf{hk}, \mathsf{K}_{\mathrm{main}}, \mathsf{K}_E, \mathsf{K}_{\mathrm{Det}}, v, a$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $j \in [\ell]$, let $u_j = H(\mathsf{hk}, (s_j, \mathsf{vk}_j))$.
3. For each $j \in [\ell]$, let $t_j = \mathsf{F}(\mathsf{K}_{\mathrm{main}}, u_j)$.
4. For each $j \in [\ell]$ do the following:
   (a) Compute $d = \mathsf{Dec}(\mathsf{K}_{\mathrm{Det}}, s_j)$. Interpret d as a positive integer.
   (b) Compute $(m_j, r_j) = \mathsf{F}(\mathsf{K}_E, s_j)$.
   (c) If $(\sigma_{s_j}, \mathsf{vk}_{s_j}) = \mathsf{GenBind}_{sig}(m_j; r_j)$ and $\mathsf{vk}_{s_j} = \mathsf{vk}_i$ and $d \leq y$, $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. Compute $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and output v^t.

  * It chooses PRF key K_E, PPDE key $\mathsf{K}_{\mathrm{Det}}$, PRF keys $\mathsf{K}_{\mathrm{C}}, \mathsf{K}_{\mathrm{D}}$.
  * It chooses a string α, $g \leftarrow \mathbb{G}$, $a, \nu \leftarrow \mathbb{Z}_p$. It sets $v = g^\nu$.
  * It computes an obfuscation of $\mathsf{KGenAlt\text{-}2}_{y,\mathsf{hk},\mathsf{K}_{\mathrm{main}},\mathsf{K}_{\mathrm{Det}},\mathsf{K}_E,\mathsf{K}_{\mathrm{C}},\mathsf{K}_{\mathrm{D}}}$, where KGenAlt-2 is defined in Figure 12.

– **Pre/Post Challenge Queries** The challenger receives the following queries in the pre-challenge/post-challenge phase.

  * honest user registration queries - The challenger receives Q honest user registration queries.
    For the j^{th} query, if $j \leq y$, the challenger does the following:
    · The challenger computes $s_j = \mathsf{DetEnc}(\mathsf{K}_{\mathrm{Det}}, j)$.
    · The challenger computes $r_j = \mathsf{F}(\mathsf{K}_{\mathrm{C}}, s_j)$.
    · It computes $(\mathsf{sigk}_j, \mathsf{vk}_j) = \mathsf{Gen}_{sig}(1^\lambda; r_j)$.
    · The challenger sets $\mathsf{sk}_j = \ddagger\ddagger$ (indicating that the secret key is not yet set).
    If $j > y$, it sets $s_j = \mathsf{DetEnc}(\mathsf{K}_{\mathrm{Det}}, j)$, $(m_j, r_j) = \mathsf{F}(\mathsf{K}_E, s_j)$ and computes $(\sigma_j, \mathsf{vk}_j) \leftarrow \mathsf{GenBind}_{sig}(m_j; r_j)$.
    It sends $\mathsf{pk}_j = (s_j, \mathsf{vk}_j)$ to the adversary, sets $\mathsf{sk}_j = (m_j, \sigma_j)$, and adds $(\mathsf{pk}_j, \mathsf{sk}_j, 0)$ to its records.
  * corruption queries (at most c such queries) - On receiving a query for public key pk, the challenger checks if $\mathsf{pk} = \mathsf{pk}_j$ for some registered public key pk_j. If $j > y$, then the corresponding secret key is already set (during the honest user registration), and the challenger sends the corresponding secret key.
    If $\mathsf{pk} = \mathsf{pk}_j$ for some $j \leq y$, then the corresponding secret key is not yet set. It is chosen as follows:
    · Let $u^* = \mathsf{Hash}(\mathsf{hk}, \mathsf{pk}_j)$, $t^* = \mathsf{F}(\mathsf{K}_{\mathrm{main}}, u^*)$, $w = g^{\nu \cdot a^\ell}$, and let $\mathsf{EvalProg}_{\mathsf{K}_{\mathrm{main}},w,u^*,t^*}$ be the program defined in Figure 13. The challenger sets program $P_j = \mathsf{iO}(\mathsf{EvalProg}_{\mathsf{K}_{\mathrm{main}},w,u^*,t^*})$, message $m_j = P_j \oplus \mathsf{F}(K_D, (s_j, \mathsf{vk}_j)) \oplus \alpha$.

· It computes $\sigma_j = \mathsf{Sign}(\mathsf{sigk}_j, m_j)$, sends (m_j, σ_j).

· It sets $\mathsf{sk}_j = (m_j, \sigma_j)$, adds $(\mathsf{pk}_j, \mathsf{sk}_j, 1)$ to its records.

∗ malicious user registration queries - same as before

∗ shared key queries - same as before

– **Challenge Query** same as before

Figure 12: The program $\mathsf{KGenAlt\text{-}2}_{y,\mathsf{hk},\mathsf{K}_{\text{main}},\mathsf{K}_E,\mathsf{K}_{\text{C}},\mathsf{K}_{\text{D}},\mathsf{K}_{\text{Det}},v,a}$.

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: $y, \mathsf{hk}, \mathsf{K}_{\text{main}}, \mathsf{K}_E, \mathsf{K}_{\text{C}}, \mathsf{K}_{\text{D}}, \mathsf{K}_{\text{Det}}, v, a$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $k \in [\ell]$, let $u_k = H(\mathsf{hk}, (s_k, \mathsf{vk}_k))$.
3. For each $k \in [\ell]$, let $t_k = \mathsf{F}(\mathsf{K}_{\text{main}}, u_k)$.
4. Set $\mathsf{cnt} = 0$. For each $k \in [\ell]$ do the following:
   (a) Set $\mathsf{type}_k = 0$, compute $j_k = \mathsf{Dec}(\mathsf{K}_{\text{Det}}, s_k)$. Interpret j_k as a positive integer.
   (b) Compute $(m_k, r_k) = \mathsf{F}(\mathsf{K}_E, s_k)$ and $r'_k = \mathsf{F}(\mathsf{K}_{\text{C}}, s_k)$.
   (c) If $(\sigma_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{GenBind}_{sig}(m_k; r_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $y < j_k \leq Q$, then $\mathsf{type}_k = 1$. Else if $(\mathsf{sigk}_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{Gen}_{sig}(1^\lambda; r'_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $j_k \leq y$, then $\mathsf{type}_k = 2$.
   (d) If $\mathsf{type}_k = 1$ or $\mathsf{type}_k = 2$, then $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. If $\mathsf{cnt} == \ell$ and $\mathsf{type}_i == 2$
   – Compute $P = m_i \oplus \mathsf{F}(\mathsf{K}_{\text{D}}, (s_i, \mathsf{vk}_i)) \oplus \alpha$.
   – Output $P((u_j)_{j \neq i})$.

   Else compute $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and output v^t.

Figure 13: The program $\mathsf{EvalProg}_{\mathsf{K}_{\text{main}},w,u^*}$.

Inputs: $u_1, u_2, \ldots, u_{\ell-1}$
Constants: $\mathsf{K}_{\text{main}}, w, u^*, t^*$

1. For each $k \in [\ell - 1]$, let $t_k = \mathsf{F}(\mathsf{K}_{\text{main}}, u_k)$. Let $u_\ell = u^*$ and $t_\ell = t^*$.
2. Compute $t = \prod_k t_k$ and output w^t.

Analysis of Security Games

Let $\mathsf{adv}_{\mathcal{A},\text{real}}$ denote the advantage of adversary $\mathcal{A}$ in $\mathsf{Game}_{\text{real}}$, and $\mathsf{adv}_{\mathcal{A},i}$ the advantage in Game_i.

6.1 From $\mathsf{Game}_{A,y}$ to $\mathsf{Game}_{A,y+1}$

There are two main differences when we go from $\mathsf{Game}_{A,y}$ to $\mathsf{Game}_{A,y+1}$. In $\mathsf{Game}_{A,y}$, the first y honest user registrations are in 'special mode' (that is, the public key is of $\mathsf{type} = 1$). The crs (which is an obfuscated program) detects how many of the public keys are in special mode. If there are cnt public keys in the special mode, then the output is $v^{a^{\mathsf{cnt}} \cdot (\ldots)}$ instead of $v^{(\ldots)}$. As a result, when going from $\mathsf{Game}_{A,y}$ to $\mathsf{Game}_{A,y+1}$, there are two main changes: the $(y+1)^{\text{th}}$ honest user registration is in special mode, and the crs program is altered so that it can detect how many of the first $y+1$ honest public keys are present in the input to the program.

If the j^{th} honest user registration is in special mode, then the string s_j is a PPDE encryption of j (instead of being a uniformly random string). The string s_j is then used to sample m_j and randomness r_j using PRF key K_E, and (m_j, r_j) are used to sample σ_j and vk_j (using $\mathsf{GenBind}_{sig}$). To go from $\mathsf{Game}_{A,y}$ to $\mathsf{Game}_{A,y+1}$, we first puncture the deterministic encryption key at $y+1$, and use this punctured key in the crs program. Note that using a PPDE key punctured at $y+1$ does not alter the functionality since in $\mathsf{Game}_{A,y}$, the program does not increment the count of special mode public keys if s is an encryption of j and $j > y$. This allows us to switch s_{y+1} from a uniformly random string to an encryption of $y+1$. After this, we switch back to the unpunctured PPDE key. Next, we puncture the PRF key K_E at s_{y+1} and use this punctured PRF key in the crs program. Again, the functionality doesn't change if we use a punctured PRF key. This allows us to switch m_{y+1} and r_{y+1} from uniformly random strings, to being the output of $\mathsf{F}(\mathsf{K}_E, s_{y+1})$. Having made this change, the crs program again uses the unpunctured PRF key K_E.

Next, we make the hash key binding at $(s_{y+1}, \mathsf{vk}_{y+1})$, and puncture the PRF key K_{main} at $u_{y+1} = \mathsf{Hash}(\mathsf{hk}, \mathsf{pk}_{y+1})$. We hardwire the PRF output at u_{y+1}. Using the security of puncturable PRFs, we replace the hardwired value $t_{y+1} = \mathsf{F}(\mathsf{K}_{\text{main}}, u_{y+1})$ with $a \cdot \mathsf{F}(\mathsf{K}_{\text{main}}, u_{y+1})$. This is equivalent to incrementing cnt if the input is $(s_{y+1}, \mathsf{vk}_{y+1})$. Finally, by sampling the hash key in non-binding mode, and removing the $(s_{y+1}, \mathsf{vk}_{y+1})$ hardwiring, we use the iO security to switch the crs program, and therefore reach $\mathsf{Game}_{A,y+1}$.

Formal proof: We will show that $\mathsf{Game}_{A,y}$ and $\mathsf{Game}_{A,y+1}$ are computationally indistinguishable, via the following sequence of hybrid experiments.

- $\mathsf{Game}_{A,y,0}$: In this experiment, the PPDE key is punctured at $y+1$. However, the crs uses the unpunctured PPDE key. The puncturing algorithm outputs a punctured key and a string z_{y+1}, and z_{y+1} is used as s_{y+1}.
- $\mathsf{Game}_{A,y,1}$: In this experiment, the PPDE key is punctured at $y+1$. The crs also uses the punctured PPDE decrytion key.
- $\mathsf{Game}_{A,y,2}$: In this experiment, s_{y+1} is encryption of $y+1$ (instead of being a uniformly random string).
- $\mathsf{Game}_{A,y,3}$: In this experiment, the crs uses an unpunctured PPDE key.
- $\mathsf{Game}_{A,y,4}$: In this experiment, the PRF key K_E is punctured at s_{y+1}.
- $\mathsf{Game}_{A,y,5}$: In this experiment, the challenger uses K_E to compute m_{y+1} and r_{y+1}.
- $\mathsf{Game}_{A,y,6}$: In this experiment, the challenger does not puncture the PRF key K_E at s_{y+1}. The unpunctured key is used in the crs, as well as for responding to honest user registration queries.

- $\mathsf{Game}_{A,y,7}$: In this experiment, the hash key is in binding mode. The challenger computes s_{y+1} and vk_{y+1} during setup, and the hash key is binding at $(s_{y+1}, \mathsf{vk}_{y+1})$.
- $\mathsf{Game}_{A,y,8}$: In this experiment, PRF key K_{main} is punctured at u_{y+1} : the hash of $(s_{y+1}, \mathsf{vk}_{y+1})$. The PRF value at this input (t_{y+1}) is hardwired in the crs program.

Figure 14: The program $\mathsf{KGenAlt\text{-}1.1}_{y,\mathsf{hk},\mathsf{K}_{\text{m}}\{u^*\},t^*,\mathsf{K}_E,\mathsf{K}_{\text{Det}},v,a}$.

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: y, hk, $\mathsf{K}_{\text{m}}\{u^*\}, t^*$, $\mathsf{K}_E, \mathsf{K}_{\text{Det}}, v, a$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $j \in [\ell]$, let $u_j = H(\mathsf{hk}, (s_j, \mathsf{vk}_j))$.
3. For each $j \in [\ell]$, if $u_j = u^*$, $t_j = t^*$, else $t_j = \mathsf{F}(\mathsf{K}_{\text{m}}\{u^*\}, u_j)$.
4. For each $j \in [\ell]$ do the following:
   (a) Compute $d = \mathsf{Dec}(\mathsf{K}_{\text{Det}}, s_j)$. Interpret d as a positive integer.
   (b) Compute $(m_j, r_j) = \mathsf{F}(\mathsf{K}_E, s_j)$. If the PRF key is punctured at the input, then set $m_j = r_j = \perp$.
   (c) If $(\sigma_{s_j}, \mathsf{vk}_{s_j}) = \mathsf{GenBind}_{sig}(m_j; r_j)$ and $\mathsf{vk}_{s_j} = \mathsf{vk}_j$ and $d \leq y$, $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. Compute $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and output v^t.

- $\mathsf{Game}_{A,y,9}$: In this experiment, the value t_{y+1} (which is hardwired in the crs program) is a uniformly random element in $\mathbb{Z}_p$.
- $\mathsf{Game}_{A,y,10}$: In this experiment, the challenger alters the program as follows. The program has $s_{y+1}, \mathsf{vk}_{y+1}$ hardwired. If $(s_j, \mathsf{vk}_j) = (s_{y+1}, \mathsf{vk}_{y+1})$, then the program sets $t_j = a \cdot t^*$. Else it proceeds as before.
- $\mathsf{Game}_{A,y,11}$: In this experiment, the challenger sets t^* to be the PRF evaluation at u_{y+1}.
- $\mathsf{Game}_{A,y,12}$: In this experiment, the challenger uses unpunctured key K_{main} in the crs program, and t^* is computed in the program (instead of being hardwired).
- $\mathsf{Game}_{A,y,13}$: In this experiment, the challenger samples the hash key in the non-binding mode.
- $\mathsf{Game}_{A,y,14}$: This experiment corresponds to $\mathsf{Game}_{A,y+1}$.

Lemma 15. *For any PPT adversary $\mathcal{A}$, $\mathsf{adv}_{A,y} \approx_c \mathsf{adv}_{A,y+1}$.*

Proof. We will show that the intermediate hybrids are computationally indistinguishable.

Claim 2. *Assuming the correctness property 3 of PPDE, $\mathsf{adv}_{A,y} = \mathsf{adv}_{A,y,0}$.*

Figure 15: The program $\mathsf{KGenAlt\text{-}1.2}_{y,\mathsf{hk},s^*,\mathsf{vk}^*,\mathsf{K_m}\{u^*\},t^*,\mathsf{K}_E,\mathsf{K}_{\mathrm{Det}},v,a}$.

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: y, hk, $s^*, \mathsf{vk}^*, \mathsf{K_m}\{u^*\}, t^*$, $\mathsf{K}_E, \mathsf{K}_{\mathrm{Det}}, v, a$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $j \in [\ell]$, let $u_j = H(\mathsf{hk}, (s_j, \mathsf{vk}_j))$.
3. For each $j \in [\ell]$, if $s_j = s^*$, $\mathsf{vk}_j = \mathsf{vk}^*$ then $t_j = a \cdot t^*$, else $t_j = \mathsf{F}(\mathsf{K_m}\{u^*\}, u_j)$.
4. For each $j \in [\ell]$ do the following:
   (a) Compute $d = \mathsf{Dec}(\mathsf{K}_{\mathrm{Det}}, s_k)$. Interpret j as a positive integer.
   (b) Compute $(m_j, r_j) = \mathsf{F}(\mathsf{K}_E, s_j)$. If the PRF key is punctured at the input, then set $m_j = r_j = \perp$.
   (c) If $(\sigma_{s_j}, \mathsf{vk}_{s_j}) = \mathsf{GenBind}_{sig}(m_j; r_j)$ and $\mathsf{vk}_{s_j} = \mathsf{vk}_i$ and $d \leq y$, $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. Compute $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and output v^t.

Proof. Note that the adversary does not receive the punctured PPDE key (since the PPDE key is used in the crs). As a result, using the property of PPDE, for any key $\mathsf{K}_{\mathrm{Det}}$ and any message m, if $(\mathsf{K}_{\mathrm{Det},i+1}, y) \leftarrow \mathsf{Puncture}(\mathsf{K}_{\mathrm{Det}}, m)$, then y is perfectly indistinguishable from a uniformly random string. □

Claim 3. *Assuming* iO *is secure, and assuming the correctness property 2 of PPDE scheme, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ, $\mathsf{adv}_{A,y,0} - \mathsf{adv}_{A,y,1} \leq \mathsf{negl}(\lambda)$.

Proof. The main difference between the experiments $\mathsf{Game}_{A,y,0}$ and $\mathsf{Game}_{A,y,1}$ is that the adversary receives an obfuscation of $\mathsf{KGenAlt\text{-}1}_{\mathsf{hk},\mathsf{K}_{\mathrm{main}},\mathsf{K}_{\mathrm{Det},i+1},\mathsf{K}_E,y}$. We need to show that the two programs are functionally identical (and hence their obfuscations are computationally indistinguishable). In Step 4a, one program uses a punctured key, while the other one uses a non-punctured key. These two decryptions differ on only one input : the encryption of $y+1$. However, if s_k is an encryption of $y+1$, then in both programs, the count cnt is not incremented. Therefore, the two programs are functionally identical. □

Claim 4. *Assuming the security of PPDE scheme, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ, $|\mathsf{adv}_{A,y,1} - \mathsf{adv}_{A,y,2}| \leq \mathsf{negl}(\lambda)$.

Proof. The only difference in the two experiments is that s_{y+1} is uniformly random in one experiment, and is set to encryption of $y+1$ in another. Since the adversary only receives the PPDE key punctured at $y+1$, the encryption of $y+1$ is computationally indistinguishable from a uniformly random string. □

Claim 5. *Assuming* iO *is secure, and assuming the correctness property 2 of PPDE scheme, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ, $\mathsf{adv}_{A,y,2} - \mathsf{adv}_{A,y,3} \leq \mathsf{negl}(\lambda)$.

Figure 16: The program $\mathsf{KGenAlt\text{-}1.3}_{y,\mathsf{hk},s^*,\mathsf{vk}^*,\mathsf{K}_{\mathrm{main}},\mathsf{K}_E,\mathsf{K}_{\mathrm{Det}},v,a}$.

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: $y, \mathsf{hk}, s^*, \mathsf{vk}^*, \mathsf{K}_{\mathrm{main}}, \mathsf{K}_E, \mathsf{K}_{\mathrm{Det}}, v, a$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $j \in [\ell]$, let $u_j = H(\mathsf{hk}, (s_j, \mathsf{vk}_j))$.
3. For each $j \in [\ell]$, if $s_j = s^*$, $\mathsf{vk}_j = \mathsf{vk}^*$ then $t_j = a \cdot \mathsf{F}(\mathsf{K}_{\mathrm{main}}, u_j)$, else $t_j = \mathsf{F}(\mathsf{K}_{\mathrm{m}}\{u^*\}, u_j)$.
4. For each $j \in [\ell]$ do the following:
   (a) Compute $d = \mathsf{Dec}(\mathsf{K}_{\mathrm{Det}}, s_k)$. Interpret d as a positive integer.
   (b) Compute $(m_j, r_j) = \mathsf{F}(\mathsf{K}_E, s_j)$. If the PRF key is punctured at the input, then set $m_j = r_j = \perp$.
   (c) If $(\sigma_{s_j}, \mathsf{vk}_{s_j}) = \mathsf{GenBind}_{sig}(m_j; r_j)$ and $\mathsf{vk}_{s_j} = \mathsf{vk}_i$ and $d \leq y$, $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. Compute $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and output v^t.

Proof. This proof is identical to the proof of Claim 3. □

Claim 6. *Assuming* iO *is secure, and assuming the correctness of puncturable PRF scheme, for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that for all λ, $\mathsf{adv}_{A,y,3} - \mathsf{adv}_{A,y,4} \leq \mathsf{negl}(\lambda)$.*

Proof. The proof of this claim relies on the observation that the programs $P_1 \equiv \mathsf{KGenAlt\text{-}1}_{\mathsf{hk},\mathsf{K}_{\mathrm{main}},\mathsf{K}_{\mathrm{Det},i+1},\mathsf{K}_E,y}$ and $P_2 \equiv \mathsf{KGenAlt\text{-}1}_{\mathsf{hk},\mathsf{K}_{\mathrm{main}},\mathsf{K}_{\mathrm{Det},i+1},\mathsf{K}_E\{s_{y+1}\},y}$ are functionally identical. Note that the punctured key is also used for the honest user registration queries. However, for these queries, the PRF is never evaluated on input s_{y+1} (as m_{y+1} and r_{y+1} are chosen at random in both the experiments).

The only difference in the two programs is in Step 4b, where P_1 uses the unpunctured key, while P_2 uses the key punctued at s_{y+1}. These two evaluations only differ on the input s_{y+1}. However, if $s_k = s_{y+1}$ (which is encryption of $y+1$), then count cnt is not incremented in both programs (due to the $d \leq y$ condition).

□

Claim 7. *Assuming* F *is a secure puncturable PRF, for any PPT adversary A, there exists a negligible function* negl *such that for all λ, $\mathsf{adv}_{A,y,4} - \mathsf{adv}_{A,y,5} \leq \mathsf{negl}(\lambda)$.*

Proof. The only difference in the two experiments is that m_{y+1}, r_{y+1} are chosen uniformly at random in $\mathsf{Game}_{A,y,4}$, while they are computed using F on input s_{y+1} in $\mathsf{Game}_{A,y,5}$. The adversary receives only the PRF key punctured at s_{y+1}, and as a result, any adversary that can distinguish between $\mathsf{Game}_{A,y,4}$ and $\mathsf{Game}_{A,y,5}$ can be used to win the puncturable PRF security game against F. □

Claim 8. *Assuming* iO *is secure, and assuming the correctness of puncturable PRF scheme, for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that for all λ, $\mathsf{adv}_{A,y,5} - \mathsf{adv}_{A,y,6} \leq \mathsf{negl}(\lambda)$.*

Proof. This proof is identical to the proof of Claim 6. □

Claim 9. *Assuming the indistinguishability of normal and binding mode for the hash function, for any PPT adversary A, there exists a negligible function* negl *such that for all* λ, $\mathsf{adv}_{A,y,6} - \mathsf{adv}_{A,y,7} \leq \mathsf{negl}(\lambda)$.

Proof. The only difference in the two experiments is that hk is chosen in (non-binding) mode in $\mathsf{Game}_{A,y,6}$, and is chosen in binding mode in $\mathsf{Game}_{A,y,7}$. Note that $s_{y+1} = \mathsf{DetEnc}(\mathsf{K}_{\mathrm{Det}}, y+1)$, $(m_{y+1}, r_{y+1}) = \mathsf{F}(\mathsf{K}_E, s_{y+1})$ and $(\sigma_{y+1}, \mathsf{vk}_{y+1}) = \mathsf{GenBind}_{sig}(m_{y+1}; r_{y+1})$ can both be computed during setup. Therefore, using the indistinguishability of binding and non-binding mode for Hash, we can conclude that the two games are computationally indistinguishable. □

Claim 10. *Assuming* iO *is a secure indistinguishability obfuscator, and assuming correctness of* F*, for any PPT adversary A, there exists a negligible function* negl *such that for all* λ, $\mathsf{adv}_{A,y,7} - \mathsf{adv}_{A,y,8} \leq \mathsf{negl}(\lambda)$.

Proof. The main difference in the two experiments is the crs computation. In $\mathsf{Game}_{A,y,7}$, the crs is an obfuscation of $P_1 \equiv \mathsf{KGenAlt\text{-}1}_{\mathsf{hk},\mathsf{K}_{\mathrm{main}},\mathsf{K}_{\mathrm{Det},i+1},\mathsf{K}_E,y}$, while in $\mathsf{Game}_{A,y,8}$, it is an obfuscation of $P_2 = \mathsf{KGenAlt\text{-}1.1}_{y,\mathsf{hk},\mathsf{K}_{\mathrm{m}}\{u_{y+1}\},t_{y+1},\mathsf{K}_E,\mathsf{K}_{\mathrm{Det}},v,a}$ where $u_{y+1} = H(\mathsf{hk}, (s_{y+1}, \mathsf{vk}_{y+1}))$ and $t_{y+1} = \mathsf{F}(\mathsf{K}_{\mathrm{main}}, u_{y+1})$. The programs are functionally identical as the only input where $\mathsf{F}(\mathsf{K}_{\mathrm{main}}, \cdot)$ and $\mathsf{F}(\mathsf{K}_{\mathrm{m}}\{u_{y+1}\}, \cdot)$ differ is u_{y+1}. For this input, P_2 uses the hardwired constant t_{y+1} in Step 3. The remaining computation is identical in both the programs. □

Claim 11. *Assuming* F *is a secure puncturable PRF, for any PPT adversary A, there exists a negligible function* negl *such that for all* λ, $\mathsf{adv}_{A,y,8} - \mathsf{adv}_{A,y,9} \leq \mathsf{negl}(\lambda)$.

Proof. The only difference in the two experiments is the constant t^* that is hardwired in the crs program. In one case, t^* is equal to $\mathsf{F}(\mathsf{K}_{\mathrm{main}}, u_{y+1})$, while in the other case, it is uniformly random. The remaining experiment only uses $\mathsf{K}_{\mathrm{m}}\{u_{y+1}\}$ (that is, the key $\mathsf{K}_{\mathrm{main}}$ punctured at u_{y+1}). As a result, the adversary only receives $\mathsf{K}_{\mathrm{m}}\{u_{y+1}\}$, and hence, if an adversary can distinguish between $\mathsf{Game}_{A,y,8}$ and $\mathsf{Game}_{A,y,9}$, then this adversary can be used to break the puncturable PRF security of F. □

Claim 12. *Assuming* iO *is a secure indistinguishability obfuscator, and assuming the binding property of* $\mathsf{GenBind}_{Hash}$*, for any PPT adversary A, there exists a negligible function* negl *such that for all* λ, $\mathsf{adv}_{A,y,9} - \mathsf{adv}_{A,y,10} \leq \mathsf{negl}(\lambda)$.

Proof. The main difference between the two experiments is that the first one uses program $P_1 \equiv \mathsf{KGenAlt\text{-}1.1}_{\mathsf{hk},u_{y+1},\mathsf{K}_{\mathrm{m}}\{u_{y+1}\},t^*,\mathsf{K}_{\mathrm{Det}},\mathsf{K}_E,y,a}$, and the second one uses the program P_2, which is an obfuscation of $\mathsf{KGenAlt\text{-}1.2}_{\mathsf{hk},s_{y+1},\mathsf{vk}_{y+1},\mathsf{K}_{\mathrm{m}}\{u_{y+1}\},t^*,\mathsf{K}_{\mathrm{Det}},\mathsf{K}_E,y,a}$. Program P_1, in Step 3, sets $t_j = t^*$ if $H(s_j, \mathsf{vk}_j) = u_{y+1}$, while program P_2, in Step 3, sets $t_j = a \cdot t^*$ if $s_j = s_{y+1}$ and $\mathsf{vk}_j = \mathsf{vk}_{y+1}$. Using the binding property of H, we know that $H(s_j, \mathsf{vk}_j) = u_{y+1}$ if and only if $s_j = s_{y+1}$ and $\mathsf{vk}_j = \mathsf{vk}_{y+1}$ (since it is binding at $(s_{y+1}, \mathsf{vk}_{y+1})$. Secondly, since t^* is uniformly random, both t^* and $a \cdot t^*$ are identically distributed. Hence, the two programs are functionally identical, and therefore their obfuscations are indistinguishable.

□

Claim 13. *Assuming* F *is a secure puncturable PRF, for any PPT adversary A, there exists a negligible function* negl *such that for all* λ, $\mathsf{adv}_{A,y,10} - \mathsf{adv}_{A,y,11} \leq \mathsf{negl}(\lambda)$.

Proof. In both the experiments, the crs only contains the punctured PRF key $\mathsf{K}_{\mathrm{m}}\{u_{y+1}\}$. As a result, using the PRF security, the PRF evaluation at u_{y+1} is indistinguishable from a uniformly random string. □

Claim 14. *Assuming* iO *is a secure indistinguishability obfuscator, for any PPT adversary A, there exists a negligible function* negl *such that for all* λ, $\mathsf{adv}_{A,y,11} - \mathsf{adv}_{A,y,12} \leq \mathsf{negl}(\lambda)$.

Proof. The two programs $\mathsf{KGenAlt}$-1.2 and $\mathsf{KGenAlt}$-1.3 are functionally identical. In one case, the program has $t^* = \mathsf{F}(\mathsf{K}_{\mathrm{main}}, u_{y+1})$ hardwired, while in the other case, it computes t^* in the program. Since the programs are functionally identical, their obfuscations are indistinguishable. □

Claim 15. *Assuming the indistinguishability of normal and binding mode for the hash function, for any PPT adversary A, there exists a negligible function* negl *such that for all* λ, $\mathsf{adv}_{A,y,12} - \mathsf{adv}_{A,y,13} \leq \mathsf{negl}(\lambda)$.

Proof. The proof of this claim is similar to the proof of Claim 9. □

Claim 16. *Assuming* iO *is a secure indistinguishability obfuscator, for any PPT adversary A, there exists a negligible function* negl *such that for all* λ, $\mathsf{adv}_{A,y,13} - \mathsf{adv}_{A,y,14} \leq \mathsf{negl}(\lambda)$.

Proof. The programs $\mathsf{KGenAlt}$-1.3_y and $\mathsf{KGenAlt}$-1_{y+1} differ in two locations: in Step 3 and Step 4c.

Let s^* denote the encryption of $y+1$, and vk^* the verification key derived from s^* (as in $\mathsf{Game}_{A,y,14}$). We consider the following cases:

- There exists some input public key $\mathsf{pk}_j = (s^*, \mathsf{vk}^*)$ The program $\mathsf{KGenAlt}$-1.3_y sets $t_j = a \cdot \mathsf{F}(\mathsf{K}_{\mathrm{main}}, u_j)$, while the program $\mathsf{KGenAlt}$-1_{y+1} increments the value of cnt. As a result, the exponent of a in Step 5 is same in both programs. Therefore, the outputs are identical.
- For all j, $\mathsf{pk}_j \neq (s^*, \mathsf{vk}^*)$ Both programs have identical behaviour in Steps 1-3, since $\mathsf{pk}_j \neq (s^*, \mathsf{vk}^*)$ for all j.

  In Step 4c, the programs can potentially differ if the decryption of some s_j is equal to $y+1$, and the corresponding vk_j is equal to vk^*. However, since either $s_j \neq s^*$ or $\mathsf{vk}_j \neq \mathsf{vk}^*$, both programs have the same value of cnt before the last step. As a result, the outputs are same.

□

□

6.2 From $\mathsf{Game}_{B,y}$ to $\mathsf{Game}_{B,y+1}$

There are three main differences when we go from $\mathsf{Game}_{B,y}$ to $\mathsf{Game}_{B,y+1}$.

1. In $\mathsf{Game}_{B,y+1}$, when the challenger registers the $(y+1)^{\text{th}}$ honest user, the verification key is in non-binding mode (in $\mathsf{Game}_{B,y}$, the message m_{y+1} and randomness r_{y+1} are pseudorandomly derived from s_{y+1} (which is a PPDE encryption of $y+1$), and verification key is in binding mode, derived from m_{y+1} using randomness r_{y+1}). Having the verification key in non-binding mode allows the challenger to choose the message at a later point (that is, when this key is corrupted).

2. At a later point, if the $(y+1)^{\text{th}}$ key is corrupted, the challenger sets message m_{y+1} to be an obfuscated program, appropriately masked with a pseudorandom string. The challenger then signs the message (computing σ_{y+1}) and sends (m_{y+1}, σ_{y+1}) as the corrupted key.

3. The final difference in the two games is with regard to the crs program. In $\mathsf{Game}_{B,y}$, the crs program assigns a 'type' to each of the ℓ public keys. If the verification key is generated (using appropriate pseudorandomness) in binding mode, and if the decryption of s is greater than y, then the 'type' is set to be 1. If the verification key is generated in non-binding mode, and the decryption of s is at most y, then the 'type' is set to 2. In $\mathsf{Game}_{B,y+1}$, if the decryption of s is $y+1$ and it is in non-binding mode, then 'type' is set to 2, otherwise (if it is in binding mode) then 'type' remains 0.

Let s_{y+1} denote the encryption of $(y+1)$. In $\mathsf{Game}_{B,y}$, the verification key is generated by first generating (m_{y+1}, r_{y+1}) from s_{y+1} using the PRF key K_E, and then using the message and randomness to generate a verification key (together with a signature) in the binding mode. Therefore, as a first step, we puncture the key K_E, and use the punctured key in the crs program. The program has $(s_{y+1}, \mathsf{vk}_{y+1})$ hardwired (and the program's logic is suitably implemented to be functionally identical to the previous hybrid). With the key punctured, we can first switch m_{y+1}, r_{y+1} to be truly random strings. Next, through a sequence of hybrid experiments, we alter the condition in which the crs program uses the program extracted from the message. If all the ℓ input public keys are of type 1 or 2, and if the secret key corresponds to $(s_{y+1}, \mathsf{vk}_{y+1})$, then the crs program extracts program P from the message, and uses that to compute the final output. This is possible because the verification key is in binding mode, which means that if the signature is accepted, then the message must be the program P_{y+1} masked by a pseudorandom string. Using the fact that the output of program P_{y+1} is identical to the output of the crs program, it follows that this switch is indistinguishable.

Once we make the above alteration to the crs program, the verification key can now be in non-binding mode, and the message m_{y+1} does not need to be chosen during setup. Again, using the PRF security and the security of iO, we can alter the verification key to be derived pseudorandomly from s_{y+1}, and finally, we can remove the $(s_{y+1}, \mathsf{vk}_{y+1})$ hardwiring, and use the unpunctured PRF key K_E.

Below, we describe the hybrid experiments formally.

- $\mathsf{Game}_{B,y,1}$: In this experiment, the challenger computes $s_{y+1}, \mathsf{vk}_{y+1}$ during setup, and in the crs obfuscated program, it hardwires the 'type' information when $s_i = s_{y+1}$, $\mathsf{vk}_i = \mathsf{vk}_{y+1}$.

  - **Setup Phase**
    - The challenger chooses hash key hk, PRF K_{main}.
    - It chooses PRF key K_E, PPDE key K_{Det}, PRF keys $\mathsf{K}_{\text{C}}, \mathsf{K}_{\text{D}}$.
    - It chooses a string α, $g \leftarrow \mathbb{G}$, $a, \nu \leftarrow \mathbb{Z}_p$.
      It sets $v = g^{\nu}$, $w = g^{\nu \cdot a^{\ell}}$.
    - It computes $s_{y+1} = \mathsf{DetEnc}(\mathsf{K}_{\text{Det}}, y+1)$, $(m_{y+1}, r_{y+1}) = \mathsf{F}(\mathsf{K}_E, s_{y+1})$.
      Using m_{y+1}, r_{y+1} it computes $(\sigma_{y+1}, \mathsf{vk}_{y+1}) = \mathsf{GenBind}_{sig}(m_{y+1}; r_{y+1})$. These are used to answer queries related to the $(y+1)^{\text{th}}$ registration.
      It computes punctured key $\mathsf{K}_E\{s_{y+1}\} = \mathsf{Puncture}(\mathsf{K}_E, s_{y+1})$.

* It computes an obfuscation of $\mathsf{KGenAlt\text{-}2.1}_{y,\mathsf{hk},\mathsf{K}_{\mathrm{main}},\mathsf{K}_{\mathrm{Det}},\mathsf{K}_E\{s_{y+1}\},\mathsf{K}_{\mathrm{C}},\mathsf{K}_{\mathrm{D}},s_{y+1},\mathsf{vk}_{y+1}}$, where $\mathsf{KGenAlt\text{-}2.1}$ is defined in Figure 17. Note that this program has $s_{y+1}, \mathsf{vk}_{y+1}$ hard-wired.

– **Pre/Post Challenge Queries** The challenger receives the following queries in the pre-challenge/post-challenge phase.

* honest user registration queries - The challenger receives Q honest user registration queries.
  For the j^{th} query, if $j \leq y$, the challenger does the following:
  · The challenger computes $s_j = \mathsf{DetEnc}(\mathsf{K}_{\mathrm{Det}}, j)$.
  · The challenger computes $r_j = \mathsf{F}(\mathsf{K}_{\mathrm{C}}, s_j)$.
  · It computes $(\mathsf{sigk}_j, \mathsf{vk}_j) = \mathsf{Gen}_{sig}(1^\lambda; r_j)$.
  · The challenger sets $\mathsf{sk}_j = \ddagger\ddagger$ (indicating that the secret key is not yet set).
  If $j > y$, it sets $s_j = \mathsf{DetEnc}(\mathsf{K}_{\mathrm{Det}}, j), (m_j, r_j) = \mathsf{F}(\mathsf{K}_E, s_j)$ and computes $(\sigma_j, \mathsf{vk}_j) \leftarrow \mathsf{GenBind}_{sig}(m_j; r_j)$.
  It sends $\mathsf{pk}_j = (s_j, \mathsf{vk}_j)$ to the adversary, sets $\mathsf{sk}_j = (m_j, \sigma_j)$, and adds $(\mathsf{pk}_j, \mathsf{sk}_j, 0)$ to its records.
* corruption queries (at most c such queries) - On receiving a query for public key pk, the challenger checks if $\mathsf{pk} = \mathsf{pk}_j$ for some registered public key pk_j. If $j > y$, then the corresponding secret key is already set (during the honest user registration), and the challenger sends the corresponding secret key.
  If $\mathsf{pk} = \mathsf{pk}_j$ for some $j \leq y$, then the corresponding secret key is not yet set. It is chosen as follows:
  · Let $u^* = \mathsf{Hash}(\mathsf{hk}, \mathsf{pk}_j)$, $t^* = \mathsf{F}(\mathsf{K}_{\mathrm{main}}, u^*)$, $w = g^{\nu \cdot a^\ell}$, and let $\mathsf{EvalProg}_{\mathsf{K}_{\mathrm{main}},w,u^*,t^*}$ be the program defined in Figure 21. The challenger sets program $P_j = \mathsf{iO}(\mathsf{EvalProg}_{\mathsf{K}_{\mathrm{main}},w,u^*,t^*})$, message $m_j = P_j \oplus \mathsf{F}(K_D, s_j) \oplus \alpha$.
  · It computes $\sigma_j = \mathsf{Sign}(\mathsf{sigk}_j, m_j)$, sends (m_j, σ_j).
  · It sets $\mathsf{sk}_j = (m_j, \sigma_j)$, adds $(\mathsf{pk}_j, \mathsf{sk}_j, 1)$ to its records.
* malicious user registration queries - same as before
* shared key queries - same as before

– **Challenge Query** same as before

- $\mathsf{Game}_{B,y,2}$: In this experiment, the challenger switches m_{y+1} and r_{y+1} to be uniformly random strings.
- $\mathsf{Game}_{B,y,3}$: In this experiment, the challenger punctures the PRF key K_{D} at s_{y+1} and uses this punctured key in $\mathsf{KGenAlt\text{-}2.1}$, as well as for responding to corruption queries.
- $\mathsf{Game}_{B,y,4}$: In this experiment, the challenger chooses m_{y+1} uniformly at random, and sets $\alpha = P_{y+1} \oplus m_{y+1} \oplus \mathsf{F}(\mathsf{K}_{\mathrm{D}}, s_{y+1})$ (note that P_{y+1} depends on the verification key vk_{y+1}, which is derived from m_{y+1} in binding mode).
- $\mathsf{Game}_{B,y,5}$: In this experiment, the challenger uses the unpunctured PRF key K_{D} in $\mathsf{KGenAlt\text{-}2.1}$.
- $\mathsf{Game}_{B,y,6}$: In this experiment, the challenger uses $\mathsf{KGenAlt\text{-}2.2}$ (defined in Figure 18) for computing the crs. The constants hardwired in the program are same as in previous hybrid.

Figure 17: The program $\mathsf{KGenAlt\text{-}2.1}_{y,\mathsf{hk},\mathsf{K}_{\text{main}},\mathsf{K}_E\{s_{y+1}\},\mathsf{K}_{\text{C}},\mathsf{K}_{\text{D}},\mathsf{K}_{\text{Det}},v,a,\alpha,s^*,\mathsf{vk}^*}$

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: $y, \mathsf{hk}, \mathsf{K}_{\text{main}}$, $\mathsf{K}_E\{s^*\}$, $\mathsf{K}_{\text{C}}, \mathsf{K}_{\text{D}}, \mathsf{K}_{\text{Det}}, v, a$, α, s^*, vk^*

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $k \in [\ell]$, let $u_k = H(\mathsf{hk}, (s_k, \mathsf{vk}_k))$.
3. For each $k \in [\ell]$, let $t_k = \mathsf{F}(\mathsf{K}_{\text{main}}, u_k)$.
4. Set $\mathsf{cnt} = 0$. For each $k \in [\ell]$ do the following:
   (a) Set $\mathsf{type}_k = 0$, compute $j_k = \mathsf{Dec}(\mathsf{K}_{\text{Det}}, s_k)$. Interpret j_k as a positive integer.
   (b) Compute $(m_k, r_k) = \mathsf{F}(\mathsf{K}_E\{s^*\}, s_k)$ and $r'_k = \mathsf{F}(\mathsf{K}_{\text{C}}, s_k)$. If punctured key K_E (resp. K_{C}) is used, then set $m_k = r_k = \perp$ (resp. set $r'_k = \perp$).
   (c) If $(s_k, \mathsf{vk}_k) == (s^*, \mathsf{vk}^*)$ then $\mathsf{cnt} = \mathsf{cnt} + 1$.
   Continue to next iteration (go to step (a)).
   Else if $(\sigma_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{GenBind}_{sig}(m_k; r_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $y < j_k \leq Q$, then $\mathsf{type}_k = 1$.
   Else if $(\mathsf{sigk}_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{Gen}_{sig}(1^\lambda; r'_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $j_k \leq y$, then $\mathsf{type}_k = 2$.
   (d) If $\mathsf{type}_k = 1$ or $\mathsf{type}_k = 2$, then $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. If $\mathsf{type}_i == 2$ and $\mathsf{cnt} == \ell$
   - Compute $P = m_i \oplus \mathsf{F}(\mathsf{K}_{\text{D}}, s_i) \oplus \alpha$.
   - Output $P((u_j)_{j \neq i})$.

   Else compute $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and output v^t.

- $\mathsf{Game}_{B,y,7}$: In this experiment, the challenger samples $(\mathsf{sk}^*, \mathsf{vk}^*)$ using Gen_{sig} and sets $\mathsf{vk}_{y+1} = \mathsf{vk}^*$. It computes m_{y+1} as in the previous experiment. When the $(y+1)^{\text{th}}$ public key is corrupted, the challenger computes a signature σ_{y+1} on m_{y+1} using sk^* and sends (m_{y+1}, σ_{y+1}) as the corrupted key.
- $\mathsf{Game}_{B,y,8}$: In this experiment, the challenger chooses α uniformly at random, and sets m_{y+1} as $P_{y+1} \oplus \alpha \oplus \mathsf{F}(\mathsf{K}_{\text{D}}, s_{y+1})$. Note that, as in the previous experiment, vk_{y+1} does not depend on m_{y+1} (since it is in non-binding mode), and therefore P_{y+1} does not depend on m_{y+1}.
- $\mathsf{Game}_{B,y,9}$: In this experiment, the challenger punctures K_{C} at s^*, and uses this punctured PRF key in $\mathsf{KGenAlt\text{-}2.2}$.
- $\mathsf{Game}_{B,y,10}$: In this experiment, the challenger computes $(\mathsf{sk}_{y+1}, \mathsf{vk}_{y+1})$ using Gen_{sign}, but with a pseudorandom string derived from s_{y+1}. It computes s_{y+1} (as in the previous experiment). Then it sets $r' = \mathsf{F}(\mathsf{K}_{\text{C}}, s_{y+1})$, $(\mathsf{sk}_{y+1}, \mathsf{vk}_{y+1}) = \mathsf{Gen}_{sign}(1^\lambda; r')$. The rest of the experiment is same as the previous experiment.

Figure 18: The program $\mathsf{KGenAlt\text{-}2.2}_{y,\mathsf{hk},\mathsf{K}_{\text{main}},\mathsf{K}_E\{s_{y+1}\},\mathsf{K}_{\text{C}},\mathsf{K}_{\text{D}},\mathsf{K}_{\text{Det}},v,a,\alpha,s^*,\mathsf{vk}^*}$

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: $y, \mathsf{hk}, \mathsf{K}_{\text{main}}, \mathsf{K}_E\{s^*\}, \mathsf{K}_{\text{C}}, \mathsf{K}_{\text{D}}, \mathsf{K}_{\text{Det}}, v, a, \alpha, s^*, \mathsf{vk}^*$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $k \in [\ell]$, let $u_k = H(\mathsf{hk}, (s_k, \mathsf{vk}_k))$.
3. For each $k \in [\ell]$, let $t_k = \mathsf{F}(\mathsf{K}_{\text{main}}, u_k)$.
4. Set $\mathsf{cnt} = 0$. For each $k \in [\ell]$ do the following:
   (a) Set $\mathsf{type}_k = 0$, compute $j_k = \mathsf{Dec}(\mathsf{K}_{\text{Det}}, s_k)$. Interpret j_k as a positive integer.
   (b) Compute $(m_k, r_k) = \mathsf{F}(\mathsf{K}_E\{s^*\}, s_k)$ and $r'_k = \mathsf{F}(\mathsf{K}_{\text{C}}, s_k)$. If punctured key K_E (resp. K_{C}) is used, then set $m_k = r_k = \perp$ (resp. set $r'_k = \perp$).
   (c) If $(s_k, \mathsf{vk}_k) == (s^*, \mathsf{vk}^*)$ then $\mathsf{cnt} = \mathsf{cnt} + 1$.
   Continue to next iteration (go to step (a)).
   Else if $(\sigma_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{GenBind}_{sig}(m_k; r_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $y < j_k \leq Q$, then $\mathsf{type}_k = 1$.
   Else if $(\mathsf{sigk}_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{Gen}_{sig}(1^\lambda; r'_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $j_k \leq y$, then $\mathsf{type}_k = 2$.
   (d) If $\mathsf{type}_k = 1$ or $\mathsf{type}_k = 2$, then $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. If ($\mathsf{type}_i == 2$ or $(s_i, \mathsf{vk}_i) == (s^*, \mathsf{vk}^*)$) and $\mathsf{cnt} == \ell$
   (a) Compute $P = m_i \oplus \mathsf{F}(\mathsf{K}_{\text{D}}, s_i) \oplus \alpha$.
   (b) Output $P((u_j)_{j \neq i})$.

   Else compute $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and output v^t.

- $\mathsf{Game}_{B,y,11}$: In this experiment, the challenger uses the unpunctured key K_{C} in $\mathsf{KGenAlt\text{-}2.2}$.
- $\mathsf{Game}_{B,y,12}$: In this experiment, the challenger uses $\mathsf{KGenAlt\text{-}2.3}$ for computing the crs. This program is similar to $\mathsf{KGenAlt\text{-}2.2}$ (and the constants hardwired are same as in previous experiment), except in Step 4c, where j_k must be greater than $y + 1$ for $\mathsf{type}_k = 1$. The program is described in Figure 19.

Analysis Let $\mathsf{adv}_{B,y,j}$ denote the advantage of an adversary $\mathcal{A}$ in $\mathsf{Game}_{B,y,j}$.

Claim 17. *Assuming* iO *is a secure indistinguishability obfuscator, and assuming the correctness of* F*, for any adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ*,* $|\mathsf{adv}_{B,y} - \mathsf{adv}_{B,y,1}| \leq \mathsf{negl}(\lambda)$.

Proof. The only difference in the two hybrids is that $\mathsf{Game}_{B,y}$ uses $\mathsf{KGenAlt\text{-}2}$, while $\mathsf{Game}_{B,y,1}$ uses $\mathsf{KGenAlt\text{-}2.1}$. The only difference in the two programs is that $\mathsf{KGenAlt\text{-}2.1}$ uses a punctured PRF key K_E, and for tuple (s^*, vk^*), it increments cnt and moves to the next iteration. Note that the programs take ℓ pairs as input, $\mathsf{Game}_{B,y,1}$ sets s^* to be PPDE encryption of $y + 1$, computes $(m^*, r^*) = \mathsf{F}(\mathsf{K}_E, s^*)$ and $(\sigma^*, \mathsf{vk}^*) = \mathsf{GenBind}_{sig}(m^*; r^*)$. We will consider the following cases for

Figure 19: The program $\mathsf{KGenAlt\text{-}2.3}_{y,\mathsf{hk},\mathsf{K}_{\text{main}},\mathsf{K}_E\{s_{y+1}\},\mathsf{K}_\mathrm{C},\mathsf{K}_\mathrm{D},\mathsf{K}_{\text{Det}},v,a,\alpha,s^*,\mathsf{vk}^*}$

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: $y, \mathsf{hk}, \mathsf{K}_{\text{main}}, \mathsf{K}_E\{s^*\}, \mathsf{K}_\mathrm{C}, \mathsf{K}_\mathrm{D}, \mathsf{K}_{\text{Det}}, v, a, \alpha, s^*, \mathsf{vk}^*$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $k \in [\ell]$, let $u_k = H(\mathsf{hk}, (s_k, \mathsf{vk}_k))$.
3. For each $k \in [\ell]$, let $t_k = \mathsf{F}(\mathsf{K}_{\text{main}}, u_k)$.
4. Set $\mathsf{cnt} = 0$. For each $k \in [\ell]$ do the following:
   (a) Set $\mathsf{type}_k = 0$, compute $j_k = \mathsf{Dec}(\mathsf{K}_{\text{Det}}, s_k)$. Interpret j_k as a positive integer.
   (b) Compute $(m_k, r_k) = \mathsf{F}(\mathsf{K}_E\{s^*\}, s_k)$ and $r'_k = \mathsf{F}(\mathsf{K}_\mathrm{C}, s_k)$. If punctured key K_E (resp. K_C) is used, then set $m_k = r_k = \perp$ (resp. set $r'_k = \perp$).
   (c) If $(s_k, \mathsf{vk}_k) == (s^*, \mathsf{vk}^*)$ then $\mathsf{cnt} = \mathsf{cnt} + 1$.
   Continue to next iteration (go to step (a)).
   Else if $(\sigma_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{GenBind}_{sig}(m_k; r_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $y + 1 < j_k \leq Q$, then $\mathsf{type}_k = 1$.
   Else if $(\mathsf{sigk}_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{Gen}_{sig}(1^\lambda; r'_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $j_k \leq y$, then $\mathsf{type}_k = 2$.
   (d) If $\mathsf{type}_k = 1$ or $\mathsf{type}_k = 2$, then $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. If ($\mathsf{type}_i == 2$ or $(s_i, \mathsf{vk}_i) == (s^*, \mathsf{vk}^*)$) and $\mathsf{cnt} == \ell$
   (a) Compute $P = m_i \oplus \mathsf{F}(\mathsf{K}_\mathrm{D}, s_i) \oplus \alpha$.
   (b) Output $P((u_j)_{j \neq i})$.

   Else compute $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and output v^t.

each input pair, and argue that either both programs increment cnt, or neither does. Secondly, the final output of the two programs remains the same.

- the input tuple doesn't contains s^*: in this case, using the correctness of PRF evaluation, we can argue that the programs are functionally identical.
- the tuples is $(s_k, \mathsf{vk}_k) = (s^*, \tilde{vk})$ and $\tilde{vk} \neq \mathsf{vk}^*$: in this case, KGenAlt-2 doesn't increment cnt, and $\mathsf{type}_k = 0$. In the program KGenAlt-2.1, the behaviour is similar; $m_k = r_k = \perp$, and since $\tilde{\mathsf{vk}} \neq \mathsf{vk}^*$, cnt is not incremented and $\mathsf{type}_k = 0$.
- the tuple is (s^*, vk^*): in this case, KGenAlt-2 sets $\mathsf{type}_k = 1$ and increments cnt. The program KGenAlt-2.1 increments cnt but sets $\mathsf{type}_k = 0$. However, note that the two programs don't differentiate based on whether type_k is 0 or 1.

As a result, the two programs are functionally identical, and therefore the obfuscations are indistinguishable. □

Claim 18. *Assuming* F *is a secure puncturable PRF, for any adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ*,* $|\mathsf{adv}_{B,y,1} - \mathsf{adv}_{B,y,2}| \leq \mathsf{negl}(\lambda)$*.*

Proof. This follows from the definition of the two hybrid experiments. The PRF key K_E, punctured at s_{y+1}, is used in the two experiments. As a result, the PRF evaluation at s_{y+1} is indistinguishable from a truly random pair (m_{y+1}, r_{y+1}). □

Claim 19. *Assuming* iO *is a secure indistinguishability obfuscator, and assuming the correctness of* F*, for any adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ*,* $|\mathsf{adv}_{B,y,2} - \mathsf{adv}_{B,y,3}| \leq \mathsf{negl}(\lambda)$*.*

Proof. The only difference in the two experiments is that one uses PRF key K_{D} punctured at s^* (where s^* is the encryption of $y+1$.

The two programs can potentially differ on inputs where $(s_i, \mathsf{vk}_i) = (s^*, \cdots)$. However, note that if $s_i = s^*$, then type_i cannot be 2 (since s^* is an encryption of $y+1$). As a result, computation using $\mathsf{K}_{\mathrm{D}}/\mathsf{K}_{\mathrm{D}}\{s^*\}$ is not executed, and hence the two programs are functionally identical.

□

Claim 20. *Assuming* F *is a secure puncturable PRF, for any adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ*,* $|\mathsf{adv}_{B,y,3} - \mathsf{adv}_{B,y,4}| \leq \mathsf{negl}(\lambda)$*.*

Proof. In both the hybrids, the PRF key K_{D} is punctured at s_{y+1}, and as a result, the PRF evaluation $\mathsf{F}(\mathsf{K}_{\mathrm{D}}, s_{y+1})$ is indistinguishable from a uniformly random string. The only difference in the two hybrids is that α is uniformly random in one case (which is same as 'uniformly random' $\oplus m_{y+1} \oplus P_{y+1}$), and it is equal to $\mathsf{F}(\mathsf{K}_{\mathrm{D}}, s_{y+1}) \oplus m_{y+1} \oplus P_{y+1}$ in the other. Hence, using puncturable PRF security, we can argue that the two hybrids are indistinguishable. □

Claim 21. *Assuming* iO *is a secure indistinguishability obfuscator, and assuming the correctness of* F*, for any adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ*,* $|\mathsf{adv}_{B,y,4} - \mathsf{adv}_{B,y,5}| \leq \mathsf{negl}(\lambda)$*.*

Proof. The proof is identical to the proof of Claim 19. □

Claim 22. *Assuming* iO *is a secure indistinguishability obfuscator, and assuming the binding property of the signature scheme, for any adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ*,* $|\mathsf{adv}_{B,y,5} - \mathsf{adv}_{B,y,6}| \leq \mathsf{negl}(\lambda)$*.*

Proof. The challenger uses KGenAlt-2.1 in $\mathsf{Game}_{B,y,5}$, and uses KGenAlt-2.2 in $\mathsf{Game}_{B,y,6}$. The only difference in these two programs is in Step 5. The program KGenAlt-2.2 uses m_i if $(s_i, \mathsf{vk}_i) == (s^*, \mathsf{vk}^*)$ and $\mathsf{cnt} == \ell$, while program KGenAlt-2.1 outputs $v^{a^\ell \cdot \prod_k t_k}$ in this case. Hence it suffices to focus on inputs where the signature σ_i verifies in Step 1, $(s_i, \mathsf{vk}_i) == (s^*, \mathsf{vk}^*)$ and $\mathsf{cnt} == \ell$. Note that s^* is encryption of $y+1$, and vk^* is binding at $m^* = P_{y+1} \oplus F(\mathsf{K}_{\mathrm{D}}, s^*) \oplus \alpha$.

If the execution reaches Step 5, then the verification using $\mathsf{vk}_i = \mathsf{vk}^*$ passes. Using the binding property of the signature scheme, it follows that m_i must be m^*. Since $s_i = s^*$, the program computed in Step 5a must be $m^* \oplus \mathsf{F}(\mathsf{K}_{\mathrm{D}}, s^*) \oplus \alpha = P_{y+1}$.

Finally, since $\mathsf{cnt} == \ell$, the output of $P_{y+1}(\{u_j\}_{j \neq i})$ is exactly $v^{a^\ell \cdot \prod_k t_k}$, and therefore the outputs of KGenAlt-2.1 and KGenAlt-2.2 are identical. □

Claim 23. *Assuming the mode-indistinguishability of the signature scheme, for any PPT adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ, $|\mathsf{adv}_{B,y,6} - \mathsf{adv}_{B,y,7}| \leq \mathsf{negl}(\lambda)$.

Proof. Note that in $\mathsf{Game}_{B,y,6}$, the signature keys are generated in binding mode using $m_{y+1} = P_{y+1} \oplus \alpha \oplus \mathsf{F}(\mathsf{K}_\mathrm{D}, s_{y+1})$. In $\mathsf{Game}_{B,y,7}$, the challenger chooses $(\mathsf{sk}_{y+1}, \mathsf{vk}_{y+1})$ using Gen_{sig}, sets m_{y+1} as in $\mathsf{Game}_{B,y,6}$, and computes a signature on m_{y+1} using sk_{y+1}. Indistinguishability of the two experiments follows from the mode-indistinguishability of the signature scheme.

Suppose there exists a polynomial time adversary $\mathcal{A}$ that can distinguish between $\mathsf{Game}_{B,y,6}$ and $\mathsf{Game}_{B,y,7}$. Then we can use $\mathcal{A}$ to break the mode-indistinguishability of the signature scheme. The reduction algorithm computes $s_{y+1} = \mathsf{DetEnc}(\mathsf{K}_{\mathrm{Det}}, y+1)$, sets $m_{y+1} = P_{y+1} \oplus \alpha \oplus \mathsf{F}(\mathsf{K}_\mathrm{D}, s_{y+1})$ and sends m_{y+1} to the challenger. It receives a verification key vk_{y+1} and a signature σ_{y+1}. The reduction algorithm chooses the remaining components for interacting with $\mathcal{A}$. In the setup phase, the reduction sends $s_{y+1}, \mathsf{vk}_{y+1}$ (along with other components). If the $(y+1)^{\text{th}}$ key is corrupted, the reduction algorithm uses m_{y+1} and the signature σ_{y+1}. Note that the signing key is not needed anywhere else in the game(s).

Therefore, an adversary with distinguishable advantage in $\mathsf{Game}_{B,y,6}$ and $\mathsf{Game}_{B,y,7}$ can be used to break the mode-indistinguishability of the signature scheme. □

Claim 24. *For any adversary $\mathcal{A}$,* $\mathsf{adv}_{B,y,7} = \mathsf{adv}_{B,y,8}$.

Proof. The only difference in the two experiments is the choice of m_{y+1} and α. In $\mathsf{Game}_{B,y,7}$, m_{y+1} is chosen uniformly at random and α is set to $P_{y+1} \oplus m_{y+1} \oplus \mathsf{F}(\mathsf{K}_\mathrm{D}, s_{y+1})$, while in $\mathsf{Game}_{B,y,8}$, α is chosen uniformly at random, and m_{y+1} is set to $P_{y+1}\alpha \oplus \mathsf{F}(\mathsf{K}_\mathrm{D}, s_{y+1})$. In both experiments, the program P_{y+1} does not have any constants that are dependent on m_{y+1} or α, and as a result, the pair (m_{y+1}, α) are distributed identically in both experiments. □

Claim 25. *Assuming* iO *is a secure indistinguishability obfuscator, and assuming the correctness of* F*, for any adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ, $|\mathsf{adv}_{B,y,8} - \mathsf{adv}_{B,y,9}| \leq \mathsf{negl}(\lambda)$.

Proof. The only difference in the two experiments is that $\mathsf{Game}_{B,y,7}$ uses an unpunctured PRF key K_C in $\mathsf{KGenAlt}$-2.1, and $\mathsf{Game}_{B,y,8}$ uses $\mathsf{K}_\mathrm{C}\{s^*\}$ punctured at s^* (which is encryption of $y+1$). As in the proof of Claim 17, we will consider different cases for the (s, vk) input pairs, and compare the internal state of both programs just before before Step 5. Note that s^* is the encryption of $(y+1)$, r' is chosen uniformly at random and $(\mathsf{sk}^*, \mathsf{vk}^*) = \mathsf{Gen}_{sig}(1^\lambda; r')$.

- the input tuple does not contain s^*: in this case, both programs have identical behaviour (using the correctness of PRF evaluation on non-punctured points)
- the input tuple $(s_k, \mathsf{vk}_k) = (s^*, \tilde{vk})$ and $\tilde{vk} \neq \mathsf{vk}^*$: Note that both programs set $m_k = r_k = \perp$. As a result, type_k cannot be 1 in both cases. Similarly, type_k cannot be 2 in both programs. This is because in $\mathsf{Game}_{B,y,7}$ (where K_C is used), the check "$j_k \leq y$" is violated in Step 4c, while in $\mathsf{Game}_{B,y,8}$, since K_C is punctured, the program sets $r'_k = \perp$.
- $(s_k, \mathsf{vk}_k) = (s^*, \mathsf{vk}^*)$: in this case, both programs increment cnt and set $\mathsf{type}_k = 0$.

Using the above cases, it follows that the programs are functionally identical, and hence their obfuscations are computationally indistinguishable. □

Claim 26. *Assuming* F *is a secure puncturable PRF, for any adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ*,* $|\mathsf{adv}_{B,y,9} - \mathsf{adv}_{B,y,10}| \leq \mathsf{negl}(\lambda)$*.*

Proof. The key K_{C} is punctured at s_{y+1}, hence $\mathsf{F}(\mathsf{K}_{\mathrm{C}}, s_{y+1})$ is indistinguishable from a random string. In $\mathsf{Game}_{B,y,9}$, a random string is used to sample vk_{y+1}, while in $\mathsf{Game}_{B,y,10}$, $\mathsf{F}(\mathsf{K}_{\mathrm{C}}, s_{y+1})$ is used. These two distributions are computationally indistinguishable. □

Claim 27. *Assuming* iO *is a secure indistinguishability obfuscator, and assuming the correctness of* F*, for any adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ*,* $|\mathsf{adv}_{B,y,10} - \mathsf{adv}_{B,y,11}| \leq \mathsf{negl}(\lambda)$*.*

Proof. This is identical to the proof of Claim 25.

□

Claim 28. *Assuming* iO *is a secure indistinguishability obfuscator, for any adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ*,* $|\mathsf{adv}_{B,y,11} - \mathsf{adv}_{B,y,12}| \leq \mathsf{negl}(\lambda)$*.*

Proof. Similar to the previous proofs, we will show that for each pair (s_k, vk_k), the corresponding value of type_k is identical before Step 5. Note that the only difference in the two programs is the additional check $j_k > y + 1$ when checking for $\mathsf{type}_k = 1$ in $\mathsf{KGenAlt}$-2.3. Therefore, the value of type_k can differ in the two programs only if s_k is an encryption of $y + 1$. If s_k is an encryption of $y + 1$, then $s_k = s^*$, and if $s_k = s^*$, then both programs set $m_k = r_k = \perp$. As a result, the value of type_k is 0 (the value of cnt may be incremented, based on the value of vk_k, but this behaviour is same in both programs). □

Claim 29. *Assuming* iO *is a secure indistinguishability obfuscator, and assuming the correctness of* F*, for any adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* λ*,* $|\mathsf{adv}_{B,y,12} - \mathsf{adv}_{B,y+1}| \leq \mathsf{negl}(\lambda)$*.*

Proof. First, note that for any $(s_k, \mathsf{vk}_k) \neq (s^*, \mathsf{vk}^*)$, the value of type_k is same in both programs. If $s_k \neq s^*$, then both programs compute m_k, r_k and r'_k correctly. The check for $\mathsf{type}_k = 1$ is identical in both programs. The check for $\mathsf{type}_k = 2$ is different, since $\mathsf{KGenAlt}$-2.3 checks if $j_k \leq y$, while $\mathsf{KGenAlt}$-2 checks if $j_k \leq y + 1$. However, since $s_k \neq s^*$, the value of type_k is same in both programs. If $s_k == s^*$ but $\mathsf{vk}_k \neq vk^*$, then $\mathsf{type}_k \neq 2$. Since the check for $\mathsf{type}_k = 1$ is same in both programs, either both set $\mathsf{type}_k = 1$, or both set it to 0.

If $(s_k, \mathsf{vk}_k) == (s^*, \mathsf{vk}^*)$, then program $\mathsf{KGenAlt}$-2.3 sets $\mathsf{type}_k = 0$ and increments cnt, while $\mathsf{KGenAlt}$-2 sets $\mathsf{type}_k = 2$. Note that the value of cnt is same in both programs. If $k \neq i$, then it does not matter whether $\mathsf{type}_k = 0$ or $\mathsf{type}_k = 2$ (only the value of cnt matters). If $k == i$, then program $\mathsf{KGenAlt}$-2.3 checks if $(\mathsf{type}_k == 2)$ or $(s_k, \mathsf{vk}_k) == (s^*, \mathsf{vk}^*)$, and uses m_i in this case. As a result, if $(s_i, \mathsf{vk}_i) == (s^*, \mathsf{vk}^*)$, then either both programs use m_i, or both compute v^t. Hence the two programs are functionally identical.

□

6.3 Adversary's advantage in $\mathsf{Game}_{B,Q}$

In $\mathsf{Game}_{B,Q}$, the adversary only receives $\mathsf{type} = 2$ public keys. Recall, the j^{th} key is sampled as follows: the challenger computes s_j — a pseudorandom deterministic encryption of j. Next, it derives $(\mathsf{sigk}_j, \mathsf{vk}_j)$ pseudorandomly from s_j. If this key is corrupted, the challenger computes P_j — an obfuscation of the $\mathsf{EvalProg}$ program. This program has $u^* = \mathsf{Hash}(\mathsf{hk}, \mathsf{pk}_j)$ and $t^* = \mathsf{F}(\mathsf{K}_{\text{main}}, u^*)$ hardwired, takes $\ell - 1$ inputs $\{t_k\}$, sets $t_\ell = t^*$, and outputs $v^{a^{\mathsf{cnt}} \cdot \prod_k t_k}$. After computing P_j, it expresses this as a bit-string, and masks this with a pseudorandom string to compute the message m_j. The challenger then signs this message using sigk_j and sends m_j together with the signature.

Note that in $\mathsf{Game}_{B,Q}$, the crs program can compute the outputs on all inputs using just $v^{a^0}, v^{a^1}, \ldots, v^{a^{\ell-1}}$. However, the P_j programs still need $v^{a^{\mathsf{cnt}}}$. Therefore, we will gradually alter the outputs of these programs.

We alter the outputs of these programs via a sequence of c hybrid experiments. Let $u^*_1, \ldots, u^*_\eta$ be the hash of the first η corrupted public keys. In the η^{th} hybrid experiment, the programs have $u^*_1, \ldots, u^*_\eta$ hardwired (along with the hash of the corresponding public key),[7] as well as group elements $\left\{h_{j,j'}\right\}_{j \in [\eta], j' \in [\ell]}$ (that is, corresponding to each u^*_j, we have ℓ group elements hardwired). Consider some program P_z that takes inputs $u_1, \ldots, u_{\ell-1}$ (and sets u_{kbd} according to the hardwired u^* value). The program checks if one of the inputs is equal to the η hardwired values. If so, let j^* be the smallest value such that $u^*_{j^*}$ is one of the inputs, and suppose this input appears j' times. The program outputs $h^{\prod_k t_k}_{j^*,j'}$.

To go from the η^{th} game to the next one, we first puncture K_{main} at $u^*_{\eta+1}$ (the hash of the $(\eta+1)^{\text{th}}$ key that is corrupted). Next, we replace $t^*_{\eta+1} = \mathsf{F}(\mathsf{K}_{\text{main}}, u^*_{\eta+1})$ with $a \cdot \mathsf{F}(\mathsf{K}_{\text{main}}, u^*_{\eta+1})$. Due to this, the product of the t_k values will have extra a factors, and as a result, the output will be $v^{(a^{\ell+\cdots} \cdot (\prod_k t_k))}$. Using the DDH-powers assumption, we can therefore switch $v^{a^{\ell+\cdots}}$ to a random group element.

Formal proof

We will now show that the adversary's advantage in $\mathsf{Game}_{B,Q}$, denoted by $\mathsf{adv}_{\mathcal{A},B,Q}$ is negligible. First, let us recall $\mathsf{Game}_{B,Q}$.

- **Setup Phase**
  - The challenger chooses hash key hk, PRF K_{main}.
  - It chooses PRF key K_E, PPDE key K_{Det}, PRF keys $\mathsf{K}_{\text{C}}, \mathsf{K}_{\text{D}}$.
  - It chooses a string α, $g \leftarrow \mathbb{G}$, $a, \nu \leftarrow \mathbb{Z}_p$. It sets $v = g^\nu$.
  - It computes an obfuscation of $\mathsf{KGenAlt\text{-}2}_{Q,\mathsf{hk},\mathsf{K}_{\text{main}},\mathsf{K}_{\text{Det}},\mathsf{K}_E,\mathsf{K}_{\text{C}},\mathsf{K}_{\text{D}}}$, where $\mathsf{KGenAlt\text{-}2}_{Q,\ldots}$ is defined in Figure 20.
- **Pre/Post Challenge Queries** The challenger receives the following queries in the pre-challenge/post-challenge phase.
  - honest user registration queries - The challenger receives Q honest user registration queries.

    For the j^{th} query, the challenger does the following:

[7] Strictly speaking, if pk_j is the j^{th} corrupted key and $j \leq \eta$, then the program P_j will only have the first j hash values.

* The challenger computes $s_j = \mathsf{DetEnc}(\mathsf{K}_{\mathrm{Det}}, j)$.
* The challenger computes $r_j = \mathsf{F}(\mathsf{K}_{\mathrm{C}}, s_j)$.
* It computes $(\mathsf{sigk}_j, \mathsf{vk}_j) = \mathsf{Gen}_{sig}(1^\lambda; r_j)$.

It sends $\mathsf{pk}_j = (s_j, \mathsf{vk}_j)$ to the adversary and adds $(\mathsf{pk}_j, \ddagger\ddagger, 0)$ to T (note that the secret key corresponding to pk_j is not set at this point).

– corruption queries (at most c such queries) - On receiving a query for public key pk, the challenger checks if $\mathsf{pk} = \mathsf{pk}_j$ for some registered public key pk_j.

If $\mathsf{pk} = \mathsf{pk}_j$ for some $j \leq y$, then the corresponding secret key is chosen as follows:

* The challenger computes $u^* = \mathsf{Hash}(\mathsf{hk}, \mathsf{pk})$, $t^* = \mathsf{F}(\mathsf{K}_{\mathrm{main}}, u^*)$, $w = g^{\nu \cdot a^\ell}$, sets program $P_j = \mathsf{iO}(\mathsf{EvalProg}_{j,u^*,t^*,\ldots})$, message $m_j = P_j \oplus \mathsf{F}(K_D, (s_j, \mathsf{vk}_j)) \oplus \alpha$.
* It computes $\sigma_j = \mathsf{Sign}(\mathsf{sigk}_j, m_j)$, sends $\mathsf{sk}_j = (m_j, \sigma_j)$.

The challenger also updates the corresponding entry in T to $(\mathsf{pk}_j, \mathsf{sk}_j, 1)$.

– malicious user registration queries - the adversary sends pk to the challenger, and the challenger adds $(\mathsf{pk}, \perp, 1)$ to the database T.

– shared key queries - the adversary sends an unordered set of public keys $S = \{\widetilde{\mathsf{pk}}_1, \ldots, \widetilde{\mathsf{pk}}_\ell\}$ and an index i. If $S^* \neq \perp$ and $S = S^*$, then the challenger sends $\perp$.

For each $k \in [\ell]$, the challenger checks if there exists a record $(\mathsf{pk}_j, \mathsf{sk}_j, b_j)$ in T such that $\mathsf{pk}_j = \widetilde{\mathsf{pk}}_k$. Additionally, it checks that $\mathsf{sk}_j \neq \perp$. Let sk^* denote the secret key corresponding to $\widetilde{\mathsf{pk}}_i$.

If these checks pass, the challenger runs the crs on input $(\{\widetilde{\mathsf{pk}}_1, \ldots, \widetilde{\mathsf{pk}}_\ell\}, i, \mathsf{sk}^*)$.

- **Challenge Query** The adversary sends a set $S^* = (\mathsf{pk^*}_1, \ldots, \mathsf{pk^*}_\ell)$. The challenger checks that for each $\mathsf{pk^*}_i$, there exists a record $(\mathsf{pk}_j, \ddagger\ddagger, b_j)$ in T s.t. $b_j = 0$. Let sk^* denote the secret key corresponding to $\mathsf{pk^*}_1$.

  If these checks pass, then the challenger chooses a random bit b. If $b = 0$, the challenger does the following:

  - it computes $u_j = H(\mathsf{hk}, \mathsf{pk^*}_j)$ and $t_j = \mathsf{F}(\mathsf{K}_{\mathrm{main}}, u_j)$.
  - it computes $t' = \prod_j t_j$, $t = a^\ell \cdot t'$ and outputs $v^t = g^{\nu a^\ell t'}$.

  If $b = 1$, it sends a uniformly random element in $\mathbb{G}$ as the shared key.

We will define a sequence of hybrid experiments to show that $\mathsf{adv}_{\mathcal{A},B,Q}$ (the advantage of $\mathcal{A}$ in $\mathsf{Game}_{B,Q}$) is at most negligible.

Sequence of games for proving security in $\mathsf{Game}_{B,Q}$ We define c hybrid experiments $\{\mathsf{Game}_{B,Q,\eta}\}_{\eta \in [c]}$, where we alter the program P_j that is used in the secret key for j^{th} corruption.

- $\mathsf{Game}_{B,Q,0}$: This game is similar to $\mathsf{Game}_{B,Q}$. However, the challenger uses an obfuscation of $\mathsf{KGenAlt\text{-}2'}$ (defined in Figure 22). This program is similar to $\mathsf{KGenAlt\text{-}2}_{Q,}$, except that it does not set $\mathsf{type}_i = 1$.

Figure 20: The program $\mathsf{KGenAlt\text{-}2}_{Q,\mathsf{hk},\mathsf{K}_{\text{main}},\mathsf{K}_E,\mathsf{K}_{\text{C}},\mathsf{K}_{\text{D}},\mathsf{K}_{\text{Det}},v,a}$.

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: $\mathsf{hk}, \mathsf{K}_{\text{main}}, \mathsf{K}_E, \mathsf{K}_{\text{C}}, \mathsf{K}_{\text{D}}, \mathsf{K}_{\text{Det}}, v, a$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $k \in [\ell]$, let $u_k = H(\mathsf{hk}, (s_k, \mathsf{vk}_k))$.
3. For each $k \in [\ell]$, let $t_k = \mathsf{F}(\mathsf{K}_{\text{main}}, u_k)$.
4. Set $\mathsf{cnt} = 0$. For each $k \in [\ell]$ do the following:
   (a) Set $\mathsf{type}_k = 0$, compute $j_k = \mathsf{Dec}(\mathsf{K}_{\text{Det}}, s_k)$. Interpret j_k as a positive integer.
   (b) Compute $(m_k, r_k) = \mathsf{F}(\mathsf{K}_E, s_k)$ and $r'_k = \mathsf{F}(\mathsf{K}_{\text{C}}, s_k)$.
   (c) If $(\sigma_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{GenBind}_{sig}(m_k; r_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $Q < j_k \leq Q$, then $\mathsf{type}_k = 1$. Else if $(\mathsf{sigk}_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{Gen}_{sig}(1^\lambda; r'_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $j_k \leq Q$, then $\mathsf{type}_k = 2$.
   (d) If $\mathsf{type}_k = 1$ or $\mathsf{type}_k = 2$, then $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. If $\mathsf{cnt} == \ell$ and $\mathsf{type}_i == 2$
   - Compute $P = m_i \oplus \mathsf{F}(\mathsf{K}_{\text{D}}, (s_i, \mathsf{vk}_i)) \oplus \alpha$.
   - Output $P(\{u_j\}_{j \neq i})$.

   Else compute $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and output v^t.

- $\mathsf{Game}_{B,Q,\eta}$: Next, we define c games, one for each $\eta \in [c]$. In $\mathsf{Game}_{B,Q,\eta}$, everything except the response to corruption queries is identical to $\mathsf{Game}_{B,Q,0}$. However, the challenger uses an obfuscation of $\mathsf{EvalProg\text{-}1}_{\eta,\ldots}$ (defined in Figure 23). The challenger samples $\ell \cdot \eta$ uniformly random group elements $\{h_{j,j'}\}_{j \in [\eta], j' \in [\ell]}$.

  On receiving a corruption query for public key pk, the challenger checks if $\mathsf{pk} = \mathsf{pk}_j$ for some registered public key pk_j. If $\mathsf{pk} = \mathsf{pk}_j$ for some j, then the corresponding secret key is chosen as follows:

  – Let $s = min(j, \eta)$. The challenger sets program $P_j = \mathsf{iO}(\mathsf{EvalProg\text{-}1}_s)$ (where $\mathsf{EvalProg\text{-}1}$ is defined in Figure 23), message $m_j = P_j \oplus \mathsf{F}(K_D, (s_j, \mathsf{vk}_j)) \oplus \alpha$.
  – It computes $\sigma_j = \mathsf{Sign}(\mathsf{sigk}_j, m_j)$, sends $\mathsf{sk}_j = (m_j, \sigma_j)$.

Figure 21: The program $\mathsf{EvalProg}_{j,\mathsf{K}_{\text{main}},w,u^*,t^*}$.

Inputs: $u_1, u_2, \ldots, u_{\ell-1}$
Constants: $j, \mathsf{K}_{\text{main}}, w, u^*, t^*$

1. For each $k \in [\ell - 1]$, let $t_k = \mathsf{F}(\mathsf{K}_{\text{main}}, u_k)$. Let $u_\ell = u^*$ and $t_\ell = t^*$.
2. Compute $t' = \prod_k t_k$ and output $w^{t'}$.

Figure 22: The program $\mathsf{KGenAlt\text{-}2}'_{\mathsf{hk},\mathsf{K}_{\mathrm{main}},\mathsf{K}_E,\mathsf{K}_{\mathrm{C}},\mathsf{K}_{\mathrm{D}},\mathsf{K}_{\mathrm{Det}}}$.

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: $\mathsf{hk}, \mathsf{K}_{\mathrm{main}}, \mathsf{K}_E, \mathsf{K}_{\mathrm{C}}, \mathsf{K}_{\mathrm{D}}, \mathsf{K}_{\mathrm{Det}}, v$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $k \in [\ell]$, let $u_k = H(\mathsf{hk}, (s_k, \mathsf{vk}_k))$.
3. For each $k \in [\ell]$, let $t_k = \mathsf{F}(\mathsf{K}_{\mathrm{main}}, u_k)$.
4. Set $\mathsf{cnt} = 0$. For each $k \in [\ell]$ do the following:
   (a) Set $\mathsf{type}_k = 0$, compute $j_k = \mathsf{Dec}(\mathsf{K}_{\mathrm{Det}}, s_k)$. Interpret j_k as a positive integer.
   (b) Compute $(m_k, r_k) = \mathsf{F}(\mathsf{K}_E, s_k)$ and $r'_k = \mathsf{F}(\mathsf{K}_{\mathrm{C}}, s_k)$.
   (c) If $(\mathsf{sigk}_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{Gen}_{sig}(1^\lambda; r'_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $j_k \le Q$, then $\mathsf{type}_k = 2$.
   (d) If $\mathsf{type}_k = 2$, then $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. If $\mathsf{cnt} == \ell$ and $\mathsf{type}_i == 2$
   – Compute $P = m_i \oplus \mathsf{F}(\mathsf{K}_{\mathrm{D}}, (s_i, \mathsf{vk}_i)) \oplus \alpha$.
   – Output $P(\{u_j\}_{j \neq i})$.

   Else compute $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and output v^t.

Let $\mathsf{adv}_{\mathcal{A},B,Q,\eta}$ denote the advantage of $\mathcal{A}$ in $\mathsf{Game}_{B,Q,\eta}$.

Claim 30. *For any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that for all λ,* $\mathsf{adv}_{\mathcal{A},B,Q} - \mathsf{adv}_{\mathcal{A},B,Q,0}$ *is at most* $\mathsf{negl}(\lambda)$.

Proof. The two programs are functionally identical, since the program in $\mathsf{Game}_{B,Q}$ sets $\mathsf{type}_k = 1$ only if $Q < j_k \le Q$, and since no j_k can satisfy these two inequalities, the program never sets $\mathsf{type}_k = 1$. □

Claim 31. *For any $\eta \in \{0, \ldots, c-1\}$, for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that for all λ,* $\mathsf{adv}_{\mathcal{A},B,Q,\eta} - \mathsf{adv}_{\mathcal{A},B,Q,\eta+1}$ *is at most* $\mathsf{negl}(\lambda)$.

The proof of this claim involves a sequence of hybrid experiments, and it is described in Section 6.3.1.

Claim 32. *For any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that for all λ,* $\mathsf{adv}_{\mathcal{A},B,Q,c}$ *is at most* $\mathsf{negl}(\lambda)$.

Proof. The main idea is that the programs $\mathsf{KGenAlt\text{-}2}'$ and $\mathsf{EvalProg\text{-}1}$ do not require v^{a^ℓ}. Note that $\mathsf{KGenAlt\text{-}2}'$ uses m_i if $\mathsf{cnt} == \ell$ and $\mathsf{type}_i == 2$. If $\mathsf{cnt} < \ell$, then the program can be evaluated using constants $\{v^{a^j}\}_{j<\ell}$. If $\mathsf{cnt} == \ell$, then type_i must be 2.

Similarly, note that $\mathsf{EvalProg\text{-}1}_c$ does not use the constant $w = v^{a^\ell}$, since there exists at least one u_j (namely u_ℓ) that is equal to one of the hardwired $\{u_1^*, \ldots, u_c^*\}$.

Figure 23: The program $\mathsf{EvalProg\text{-}1}_{\mathsf{K}_{\text{main}},w,\{h_j\},\{u_j^*\}}$.

Inputs: $u_1, u_2, \ldots, u_{\ell-1}$
Constants: s, $\mathsf{K}_{\text{main}}, w, u^*, t^*, \{h_{j,j'}\}_{j\in[s],j'\in[\ell]}, \{u_j^*\}_{j\in[s]}$

1. For each $k \in [\ell-1]$, let $t_k = \mathsf{F}(\mathsf{K}_{\text{main}}, u_k)$, $u_\ell = u^*$ and $t^\ell = t^*$.
2. Find the smallest $j^* \le s$ such that $u_j = u^*_{j^*}$ for some $j \in [\ell]$.
3. If such $j^* \le s$ exists, let j' denote the number of occurrences of $u^*_{j^*}$ in input.

   Set $z = h_{j^*,j'}$. If no such j^* exists, then set $z = w$.
4. Compute $t' = \prod_{k:u_k \neq u^*_{j^*}} t_k$ and output $z^{t'}$.

On the other hand, the response to challenge query is either $v^{a^\ell \cdot (\prod_k t_k)}$ or a uniformly random group element. By setting $v = g^{c\cdot a}$ for a uniformly random c, and using the DDH-powers assumption, we can show that any PPT adversary has negligible advantage in the final experiment. □

6.3.1 Proof of Claim 31

Proof. We will prove security via a sequence of hybrid experiments.

- G_1: This experiment is similar to $\mathsf{Game}_{B,Q,\eta}$, except that the challenger, during setup, guesses the $(\eta+1)^{\text{th}}$ corruption query. That is, the challenger guesses an index j^*, indicating that the $(\eta+1)^{\text{th}}$ corruption query will correspond to the $(j^*)^{\text{th}}$ public key. If the guess is incorrect, the challenger aborts and the adversary wins with probability $1/2$.

  The challenger computes $s^* = \mathsf{DetEnc}(\mathsf{K}_{\text{Det}}, j^*)$, $(\mathsf{sigk}^*, \mathsf{vk}^*) = \mathsf{Gen}_{sig}(1^\lambda; \mathsf{F}(\mathsf{K}_{\text{C}}, s^*))$. These will be used in the following hybrid experiments.

- G_2: In this experiment, the hash key is made binding at (s^*, vk^*); that is, it chooses $\mathsf{hk} \leftarrow \mathsf{GenBind}_{Hash}(1^\lambda, (s^*, \mathsf{vk}^*))$. Since the hash key is chosen during setup, the challenger needs to correctly guess the $(\eta+1)^{\text{th}}$ corruption query.

- G_3: In this experiment, the PRF key K_{main} is punctured at $u^* = \mathsf{Hash}(\mathsf{hk}, (s^*, \mathsf{vk}^*))$. The crs is an obfuscation of $\mathsf{KGenAlt\text{-}2''}$, which is similar to $\mathsf{KGenAlt\text{-}2'}$, except that it uses the punctured PRF key, and has the value $t^* = \mathsf{F}(\mathsf{K}_{\text{main}}, u^*)$ hardwired. Similarly, in response to the corruption queries, the challenger sends an obfuscation of EvalProg-2. This is similar to EvalProg-1, except that it has the punctured key.

- G_4: This experiment is similar to the previous one, except that the value t^* is uniformly random.

- G_5: In this experiment, the challenger uses $\mathsf{KGenAlt\text{-}2'''}$ instead of $\mathsf{KGenAlt\text{-}2''}$ for the crs, and EvalProg-3 instead of EvalProg-2 for the corruption queries. In both these programs, the main change is that the challenger sets $t^* = c^* \cdot a$ for a random c^*, and the obfuscated programs do

Figure 24: The program $\mathsf{KGenAlt\text{-}2''}_{\mathsf{hk},u^*,\mathsf{K_m}\{u^*\},t^*,\mathsf{K}_E,\mathsf{K_C},\mathsf{K_D},\mathsf{K_{Det}}}$.

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: $\mathsf{hk}, \mathsf{K_m}\{u^*\}, u^*, t^*, \mathsf{K}_E, \mathsf{K_C}, \mathsf{K_D}, \mathsf{K_{Det}}, v$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $k \in [\ell]$, let $u_k = H(\mathsf{hk}, (s_k, \mathsf{vk}_k))$.
3. For each $k \in [\ell]$, if $u_k \neq u^*$, let $t_k = \mathsf{F}(\mathsf{K_m}\{u^*\}, u_k)$. Else $u_k = u^*$.
4. Set $\mathsf{cnt} = 0$. For each $k \in [\ell]$ do the following:
   (a) Set $\mathsf{type}_k = 0$, compute $j_k = \mathsf{Dec}(\mathsf{K_{Det}}, s_k)$. Interpret j_k as a positive integer.
   (b) Compute $(m_k, r_k) = \mathsf{F}(\mathsf{K}_E, s_k)$ and $r'_k = \mathsf{F}(\mathsf{K_C}, s_k)$.
   (c) If $(\mathsf{sigk}_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{Gen}_{sig}(1^\lambda; r'_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $j_k \leq Q$, then $\mathsf{type}_k = 2$.
   (d) If $\mathsf{type}_k = 2$, then $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. If $\mathsf{cnt} == \ell$ and $\mathsf{type}_i == 2$
   - Compute $P = m_i \oplus \mathsf{F}(\mathsf{K_D}, (s_i, \mathsf{vk}_i)) \oplus \alpha$.
   - Output $P(\{u_j\}_{j \neq i})$.

   Else compute $t = a^{\mathsf{cnt}} \cdot \prod_j t_j$ and output v^t.

not contain a or t^* explicitly. Instead, the programs contain the constant a *in the exponent*. The game is described in detail below.

Setup Phase

- The challenger chooses PRF key $\mathsf{K_{main}}$.
- It chooses PRF key K_E, PPDE key $\mathsf{K_{Det}}$, PRF keys $\mathsf{K_C}, \mathsf{K_D}$.
- It guesses the $(\eta+1)^{th}$ corruption query j^*. It computes $s^* = \mathsf{DetEnc}(\mathsf{K_{Det}}, j^*)$, $(\mathsf{sigk}^*, \mathsf{vk}^*) = \mathsf{Gen}_{sig}(1^\lambda; \mathsf{F}(\mathsf{K_C}, s^*))$.
  Next, it chooses $\mathsf{hk} \leftarrow \mathsf{GenBind}_{Hash}(1^\lambda, (s^*, \mathsf{vk}^*))$ and computes $u^* = \mathsf{Hash}(\mathsf{hk}, (s^*, \mathsf{vk}^*))$.
  It then punctures $\mathsf{K_{main}}$ at u^*. Let $\mathsf{K_m}\{u^*\}$ be the punctured key.
- It chooses a string α, $g \leftarrow \mathbb{G}$, $a, \nu, c^* \leftarrow \mathbb{Z}_p$. It sets $v = g^\nu$ and $t^* = c^* \cdot a$.[8]
  It computes constants $\theta_j = g^{\nu \cdot a^j}$ for each $j \in [\ell]$. These are used as constants in $\mathsf{KGenAlt\text{-}2'''}$ (defined next).
- The challenger computes an obfuscation of $\mathsf{KGenAlt\text{-}2'''}_{\mathsf{hk},u^*,\mathsf{K_m}\{u^*\},\mathsf{K_{Det}},\mathsf{K}_E,\mathsf{K_C},\mathsf{K_D},\{\theta_j\}_j}$, where $\mathsf{KGenAlt\text{-}2'''}_{\ldots}$ is defined in Figure 26.
  The challenger chooses constants $c_{j,j'}$ for each $j \in [s], j' \in [\ell]$, and sets $h_{j,j',e} = g^{c_{j,j'} \cdot (c^*)^e \cdot a^e}$ for each $j \in [s], j' \in [\ell]$, and $e \in [\ell] \cup \{0\}$. Next, the challenger sets $z_e = g^{\nu \cdot (c^*)^e \cdot a^{\ell+e}}$

[8]The constant t^* is not hardwired in any of the programs. Instead, an appropriate power of a is used in the programs.

Figure 25: The program $\mathsf{EvalProg\text{-}2}_{s,u^*,\mathsf{K_m}\{u^*\},t^*,w,\{h_{j,j'}\},\{u^*_j\}}$.

Inputs: $u_1, u_2, \ldots, u_{\ell-1}$
Constants: s, u^*, $\mathsf{K_m}\{u^*\}, t^*, w, \{h_{j,j'}\}_{j\in[s],j'\in[\ell]}, \{u^*_j\}_{j\in[s]}$

1. For each $k \in [\ell-1]$, if $u_k \neq u^*$, let $t_k = \mathsf{F}(\mathsf{K_m}\{u^*\}, u_k)$. Else $t_k = t^*$. Let $u_\ell = u^*$ and $t_\ell = t^*$.
2. Find the smallest $j^* \leq s$ such that $u_j = u^*_{j^*}$ for some $j \in [\ell]$.
3. If such $j^* \leq s$ exists, let j' denote the number of occurrences of $u^*_{j^*}$ in input.

   Set $z = h_{j^*,j'}$. If no such j^* exists, then set $z = w$.
4. Compute $t' = \prod_{k:u_k\neq u^*_{j^*}} t_k$ and output $z^{t'}$.

and $w = g^{\nu\cdot a^\ell}$, and uses these as constants in $\mathsf{EvalProg\text{-}3}$ (which is used for handling corruption queries).

Pre/Post Challenge Queries The challenger receives the following queries in the pre-challenge/post-challenge phase.

- honest user registration queries - The challenger receives Q honest user registration queries.
  For the j^{th} query, the challenger does the following:
  The challenger computes $s_j = \mathsf{DetEnc}(\mathsf{K_{Det}}, j)$.
  The challenger computes $r_j = \mathsf{F}(\mathsf{K_C}, s_j)$.
  It computes $(\mathsf{sigk}_j, \mathsf{vk}_j) = \mathsf{Gen}_{sig}(1^\lambda; r_j)$.
  It sends $\mathsf{pk}_j = (s_j, \mathsf{vk}_j)$ to the adversary and adds $(\mathsf{pk}_j, \ddagger\ddagger, 0)$ to T (note that the secret key corresponding to pk_j is not set at this point).
- corruption queries (at most c such queries) - On receiving the f^{th} corruption query for public key pk, if $f = (\eta+1)$, the challenger checks if $\mathsf{pk} = (s^*, \mathsf{vk}^*)$ and aborts if not.
  Let $s = min(f, \eta)$, and $u^*_1, \ldots, u^*_s$ are the hash of the first s public keys.
  The challenger uses the following constants to be hardwired in the program P_f:
  u^*, u^*_j for each $j \in [s]$, $\mathsf{K_m}\{u^*\}$, s, $h_{j,j',e}$ for each $j \in [s], j' \in [\ell]$ and $e \in [\ell]\cup\{0\}$, z_e for each $e \in [\ell]$ and w.
  It sets program $P_f = \mathsf{iO}(\mathsf{EvalProg\text{-}3}_{s,})$, message $m_f = P_f \oplus \mathsf{F}(K_D, (s_f, \mathsf{vk}_f)) \oplus \alpha$.
  It computes $\sigma_f = \mathsf{Sign}(\mathsf{sigk}_f, m_f)$, sends $\mathsf{sk}_f = (m_f, \sigma_f)$.

- G_6: In this experiment, the challenger replaces z_e with uniformly random elements in $\mathbb{G}$.
- G_7: In this experiment, the challenger uses program $\mathsf{EvalProg\text{-}4}$ for responding to corruption queries. The main differences are highlighted below.

  On receiving the f^{th} corruption query for public key pk, if $f = (\eta+1)$, the challenger checks if $\mathsf{pk} = (s^*, \mathsf{vk}^*)$ and aborts if not.

Figure 26: The program $\mathsf{KGenAlt\text{-}2'''}_{\mathsf{hk},u^*,\mathsf{K}_\mathrm{m}\{u^*\},\mathsf{K}_E,\mathsf{K}_\mathrm{C},\mathsf{K}_\mathrm{D},\mathsf{K}_\mathrm{Det},\{\theta_j\}_j}$.

Inputs: $\mathsf{pk}_1 = (s_1, \mathsf{vk}_1), \ldots, \mathsf{pk}_\ell = (s_\ell, \mathsf{vk}_\ell), i, m_i, \sigma_i$
Constants: $\mathsf{hk}, \mathsf{K}_\mathrm{m}\{u^*\}, u^*, t^*, \mathsf{K}_E, \mathsf{K}_\mathrm{C}, \mathsf{K}_\mathrm{D}, \mathsf{K}_\mathrm{Det}, \{\theta_j\}_j$

1. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
2. For each $k \in [\ell]$, let $u_k = H(\mathsf{hk}, (s_k, \mathsf{vk}_k))$.
3. For each $k \in [\ell]$, if $u_k \neq u^*$, let $t_k = \mathsf{F}(\mathsf{K}_\mathrm{m}\{u^*\}, u_k)$.
4. Set $\mathsf{cnt} = 0$. For each $k \in [\ell]$ do the following:
   (a) Set $\mathsf{type}_k = 0$, compute $j_k = \mathsf{Dec}(\mathsf{K}_\mathrm{Det}, s_k)$. Interpret j_k as a positive integer.
   (b) Compute $(m_k, r_k) = \mathsf{F}(\mathsf{K}_E, s_k)$ and $r'_k = \mathsf{F}(\mathsf{K}_\mathrm{C}, s_k)$.
   (c) If $(\mathsf{sigk}_{s_k}, \mathsf{vk}_{s_k}) = \mathsf{Gen}_{sig}(1^\lambda; r'_k)$ and $\mathsf{vk}_{s_k} = \mathsf{vk}_k$ and $j_k \leq Q$, then $\mathsf{type}_k = 2$.
   (d) If $\mathsf{type}_k = 2$, then $\mathsf{cnt} = \mathsf{cnt} + 1$.
5. If $\mathsf{cnt} == \ell$ and $\mathsf{type}_i == 2$
   – Compute $P = m_i \oplus \mathsf{F}(\mathsf{K}_\mathrm{D}, (s_i, \mathsf{vk}_i)) \oplus \alpha$.
   – Output $P(\{u_j\}_{j \neq i})$.

   Else if $\exists j$ s.t. $u_j = u^*$ then compute $t' = \prod_{j:u_j \neq u^*} t_j$ and output $\theta^{c^* \cdot t'}_{\mathsf{cnt}+1}$.

   Else output $\theta^{t'}_{\mathsf{cnt}}$.

Let $s = min(f, \eta)$, and $u^*_1, \ldots, u^*_s$ are the hash of the first s public keys.

The challenger uses the following constants to be hardwired in the program P_f:

u^*, t^*, u^*_j for each $j \in [s]$, $\mathsf{K}_\mathrm{m}\{u^*\}$, s, $h_{j,j'} = g^{c_{j,j'}}$ for each $j \in [s], j' \in [\ell]$, z_e for each $e \in [\ell]$ and w.

It sets program $P_f = \mathsf{iO}(\mathsf{EvalProg\text{-}4}_{s,})$, message $m_f = P_f \oplus \mathsf{F}(K_D, (s_f, \mathsf{vk}_f)) \oplus \alpha$. The program is defined in Figure 28.

It computes $\sigma_f = \mathsf{Sign}(\mathsf{sigk}_f, m_f)$, sends $\mathsf{sk}_f = (m_f, \sigma_f)$.

- G_8: In this game, the challenger uses $\mathsf{EvalProg\text{-}2}$, but with s set to be the minimum of the index of corruption query, and $\eta + 1$.

  The challenger, during setup, chooses uniformly random group elements $h_{j,j'}$ for each $j \in [\eta+1]$, $j' \in [\ell]$.

  For the f^{th} corruption query, if $f \leq \eta$, the challenger uses an obfuscation of $\mathsf{EvalProg\text{-}2}_{f,u^*,\mathsf{K}_\mathrm{m}\{u^*\},t^*,w,\{h_{j,j'}\}_{j \leq f, j' \leq \ell},\{u^*_j\}_{j \leq f}}$.

  For $f \geq \eta + 1$, the challenger uses an obfuscation of $\mathsf{EvalProg\text{-}2}_{\eta+1,\ldots}$ where the hardwired constants are as follows:

Figure 27: The program $\mathsf{EvalProg\text{-}3}_{\mathsf{K_m}\{u^*\},\{u^*_j\},w,\{h_{j,j',e}\},\{z_e\}}$.

Inputs: $u_1, u_2, \ldots, u_{\ell-1}$
Constants: s, u^*, $\mathsf{K_m}\{u^*\}, t^*, w, \{h_{j,j',e}\}_{j\in[s],j'\in[\ell],e\in[\ell]}, \{z_e\}_{e\in[\ell]}, \{u^*_j\}_{j\in[s]}$

1. For each $k \in [\ell-1]$, if $u_k \neq u^*$, let $t_k = \mathsf{F}(\mathsf{K_{main}}, u_k)$, else set $t_k = t^*$. Let $u_\ell = u^*$ and $t_\ell = t^*$.
2. Find the smallest $j^* \leq s$ such that $u_j = u^*_{j^*}$ for some $j \in [\ell]$.
   If such $j^* \leq s$ exists, let j' denote the number of occurrences of $u^*_{j^*}$ in input.
   Let e denote the number of times u^* appears in the input.
3. Compute $t' = \prod_{k:u_k\neq u^*_{j^*}, u_k\neq u^*} t_k$.
   If j^* is defined, then output $h^{t'}_{j^*,j',e}$.
   Else if j^* is undefined but $e \neq 0$ then output $z_e^{t'}$.
   Else output $w^{t'}$.

u^* and punctured PRF key $\mathsf{K_m}\{u^*\}$, the PRF evaluation at u^* - t^*, $w = g^{\nu\cdot a^\ell}$, hash of the first $\eta+1$ corrupted public keys $\{u^*_j\}_{j\leq\eta+1}$ (note that $u^*_{\eta+1} = u^*$), and group elements $h_{j,j'}$ for each $j \leq \eta+1, j' \leq \ell$.

- G_9: In this game, the challenger sets $t^* = \mathsf{F}(\mathsf{K_{main}}, u^*)$.
- G_{10}: In this game, the challenger uses $\mathsf{EvalProg\text{-}1}_{\eta+1}$.
- G_{11}: This is similar to $\mathsf{Game}_{B,Q,\eta+1}$, except that the challenger, during setup, guesses the $(\eta+1)^{th}$ corruption query. Compared to the previous game, the only difference is that the hash function is not in binding mode.

Claim 33. *For any adversary $\mathcal{A}$, $|\mathsf{adv}_{\mathcal{A},B,Q,\eta} - \mathsf{adv}_{\mathcal{A},B,Q,\eta+1}| = \frac{1}{Q}|\mathsf{adv}_{\mathcal{A},G_1} - \mathsf{adv}_{\mathcal{A},G_{11}}|$.*

Proof. This follows from the definition of the experiments. Note that the advantage of $\mathcal{A}$ in G_1 is $\frac{1}{Q}\mathsf{adv}_{\mathcal{A},B,Q,\eta}$, since the challenger guesses the honest registration index which corresponds to the η^{th} corruption, and this guess is correct with probability $1/Q$. Similarly, $\mathsf{adv}_{\mathcal{A},G_{11}} = \frac{1}{Q}\mathsf{adv}_{\mathcal{A},B,Q,\eta+1}$. □

Claim 34. *Assuming the mode indistinguishability of Hash, for any PPT adversary $\mathcal{A}$, there exists a negligible function negl such that for all λ, $|\mathsf{adv}_{\mathcal{A},G_1} - \mathsf{adv}_{\mathcal{A},G_2}| \leq \mathsf{negl}(\lambda)$.*

Proof. The only difference in G_1 and G_2 is that the hash key is binding at u^*. Using the indistinguishability property of Hash, the two games are computationally indistinguishable. □

Claim 35. *Assuming the correctness of the PRF scheme, and the security of iO, for any PPT adversary $\mathcal{A}$, there exists a negligible function negl such that for all λ, $|\mathsf{adv}_{\mathcal{A},G_2} - \mathsf{adv}_{\mathcal{A},G_3}| \leq \mathsf{negl}(\lambda)$.*

Figure 28: The program $\mathsf{EvalProg\text{-}4}_{\mathsf{K}_m\{u^*\},t^*,w,\{h_j\},\{u_j^*\}}$.

Inputs: $u_1, u_2, \ldots, u_{\ell-1}$

Constants: s, u^*, $\mathsf{K}_m\{u^*\}, t^*, w$, $\{h_{j,j'}\}_{j\in[s],j'\in[\ell]}$, $\{z_e\}_{e\in[\ell]}, \{u_j^*\}_{j\in[s]}$

1. For each $k \in [\ell-1]$, if $u_k \neq u^*$, let $t_k = \mathsf{F}(\mathsf{K}_{\text{main}}, u_k)$, else set $t_k = t^*$. Let $u_\ell = u^*$ and $t_\ell = t^*$.
2. Find the smallest $j^* \leq s$ such that $u_j = u_{j^*}^*$ for some $j \in [\ell]$.
3. If such $j^* \leq s$ exists, let j' denote the number of occurrences of u_{j^*} in input.
4. Let e denote the number of times u^* appears in the input.
   Set $z = h_{j^*,j'}$. If no such j^* exists, then set $z = w$.
5. Compute $t' = (\prod_{k:u_k\neq u_{j^*}^*, u_k\neq u^*} t_k)$, $t = t' \cdot (t^*)^e$.
   If j^* is defined, then output $h_{j^*,j'}^t$.
   Else if j^* is undefined but $e \neq 0$ then output $z_e^{t'}$
   Else output w^t.

Proof. In G_3, the challenger punctures the PRF key K_{main} at u^*. The PRF evaluation at u^* is hardwired in the programs $\mathsf{KGenAlt\text{-}2}_Q$, and $\mathsf{EvalProg\text{-}1}$. The punctured PRF key can evaluate at all points other than u^*, and the evaluation at u^* is hardwired in these programs. Therefore, the programs in G_3 are functionally identical to the corresponding programs in G_2, and using the security of iO, the games are computationally indistinguishable. □

Claim 36. *Assuming the security of the PRF scheme, for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that for all λ, $|\mathsf{adv}_{\mathcal{A},G_3} - \mathsf{adv}_{\mathcal{A},G_4}| \leq \mathsf{negl}(\lambda)$.*

Proof. The only difference in the two games is the constant t^* hardwired in the obfuscated programs. In one case, t^* is computed using K_{main}, while in G_4, it is a uniformly random element. Since the adversary only receives the punctured PRF key, the evaluation at u^* is indistinguishable from a uniformly random element in $\mathbb{Z}_p$. □

Claim 37. *Assuming the correctness of the PRF scheme, and the security of* iO*, for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that for all λ, $|\mathsf{adv}_{\mathcal{A},G_4} - \mathsf{adv}_{\mathcal{A},G_5}| \leq \mathsf{negl}(\lambda)$.*

Proof. First, in G_5, t^* is set to be $c^* \cdot a$, where c^* is a uniformly random element in $\mathbb{Z}_p$. Hence, t^* is distributed as a uniformly random element in $\mathbb{Z}_p$. Next, the programs $\mathsf{KGenAlt\text{-}2''}$ and $\mathsf{EvalProg\text{-}2}$ contain powers of a. The constants hardwired in the programs are such that the programs are functionally identical to $\mathsf{KGenAlt\text{-}2'}$ and $\mathsf{EvalProg\text{-}1}$ respectively. Below, we argue this via a case analysis:

- In Step 5, if $\mathsf{cnt} == \ell$ and $\mathsf{type}_i = 2$, then both $\mathsf{KGenAlt\text{-}2'}$ and $\mathsf{KGenAlt\text{-}2''}$ have the same output (they both use the message m_i to recover P, and output $P(u_1, \ldots, u_\ell)$.

- In Step 5, if either $\mathsf{cnt} \neq \ell$ or $\mathsf{type}_i = 0$ then $\mathsf{KGenAlt\text{-}2'}$ outputs $v^{a^{\mathsf{cnt}} \cdot (\prod_j t_j)}$. If none of the u_j are equal to u^*, then this is equal to $v^{a^{\mathsf{cnt}} \cdot (\prod_j t_j)}$. Since $\theta_\omega = g^{\nu \cdot a^\omega}$, the outputs of $\mathsf{KGenAlt\text{-}2'}$ and $\mathsf{KGenAlt\text{-}2''}$ are identical.

- In Step 5, if either $\mathsf{cnt} \neq \ell$ or $\mathsf{type}_i = 0$ then $\mathsf{KGenAlt\text{-}2'}$ outputs $v^{a^{\mathsf{cnt}} \cdot (\prod_j t_j)}$. If there exists one u_j that is equal to u^*,[9] then the output is $v^{a^{\mathsf{cnt}} \cdot (\prod_{j:u_j \neq u^*} t_j) \cdot t^*} = (g^{\nu \cdot a^{\mathsf{cnt}+1}})^{c^* \cdot \prod_{j:u_j \neq u^*} t_j}$. This is identical to the output of $\mathsf{KGenAlt\text{-}2''}$, given that $\theta_\omega = g^{\nu \cdot a^\omega}$

□

Claim 38. *Assuming the* DDH-powers *assumption, for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that for all λ, $\mathsf{adv}_{\mathcal{A},5} - \mathsf{adv}_{\mathcal{A},6}$ is at most $\mathsf{negl}(\lambda)$.*

Proof. The only difference in the two experiments is that the z_e terms are switched to being random group elements. This follows from the DDH-powers assumption. Note that all the $h_{j,j',e}$ terms can be generated using g^{a^e}, and the z_e terms can be computed from the challenge terms T_e, which is either $g^{a^{\ell+e}}$ or a uniformly random group element.

□

Claim 39. *Assuming the security of* iO*, for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that for all λ, $|\mathsf{adv}_{\mathcal{A},6} - \mathsf{adv}_{\mathcal{A},7}| \leq \mathsf{negl}(\lambda)$.*

Proof. In both these games, the challenger sends c obfuscated programs, one for each corrupt key query. It suffices to show that for all $f \in [c]$, the f^{th} program in G_6 and G_7 are functionally identical. Fix any corrupt key query index f, and consider the following cases:

1. there exists some input u_j that's equal to one of the u_k^*. Let j^* be the smallest index such that $u_{j^*}^*$ is equal to some input u_j, and let j' denote the number of inputs that are equal to $u_{j^*}^*$. Let e denote the number of inputs that are equal to u^*. In $\mathsf{EvalProg\text{-}3}$, the program outputs $h_{j^*,j',e}^{t'}$ where t' is the product of all t_k such that $u_k \notin \{u^*, u_{j^*}^*\}$. In $\mathsf{EvalProg\text{-}4}$, the program outputs $h_{j^*,j'}^{t}$, where $t = t' \cdot (t^*)^e$. These two outputs are identical since $h_{j^*,j',e}$ in G_6 is equal to $h_{j^*,j'}^{a^{(t^*)^e}}$.

2. none of the inputs are equal to the u_k^*, but e of them are equal to u^*. In this case, both programs output $z_e^{t'}$, where t' is the product of all t_k such that $u_k \neq u^*$.

3. none of the inputs are equal to either the u_k^* or u^*. In this case, both programs output w^t where t is the product of all t_k. [10]

□

Claim 40. *Assuming the security of* iO*, for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that for all λ, $|\mathsf{adv}_{\mathcal{A},7} - \mathsf{adv}_{\mathcal{A},8}| \leq \mathsf{negl}(\lambda)$.*

[9] Note that there can be at most one such u_j.

[10] The last line of $\mathsf{EvalProg\text{-}3}$ outputs $w^{t'}$, but note that $t' = \prod_{k:u_k \notin \{u_k^*, u^*\}} t_k$, and since none of the u_k are equal to any of the u_j^* or u^*, t' is simply the product of all t_k.

Proof. The main difference in these two experiments is the response to corruption queries. In G_7, the challenger uses an obfuscation of $\mathsf{EvalProg}$-4 with the following constants: s - minimum of index of corruption query and η, u^*, t^*, u^*_j for each $j \in [s]$, $\mathsf{K}_{\mathrm{m}}\{u^*\}$, $h_{j,j'} = g^{c_{j,j'}}$ for each $j \in [s], j' \in [\ell]$, z_e for each $e \in [\ell]$ and w.

In G_8, the challenger uses an obfuscation of $\mathsf{EvalProg}$-2 with the following constants: s - minimum of index of corruption query and $\eta + 1$, u^*, $\mathsf{K}_{\mathrm{m}}\{u^*\}$, t^*, $w = g^{\nu \cdot a^\ell}$, hash of the first $\eta + 1$ corrupted public keys $\{u^*_j\}_{j \le \eta+1}$ (note that $u^*_{\eta+1} = u^*$), and group elements $h_{j,j'}$ for each $j \le \eta + 1, j' \le \ell$.

We will consider two cases:

- the index of corruption query is at least $\eta+1$ (and therefore $s = \eta$): here, we have the following sub-cases, depending on the input:

  1. the input contains a u_j that is equal to some u^*_k, $k \le \eta$: Let j^* be the smallest such index where one of the inputs is equal to $u^*_{j^*}$. In this case, both programs output $h^t_{j^*,j'}$, where $t = \prod_{k:u_k \neq u^*_{j^*}} t_k$. In one case (in program $\mathsf{EvalProg}$-2), this t is computed directly, while in $\mathsf{EvalProg}$-4, the program first computes $t' = \prod_{k:u_k \neq u^*_{j^*}, u_k \neq u^*} t_k$ and then multiplies the correct power of t^*.
  2. the input contains no u_j that is equal to some u^*_k, $k \le \eta$, but it contains some inputs that are equal to u^*: in program $\mathsf{EvalProg}\text{-}2_{\eta+1,}$, the program outputs $h^t_{\eta+1,e}$ where e is the number of times u^* is present in the input, and $t = \prod_{k:u_k \neq u^*} t_k$. In $\mathsf{EvalProg}\text{-}4_{\eta,}$, the program outputs z^t_e, where z_e is distributed identically to $h_{\eta+1,e}$ (both are uniformly random group elements) and $t = \prod_{k:u_k \neq u^*} t_k$.
  3. the input contains no u_j that is equal to some u^*_k, $k \le \eta$ or u^*: here, both programs output w^t where $t = \prod_k t_k$.

- the index of corruption query is at most η: Let f denote the index of the corruption query. There are two sub-cases, depending on the input:

  1. the input contains a u_j that is equal to some u^*_k, $k \le f$: This case is similar to the case 1 above.
  2. the input doesn't contain a u_j equal to one of the u^*_k, $k \le f$: In this case, both programs output w^t where $w = g^{\nu \cdot a^\ell}$ and $t = \prod_k t_k$.

□

Claim 41. *Assuming the PRF security of* F*, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ*,* $|\mathsf{adv}_{\mathcal{A},8} - \mathsf{adv}_{\mathcal{A},9}| \le \mathsf{negl}(\lambda)$*.*

Proof. In G_8, the constant t^* is chosen uniformly at random, while in G_9, it is computed using the PRF key $\mathsf{K}_{\mathrm{main}}$ at input u^*. The adversary receives only the punctured PRF key $\mathsf{K}_{\mathrm{m}}\{u^*\}$, and therefore, using the PRF security, these two games are computationally indistinguishable. □

Claim 42. *Assuming the security of* iO*, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ*,* $|\mathsf{adv}_{\mathcal{A},9} - \mathsf{adv}_{\mathcal{A},10}| \le \mathsf{negl}(\lambda)$*.*

Proof. The only difference between G_9 and G_{10} is with regard to the PRF computation at u^*. In G_9, this value is hardwired in the program, while it is computed within the program in G_{10}. Therefore the programs obfuscated are functionally identical, and hence their obfuscations are computationally indistinguishable.

□

Claim 43. *Assuming the mode indistinguishability of* Hash*, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ*,* $\mathsf{adv}_{\mathcal{A},10} - \mathsf{adv}_{\mathcal{A},11}$ *is at most* $\mathsf{negl}(\lambda)$*.*

Proof. The only difference between G_{10} and G_{11} is that the hash key is binding at u^* in G_{10}, and it is chosen in non-binding mode in G_{11}. Using the security of Hash, it follows that the two games are computationally indistinguishable. □

□

Acknowledgements

We thank Rachit Garg and George Lu for helpful feedback on an earlier draft of our work.

References

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Heidelberg, August 2015.

[AMN+18] Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for NC^1 in traditional groups. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 543–574. Springer, Heidelberg, August 2018.

[BDGM20] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate iO from homomorphic encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 79–109. Springer, Heidelberg, May 2020.

[BFP+15] Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens. Key-homomorphic constrained pseudorandom functions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 31–60. Springer, Heidelberg, March 2015.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

[BGK+18] Dan Boneh, Darren B. Glass, Daniel Krashen, Kristin E. Lauter, Shahed Sharif, Alice Silverberg, Mehdi Tibouchi, and Mark Zhandry. Multiparty non-interactive

key exchange and more from isogenies on elliptic curves. *Journal of Mathematical Cryptology*, 14:5 – 14, 2018.

[BGMZ18] James Bartusek, Jiaxin Guan, Fermi Ma, and Mark Zhandry. Return of GGH15: Provable security against zeroizing attacks. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 544–574. Springer, Heidelberg, November 2018.

[BIJ+20] James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai, and Mark Zhandry. Affine determinant programs: A framework for obfuscation and witness encryption. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 82:1–82:39. LIPIcs, January 2020.

[BJLS16] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the impossibility of tight cryptographic reductions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 273–304. Springer, Heidelberg, May 2016.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2002.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 264–302. Springer, Heidelberg, November 2017.

[BV15a] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, October 2015.

[BV15b] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 1–30. Springer, Heidelberg, March 2015.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 480–499. Springer, Heidelberg, August 2014.

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 446–476. Springer, Heidelberg, April / May 2017.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 257–270. Springer, Heidelberg, August 1994.

[CKS08] David Cash, Eike Kiltz, and Victor Shoup. The twin Diffie-Hellman problem and applications. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 127–145. Springer, Heidelberg, April 2008.

[CM14] Melissa Chase and Sarah Meiklejohn. Déjà Q: Using dual systems to revisit q-type assumptions. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 622–639. Springer, Heidelberg, May 2014.

[CRV16] Nishanth Chandran, Srinivasan Raghuraman, and Dhinakaran Vinayagamurthy. Reducing depth in constrained PRFs: From bit-fixing to $\mathbf{NC}^1$. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 359–385. Springer, Heidelberg, March 2016.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DKN+20] Alex Davidson, Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively secure constrained pseudorandom functions in the standard model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 559–589. Springer, Heidelberg, August 2020.

[DKW16] Apoorvaa Deshpande, Venkata Koppula, and Brent Waters. Constrained pseudorandom functions for unconstrained inputs. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 124–153. Springer, Heidelberg, May 2016.

[DKXY02] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 65–82. Springer, Heidelberg, April / May 2002.

[FHKP13] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 254–271. Springer, Heidelberg, February / March 2013.

[FKPR14] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained PRFs. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 82–101. Springer, Heidelberg, December 2014.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990.

[FN94] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 480–491. Springer, Heidelberg, August 1994.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, Heidelberg, May 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013.

[GMM+16] Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 241–268. Springer, Heidelberg, October / November 2016.

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 736–749, New York, NY, USA, 2021. Association for Computing Machinery.

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 579–604. Springer, Heidelberg, August 2016.

[GPSZ17] Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 156–181. Springer, Heidelberg, April / May 2017.

[GS16] Sanjam Garg and Akshayaram Srinivasan. Single-key to multi-key functional encryption with polynomial loss. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 419–442. Springer, Heidelberg, October / November 2016.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, August 2012.

[GWZ21] Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Incompressible cryptography. Cryptology ePrint Archive, Report 2021/1679, 2021. https://eprint.iacr.org/2021/1679.

[HHK18] Julia Hesse, Dennis Hofheinz, and Lisa Kohl. On tightly secure non-interactive key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 65–94. Springer, Heidelberg, August 2018.

[HHKL21] Julia Hesse, Dennis Hofheinz, Lisa Kohl, and Roman Langrehr. Towards tight adaptive security of non-interactive key exchange. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography*, pages 286–316, Cham, 2021. Springer International Publishing.

[HJK+16] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 715–744. Springer, Heidelberg, December 2016.

[HKKW14] Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. Cryptology ePrint Archive, Report 2014/720, 2014. https://eprint.iacr.org/2014/720.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 79–102. Springer, Heidelberg, November / December 2015.

[Hof14] Dennis Hofheinz. Fully secure constrained pseudorandom functions using random oracles. Cryptology ePrint Archive, Report 2014/372, 2014. https://eprint.iacr.org/2014/372.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. *Indistinguishability Obfuscation from Well-Founded Assumptions*, page 60–73. Association for Computing Machinery, New York, NY, USA, 2021.

[Jou00] Antoine Joux. A one round protocol for tripartite diffie–hellman. In Wieb Bosma, editor, *Algorithmic Number Theory*, pages 385–393, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[KPW17] Venkata Koppula, Andrew Poelstra, and Brent Waters. Universal samplers with fast verification. In Serge Fehr, editor, *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 II*, volume 10175 of *Lecture Notes in Computer Science*, pages 525–554. Springer, 2017.

[KRS15] Dakshita Khurana, Vanishree Rao, and Amit Sahai. Multi-party key exchange for unbounded parties from indistinguishability obfuscation. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 52–75. Springer, Heidelberg, November / December 2015.

[LOS+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 62–91. Springer, Heidelberg, May / June 2010.

[LW14] Allison B. Lewko and Brent Waters. Why proving HIBE systems secure is difficult. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 58–76. Springer, Heidelberg, May 2014.

[LZ17] Qipeng Liu and Mark Zhandry. Decomposable obfuscation: A framework for building applications of obfuscation from polynomial hardness. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 138–169. Springer, Heidelberg, November 2017.

[MZ17] Fermi Ma and Mark Zhandry. Encryptor combiners: A unified approach to multiparty NIKE, (H)IBE, and broadcast encryption. Cryptology ePrint Archive, Report 2017/152, 2017. https://eprint.iacr.org/2017/152.

[MZ18] Fermi Ma and Mark Zhandry. The MMap strikes back: Obfuscation and new multilinear maps immune to CLT13 zeroizing attacks. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 513–543. Springer, Heidelberg, November 2018.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997.

[Rao14] Vanishree Rao. Adaptive multiparty non-interactive key exchange without setup in the standard model. Cryptology ePrint Archive, Report 2014/910, 2014. https://eprint.iacr.org/2014/910.

[STW96] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman key distribution extended to group communication. In Li Gong and Jacques Stern, editors, *ACM CCS 96*, pages 31–37. ACM Press, March 1996.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484. ACM, 2014.

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer, Heidelberg, May 2005.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 619–636. Springer, Heidelberg, August 2009.

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 127–156. Springer, Heidelberg, October 2021.

A 1-SF-PRF: From Security Against Unique-Query Adversaries to General Adversaries

Let $(\mathsf{Gen}_{uq}, \mathsf{Eval}_{uq}, \mathsf{Constr}_{uq}, \mathsf{EvalC}_{uq})$ be a 1-SF-PRF scheme secure against unique query adversaries, where the constraining algorithm Constr_{uq} requires $r(\lambda)$ bits of randomness to compute the

constrained key. Let F be a standard pseudorandom function with input domain $[n] \times \Sigma$ and range $\{0,1\}^{r(\lambda)}$.

Consider the following 1-SF-PRF scheme:

- Gen: The key generation algorithm chooses $K \leftarrow \mathsf{Gen}_{uq}$ and (standard) PRF key K_{F}. The key consists of K and K_{F}.
- $\mathsf{Eval}((K, K_{\mathsf{F}}), x)$: The evaluation on input x is $\mathsf{Eval}_{uq}(K, x)$.
- $\mathsf{Constr}((K, K_{\mathsf{F}}), (i, z))$: The constraining algorithm first computes $r = \mathsf{F}(K_{\mathsf{F}}, (i, z))$. Next, it uses r as the randomness for computing the constrained key; that is, it sets $K_{(i,z)} = \mathsf{Constr}_{uq}(K, (i, z); r)$.
- $\mathsf{EvalC}(K_{i,z}, x)$: The constrained evaluation simply outputs $\mathsf{EvalC}_{uq}(K_{i,z}, x)$.

Claim 1. *Let* $(\mathsf{Gen}_{uq}, \mathsf{Eval}_{uq}, \mathsf{Constr}_{uq}, \mathsf{EvalC}_{uq})$ *be a 1-SF-PRF scheme secure against unique query adversaries, and let* F *be a standard pseudorandom function. Then* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Constr}, \mathsf{EvalC})$ *is a 1-SF-PRF scheme secure against general adversaries (as per Definition 4).*

Proof. Consider the following intermediate security game G: the challenger receives constrained key queries from the adversary, and maintains a table of (query, response) pairs. For each query (i, z), it checks if (i, z) is present in the table. If so, it returns the corresponding response. Else, it computes $K_{(i,z)} \leftarrow \mathsf{Constr}_{uq}(K, (i, z))$ (using true randomness), sends $K_{(i,z)}$ to the adversary, and adds $((i, z), K_{(i,z)})$ to the table.

If an adversary has advantage ϵ in the 1-SF-PRF security game (against our scheme), then the adversary has advantage $\epsilon - \mathsf{negl}$ in the above defined security game (for some negligible function negl). Note that the only difference between the two security games is that in one case, the randomness for constraining queries is generated using $\mathsf{F}(K_{\mathsf{F}}, \cdot)$, while in the other case, it is chosen at random. Using the security of the PRF scheme, it follows that any adversary has nearly identical advantage in both the security games.

Next, we use the security against unique-query adversaries to argue that any adversary has negligible advantage in the security game G. This concludes our proof. □

B Missing NIKE Compiler Proofs

Here, we formally prove Theorem 3.

B.1 Achieving Adversarial Correctness

Here, we prove Theorem 3 for achieving adversarial correctness, which is repeated for convenience:

Theorem 3. *Assume there exists a multi-party NIKE with perfect correctness, potentially in the crs model. Assume additionally there exists a NIZK. Then there exists a multi-party NIKE with both perfect and adversarial correctness in the crs model. If the perfectly correct scheme has unbounded honest users, corruptions, and/or set size, then so does the resulting adversarially correct scheme.*

Proof. The proofs of the various bounded/unbounded cases are essentially the same, so we focus on the case where everything is bounded. The construction and proof are straightforward: we simply

append a NIZK to every public key, proving that it was generated using the honest Pub algorithm for some choice of random coins. KeyGen will then verify all input NIZKs and abort if any are rejected. By the perfect correctness of the underling NIKE and the soundness of the NIZK, there will be no valid public keys that cause different users to output different shared keys on the same set, implying (statistical) adversarial correctness.

The main caveat is that we need a multi-theorem NIZK, which can be constructed generically from any plain NIZK [FLS90]. We will place the NIZK crs in the NIKE crs. Hence, even if we start with a setup-free NIKE, the result of Theorem 3 is a NIKE with setup. □

C Missing Proofs from Section 4

Here, we provide the missing proofs from Section 4.

C.1 Adding Eval Queries to a 1-SF-PRF

Here, we show how to upgrade a 1-SF-PRF that is adaptively secure without Eval queries to one that is adaptively secure (with Eval queries). The idea is very similar to the proof of Theorem 5. This is not technically necessary for our results, but is provided for completeness, since it shows how to add such queries without having to go through NIKE. The latter would require assuming iO, whereas our direct conversion is completely generic.

Let $(\mathsf{Gen}', \mathsf{Eval}', \mathsf{Constr}', \mathsf{EvalC}')$ be a 1-SF-PRF that is adaptively secure without Eval queries. Consider the new 1-SF-PRF $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Constr}, \mathsf{EvalC})$ defined as:

- $\mathsf{Gen}(1^\lambda, 1^{|\Sigma|}, 1^\ell)$: Let $\Sigma' = \mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq \max(|\Sigma|, 2\ell)$. Arbitrarily embed $\Sigma \subset \Sigma'$. Run and output $k \leftarrow \mathsf{Gen}'(1^\lambda, 1^{|\Sigma'|}, 1^{2\ell})$.
- $\mathsf{Eval}(k, x)$: Let P be the unique polynomial of degree $\ell-1$ such that $P(i) = x_i$ for $i = 0, \ldots, \ell-1$. Let $y = (P(1), P(2), \ldots, P(2\ell)) = (x_1, \ldots, x_\ell, P(\ell+1), P(\ell+2), \ldots, P(2\ell))$. Run $\mathsf{Eval}'(k, y)$.
- $\mathsf{Constr}(k, i, z) = \mathsf{Constr}'(k, i, z)$
- $\mathsf{Eval}(k_{i,z}, x) = \mathsf{Eval}'(k_{i,z}, y)$ where y is derived from x as above.

Theorem 8. *If* $(\mathsf{Gen}', \mathsf{Eval}', \mathsf{Constr}', \mathsf{EvalC}')$ *is adaptively secure without Eval queries, then* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Constr}, \mathsf{EvalC})$ *is adaptively secure (with Eval queries).*

Proof. Let $\mathcal{A}$ be a supposed adversary for $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Constr}, \mathsf{EvalC})$ that wins with probability $\frac{1}{2} + \epsilon$ for a non-negligible ϵ. Let q be a polynomial upper bound on the number of Eval queries made by $\mathcal{A}$. We will construct a new adversary $\mathcal{A}'$ for $(\mathsf{Gen}', \mathsf{Eval}', \mathsf{Constr}', \mathsf{EvalC}')$, which makes no Eval queries.

Before defining $\mathcal{A}'$, consider a list $L \subseteq \mathbb{F}^\ell$ of size at most q, and let $M \subseteq \mathbb{F}^{2\ell}$ where the elements $y \in M$ are derived from the elements $x \in L$ by the process above. Also consider an $x^* \in \mathbb{F}^\ell \setminus L$ and let y^* be derived from x^*. Clearly, $y^* \notin M$ and $|M| = |L|$.

Now fix an integer $t > 0$ and a value $r \in [0, 1]$ such that $r|\mathbb{F}|$ is an integer. Consider the following process:

- For $j = 1, \ldots, t$, choose a random $i_j \in [2\ell]$. Then select a random subset $S_j \subseteq \mathbb{F}$ of size $r|\mathbb{F}|$.
- Output 1 if the following conditions are met:

1. For every j, $y^*_{i_j} \notin S_j$, and
2. For each $y \in M$, there exists a j such that $y_{i_j} \in S_j$.

Otherwise, output 0.

Let p_{L,x^*} be the probability the process outputs 1.

Claim 44. *Assume $(1-r/4)^t q \leq 1$. For any L of size at most q and any x^*, $p_{L,x^*} \geq p_{\min} := [(1-r)r/4]^t$.*

For any polynomial q, we can therefore choose a constant $r \in [0,1]$ and $t = \lceil -\log(q)/\log(1-r/4) \rceil$, to get the probability to be $\Omega(q^{-c_r})$ for $c_r := \log[r(1-r)/4]/\log(1-r/4)$. Note that c_r is minimized when $r \approx 0.8258$, at which point $c_r \approx 14.38$. We now prove the claim.

Proof. For each j, the probability that $y^*_{i_j} \notin S_j$ is exactly $(1-r)$. Over all j, the probability that Condition 1 holds is $(1-r)^t$.

We will now condition on Condition 1 holding. In this case, each S_j is a random subset of $\mathbb{F} \setminus y^*_{i_j}$. For each i and for each $y \in M$, there are two cases:

- $y_i = y^*_i$. In this case, if $i_j = i$, then $y_{i_j} \notin S_j$ with probability 1.
- $y_i \neq y^*_i$. In this case, if $i_j = i$, then $y_{i_j} \in S_j$ with probability $r|\mathbb{F}|/(|\mathbb{F}|-1) > r$.

Since the polynomials P defining the various y have degree at most $d = \ell - 1$, any two distinct polynomials can agree on at most $d+1 = \ell$ points. Therefore, for each $y \in M$, there are at least $(2\ell) - \ell = \ell$ different i such that $y_i \neq y^*_i$. For these i, we have $\Pr[y_i \in S_j] > r$ from above. Thus, for each j, if we average over a uniform choice of i_j, we have that $\Pr[y_{i_j} \in S_j] > r/2$.

For $j = 0, \ldots, t$, let M_j be the set of $y \in M$ for which Condition 2 is not yet satisfied by the jth step. In other words, M_j is the set of y such that there does not exist a $j' \leq j$ such that $y_{i'_j} \in S_{j'}$.

$M_0 = M$. Clearly $M_{j+1} \subseteq M_j$. For each $y \in M_j$, $\Pr[y \in M_{j+1}] < 1 - r/2$. By linearity of expectation, once we've fixed M_j, we have that $\mathbb{E}[|M_{j+1}|] < (1-r/2)|M_j|$. Therefore, $\Pr[|M_{j+1}| < (1-r/4)|M_j|] \geq r/4$. Over all t trials, we therefore have that $\Pr[|M_t| < (1-r/4)^t |M|] \geq (r/4)^t$. If we choose $(1-r/4)^t q \leq 1$, then $|M_t| < (1-r/4)^t |M|$ can only be true if M_t is empty, meaning Condition 2 is met. Putting it all together, we have that the process outputs 1 with probability at least $[(1-r)r/4]^t$. □

We are now ready to describe $\mathcal{A}'$:

- First it sets parameters r, t as above. For $j = 1, \ldots, t$, it does the following:
  - Choose a random $i_j \in [2\ell]$. Then choose a random subset $S_j \subseteq \mathbb{F}$ of size $r|\mathbb{F}|$.
  - Make constrain queries on (i_j, z) for each $z \in S_j$, obtaining $k_{i_j,z}$.
- Now $\mathcal{A}'$ simulates $\mathcal{A}$, answering it's queries as follows:
  - **Constrain.** For a constrain query on (i, z), $\mathcal{A}'$ simply forwards the query to its challenger, obtaining $k_{i,z}$, which it sends back to $\mathcal{A}$.
  - **Eval.** For an Eval query on an input x, $\mathcal{A}'$ computes y from x as above. It then checks if $y_{i_j} \in S_j$ for some $j \in [t]$. If so, it runs $\mathsf{EvalC}'(k_{i_j,y_{i_j}}, y)$. If no such j is found, $\mathcal{A}'$ immediately aborts and outputs a random bit.

– **Challenge.** For the challenge query on input x^*, $\mathcal{A}'$ computes y^* from x^*. It then checks if $y^*_{i_j} \in S_j$ for some $j \in [t]$. If such a j exists, then $\mathcal{A}'$ immediately aborts and outputs a random bit. Otherwise, it forwards y^* to the challenger, and sends the response to $\mathcal{A}$.

- At the end of the experiment, $\mathcal{A}'$ lets L be the set of Eval queries. It then aborts with probability $p_{\min}/p_{L,x^*}$, and outputs a random bit. If no abort happens, then $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs.

First, if we ignore the final abort step, then $\mathcal{A}'$ aborts with probability p_{L,x^*}. Thus, the overall abort probability is $p_{\min}$, independent of the queries made.

Therefore, conditioned on no abort, $\mathcal{A}'$ perfectly simulates the view of $\mathcal{A}$, and moreover the challenge y^* is not covered by any constrain queries. Thus, conditioned on not aborting, $\mathcal{A}'$ has success probability $\frac{1}{2}+\epsilon$. The overall success probability is therefore $\frac{1}{2}+p_{\min}\epsilon$, which is non-negligible.

Remark 1. *The above description assumes that p_{L,x^*} can be computed exactly, which is not necessarily true. This is analogous to the artificial abort of Waters [Wat05]. As in [Wat05], we instead have $\mathcal{A}'$* estimate *p_{L,x^*} to within an error much less than $p_{\min}\epsilon$ by simply sunning $\mathsf{poly}(1/p_{\min}\epsilon, \lambda)$ trials of the process defining p_{L,x^*}. This will introduce an error $\ll p_{\min}\epsilon$ into the simulation.*

□

C.2 Lifting 1-SF-PRFs to Special PRFs

Here, we prove Theorem 7, showing that 1-SF-PRFs imply Special Constrained PRFs.

Perfect List-Recoverable Codes. We will need a special type of error correcting code, that we will call a perfect list recoverable code.

Definition 11. *A (Σ, s, u, n)-perfect list-recoverable code is a subset $C \subseteq \Sigma^u$ of size s with the following property. For any set $L \subseteq C$ of size at most n, let $L' \subset C$ be the set of all codewords c such that, for each $i \in [u]$, there exists a $c' \in L$ such that $c_i = c'_i$. Then $L' = L$.*

In other words, there is no way to mix and match the symbols from $\leq n$ codewords to obtain a new codeword.

Lemma 16. *For any s, n, there exists a (Σ, s, u, n)-perfect list-recoverable code with $|\Sigma|, u = \mathsf{poly}(\log s, n)$.*

Proof. Let $\Sigma = \mathbb{F}$ be a finite field and u, d be integers such that:

- $u > nd$.
- $|\mathbb{F}| > u$
- $s \leq |\mathbb{F}|^{d+1}$

Let $C = \mathbb{F}^u$ be a Reed-Solomon code: the set of vectors $c = (c_1, \ldots, c_u)$ such that $c_i = P(i)$ for a polynomial P of degree at most d.

Now consider any set $L \subseteq C$ of size at most n. Consider any other codeword c that agrees with some element of L in each position. Then there must be some $c' \in L$ such that c agrees with c' in at least $d+1$ positions. But then $c = c'$. Hence, $L' = L$. □

Lemma 17. *Let C be a perfect list-recoverable code. For any set $M \subseteq C$ of size at most $n+1$ and any $x \in M$, there exists an $j \in [u]$ such that $x'_j \neq x_j$ for all $x' \in M \setminus \{x\}$.*

Proof. Let $L = M \setminus \{x\}$. If for all $j \in [u]$ there exists an $x' \in L$ such that $x_j = x'_j$, then $x \in L'$, where L' is as defined in the definition of perfect list-recoverable code. But then $L' \neq L$, a contradiction. □

Constructing Special Constrained PRFs. We now turn to constructing our special constrained PRFs from any 1-SF-PRF. The high-level idea is to set the exponentially-large symbol space Σ of the special PRF to be codewords over the symbols space Σ' of the underlying 1-SF-PRF. To generate the initial punctured key for a set S, for each position k, we collect the set $S_j \subseteq \Sigma'$ of characters the words in S attain at position j. We then, for that position, reveal 1-SF-PRF keys constraining that position to every symbol *not* in S_j. By the perfect list-recoverability, for any codeword $c \notin S$, there must exist a j such that $c_j \notin S_j$. Therefore, these constrained keys allow for evaluating the PRF on any codeword not in S, as needed for the punctured key. Subsequent constrained keys for the special PRF are handled straightforwardly by making appropriate additional constrained key queries to the underlying 1-SF-PRF.

We now give the construction and proof in more detail. Let $(\mathsf{Gen}', \mathsf{Eval}', \mathsf{Constr}', \mathsf{EvalC}')$ be a 1-SF-PRF. Define the following Special Constrained PRF:

- $\mathsf{Gen}(1^\lambda, s, 1^\ell, 1^n)$: Let $\Sigma', u, C \subset (\Sigma')^u$ be a (Σ', s, u, n)-perfect list-recoverable code guaranteed by Lemma 16, where $|\Sigma'|, u$ have polynomial size. Set Σ to be an arbitrary subset of C of size s.

  Let $\ell' = \ell \times u$. Define $\mathsf{Map} : \Sigma^n \to (\Sigma')^{\ell u}$ as $\mathsf{Map}(x_1, \ldots, x_\ell)_{(i-1)u+j} = (x_i)_j$ for $i \in [\ell], j \in [u]$.

  Run $k \leftarrow \mathsf{Gen}'(1^\lambda, 1^{|\Sigma|}, 1^{\ell'})$.

- $\mathsf{Eval}(k, x)$: on input $x \in \Sigma^\ell$, run $\mathsf{Eval}'(k, \mathsf{Map}(x))$.

- $\mathsf{Punc}(k, S)$: Recall that S is a subset of $\Sigma \subseteq (\Sigma')^u$. For each $j \in [u]$, let $S_j \subset \Sigma'$ be the set of symbols attained in the jth position by elements of S. For each $i \in [\ell], j \in [u], z' \in \Sigma' \setminus S_j$, run $k'_{(i,j),z'} \leftarrow \mathsf{Constr}'(k, (i-1)u + j, z')$. Output $k_S = (k'_{(i,j),z'})_{i \in [\ell], j \in [u], z' \in \Sigma' \setminus S_j}$.

- $\mathsf{EvalP}(k_S, x)$: if $x \in S^\ell$, output $\perp$. Otherwise, let $y = \mathsf{Map}(x)$. There is some i such that $(y_{(i-1)u+j})_{j \in [u]} \notin S$, in which case there is a j such that $y_{i,j} \notin S_j$. Therefore run $\mathsf{EvalC}'(k_{(i,j),y_{(i-1)u+j}}, y)$.

- $\mathsf{Constr}(k, S, i, z)$: By Lemma 17, since $z \in S$ and $|S| \leq n$, there exists a j such that $y'_j \neq z_j$ for all $y' \in S \setminus \{z\}$. Output $k_{S,i,z} = k'_{(i,j),z_j}$.

- $\mathsf{EvalC}(k_{S,i,z}, x)$: If $x_i \neq z$, output $\perp$. Otherwise, run $\mathsf{EvalC}'(k'_{(i,j),z_j}, \mathsf{Map}(x))$, where j is defined as in Constr.

Correctness. For EvalP, we have that $\mathsf{EvalP}(k_S, x) = \mathsf{EvalC}'(k_{(i,j),y_{(i-1)u+j}}, y)$ for some i, j, where $y = \mathsf{Map}(x)$ and where $k'_{(i,j),y_{(i-1)u+j}} \leftarrow \mathsf{Constr}'(k, (i-1)u + j, y_{(i-1)u+j})$, which is a part of k_S. Thus, $\mathsf{EvalC}'(k_{(i,j),y_{(i-1)u+j}}, y) = \mathsf{Eval}'(k, y) = \mathsf{Eval}(k, x)$.

For EvalC, we have that $\mathsf{EvalC}(k_{S,i,z}, x) = \mathsf{EvalC}'(k'_{(i,j),z_j}, y)$ for some j, where again $y = \mathsf{Map}(x)$ and where $k'_{(i,j),z_j} \leftarrow \mathsf{Constr}'(k, (i-1)u + j, z_j)$. Here again, $k'_{(i,j),z_j}$ is a part of $k_{S,i,z}$. Thus, $\mathsf{EvalC}'(k'_{(i,j),z_j}, y) = \mathsf{Eval}'(k, y) = \mathsf{Eval}(k, x)$.

Security. We now state and prove security.

Theorem 9. *If* $(\mathsf{Gen}', \mathsf{Eval}', \mathsf{Constr}', \mathsf{EvalC}')$ *is secure, then so is* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Punc}, \mathsf{EvalP}, \mathsf{Constr}, \mathsf{EvalC})$.

Proof. Let $\mathcal{A}$ be an adversary for $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Punc}, \mathsf{EvalP}, \mathsf{Constr}, \mathsf{EvalC})$. We construct a new adversary $\mathcal{A}'$ for $(\mathsf{Gen}', \mathsf{Eval}', \mathsf{Constr}', \mathsf{EvalC}')$. $\mathcal{A}'$ simulates $\mathcal{A}$, making queries to its 1-SF-PRF challenger as needed to answer the queries made by $\mathcal{A}$. When $\mathcal{A}$ makes a challenge query on $x^* \in S^\ell$, $\mathcal{A}'$ makes a challenge query on $y^* = \mathsf{Map}(x^*)$, forwarding any response to $\mathcal{A}$.

Correctness of the simulation is straightforward. All that is needed is to show that $\mathcal{A}'$ never needs to make a query that will allow it to answer the challenge query. Recall that the queries $\mathcal{A}'$ makes to generate k_S can only evaluate the PRF on codewords that are outside the symbols of codewords in S. Therefore, these queries cannot help answering $x^* \in S$. We now address the constrained keys made by $\mathcal{A}$, for symbol z. Recall that we chose the constrained key according to Lemma 17. This means the constrained constrained key for z cannot be used to evaluate the PRF on any codeword in S other than z. In particular, since the symbols in the challenge x^* must be distinct from the queries $\mathcal{A}$ makes, none of the resulting constrained keys can evaluate x^*. Thus, $\mathcal{A}'$ is a valid adversary for 1-SF security, and perfectly simulates the view of $\mathcal{A}$ with the same advantage as $\mathcal{A}$. By the assumed 1-SF-PRF security, this advantage must therefore be negligible. □

C.3 Constructing Multi-point Binding Hashes

Here, we prove Lemma 12, which is repeated here for convenience:

Lemma 18. *Assuming one-way functions and iO exist, then so do multi-point binding hash functions.*

Proof. Let iO be an indistinguishability obfuscator, Gen' the generation algorithm for a puncturable PRF, and G a length-doubling pseudorandom generator.

- $\mathsf{Gen}(1^\lambda, 1^n)$: Sample $k \leftarrow \mathsf{Gen}'(1^\lambda)$, and output $\mathsf{hk} = \mathsf{iO}(1^\lambda, \mathsf{Hash}_k)$, where Hash_k is the program given in Figure 29, padded to the appropriate length (depending on n).
- $H(\mathsf{hk}, m) = \mathsf{hk}(m)$
- $\mathsf{GenBind}(1^\lambda, S^*)$: Write $S^* = (m_1^*, \ldots, m_n^*)$. Sample $k \leftarrow \mathsf{Gen}'(1^\lambda)$, and choose a vector of random strings $\mathbf{x} = x_1^*, \ldots, x_n^*$. Output $\mathsf{hk} = \mathsf{iO}(1^\lambda, \mathsf{HashBind}_{k,\mathbf{x}})$, where $\mathsf{HashBind}_{k,\mathbf{x}}$ is the program given in Figure 30.

Figure 29: The Program Hash_k.

Inputs: m
Constants: k

1. Output $\mathsf{G}(\mathsf{F}(k, m))$

Figure 30: The Program $\mathsf{HashBind}_{k,S^*,\mathbf{x}}$.

Inputs: m
Constants: $k, S^*, \mathbf{x}^*$

1. If $m = m_i^* \in S^*$, output x_i^*.
2. Otherwise, output $\mathsf{G}(\mathsf{F}(k, m))$

The binding property is proved as follows. For any point outside of S^*, the output is in the image of G. On the other hand, the image of G is sparse, and so all the points in S^* will, with overwhelming probability, not be in the image, and they will moreover be distinct. Therefore, the points in S^* will be the unique points with these images.

Security The indistinguishability of $(S^*, \mathsf{Gen}(1^\lambda, 1^n))$ and $(S^*, \mathsf{GenBind}(1^\lambda, S^*))$ follows via a sequence of hybrid experiments.

- Game_0: This corresponds to the case where the adversary receives $(S^*, \mathsf{Gen}(1^\lambda, 1^n))$.

  The adversary sends set S^*. The challenger chooses a puncturable PRF key k, and outputs Hash_k.

- Game_1: In this experiment, the challenger uses a PRF key punctured at the set S^*. For the points in S^*, the challenger hardwires the output.

  More formally, after the adversary sends $S^* = (m_1^*, \ldots, m_n^*)$, the challenger chooses a PRF key k and computes a punctured PRF key $k_{S^*} \leftarrow \mathsf{Punc}(k, S^*)$. For each $i \in [n]$, it sets $x_i = \mathsf{G}(\mathsf{F}(k, m_i))$. Finally, it computes an obfuscation of $\mathsf{HashBind}_{k_{S^*}, S^*, \mathbf{x}^*}$ where $\mathbf{x}^* = (x_1^*, \ldots, x_n^*)$.

- Game_2: This experiment is similar to the previous one, except that the challenger chooses n strings $\mathbf{y} = (y_1, \ldots, y_n)$ uniformly at random and sets $x_i^* = \mathsf{G}(y_i)$.

- Game_3: This experiment is similar to the previous one, except that the challenger chooses all x_i^* strings uniformly at random.

- Game_4: This experiment corresponds to the case where the adversary receives $\mathsf{GenBind}(1^\lambda, S^*)$.

Claim 45. *Assuming the security of* iO*,* Game_0 *and* Game_1 *are computationally indistinguishable.*

Proof. The only difference in Game_0 and Game_1 is that the program Hash_k is used in Game_0, while $\mathsf{HashBind}_{k_{S^*}, S^*, \mathbf{x}}$ is used in Game_1. Therefore, it suffices to show that these two programs are functionally identical (and as a result, their obfuscations are computationally indistinguishable).

For all inputs $m \notin S^*$, the programs have identical output, since $\mathsf{G}(\mathsf{F}(k, m)) = \mathsf{G}(\mathsf{F}(k_{S^*}, m))$ (using the correctness of punctured PRF on non-punctured points).

For any inputs $m_i \in S^*$, note that the program $\mathsf{HashBind}$ outputs x_i^*, and this is set to $\mathsf{G}(\mathsf{F}(k, m_i))$. As a result, the outputs are identical in this case too. This concludes our proof.

□

Claim 46. *Assuming the security of* F*,* Game_1 *and* Game_2 *are computationally indistinguishable.*

Proof. Suppose there exists an adversary that can distinguish between Game_1 and Game_2. Then there exists a reduction algorithm that can break the security of F.

The reduction algorithm receives set S^* from the adversary, and sends this to the PRF challenger. It receives a punctured key k_{S^*}, and n strings $\mathbf{y} = (y_1, \ldots, y_n)$ which are either PRF evaluations or truly random strings. The reduction algorithm computes $x_i^* = \mathsf{G}(y_i)$ for all $i \in [\ell]$, and then uses these strings and k_{S^*} to compute the obfuscation of $\mathsf{HashBind}_{S^*, k_{S^*}, \mathbf{x}^*}$.

If the y_i strings are pseudorandom, then this corresponds to Game_1, else this corresponds to Game_2. Therefore, an adversary that can distinguish between these two experiments can be used to break the PRF security. □

Claim 47. *Assuming the security of* G, Game_2 *and* Game_3 *are computationally indistinguishable.*

Proof. The only difference in the two experiments is that $x_i^* = \mathsf{G}(y_i)$ in Game_2, while x_i^* is chosen uniformly at random in Game_3. Note that the strings y_i are chosen uniformly at random in Game_2, and are not used anywhere else in the experiment. As a result, the two games are computationally indistinguishable, assuming G is a secure pseudorandom generator. □

Claim 48. *Assuming the security of* iO, Game_3 *and* Game_4 *are computationally indistinguishable.*

Proof. The only difference in the two experiments is that Game_3 uses a PRF key punctured at set S^*, while Game_4 uses $\mathsf{HashBind}$. On the set S^*, both programs use the hardwired strings $\mathbf{x}^*$. On the remaining strings, the PRF evaluation using k_{S^*} is identical to the evaluation using k. □

□

C.4 Analysis of Hybrids from Section 4.4

We now analyze the hybrids from Section 4.4.

Lemma 19. *For any adversary* $\mathcal{A}$, $\mathsf{adv}_{\mathcal{A},0} = \mathsf{adv}_{\mathcal{A},1}$.

Proof. This follows directly from the definition of the security experiments. In one case ($\mathsf{Game}_{\text{real}}$), the message m_i^* and verification key vk_i^* are chosen after receiving the i^{th} query. In Game_1, all these messages and verification keys are chosen during setup. Since the messages are chosen uniformly at random in both experiments, and vk_i^* is derived from m_i^* using $\mathsf{GenBind}_{sig}$ in both cases, the distributions are identical. □

Lemma 20. *Assuming the multi-point binding security of* H*, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ, $|\mathsf{adv}_{\mathcal{A},1} - \mathsf{adv}_{\mathcal{A},2}| \leq \mathsf{negl}(\lambda)$.

Proof. The only difference between Game_1 and Game_2 is the choice of hk. In one case, it is computed using $\mathsf{Gen}_{Hash}(1^\lambda, 1^n)$, while in the other case, it is computed using $\mathsf{GenBind}_{Hash}(1^\lambda, \{\mathsf{vk}_i\}_{i\in[n]})$. These two hash keys are computationally indistinguishable, assuming the multi-point binding security of H. □ □

Lemma 21. *Assuming the security of* iO*, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ, $|\mathsf{adv}_{\mathcal{A},2} - \mathsf{adv}_{\mathcal{A},3}| \leq \mathsf{negl}(\lambda)$.

Proof. The only difference in the two experiments is the CRS. In one case, the CRS is an obfuscation of $\mathsf{KGen}_{\mathsf{hk},k}$, while in the other case, it is an obfuscation of $\mathsf{KGenAlt}_{\mathsf{hk},\{u_j^*,v_j^*,K_j^*\},K_S}$. We will show that these two programs (with appropriate hardwired constants) are functionally identical, and therefore their obfuscations are computationally indistinguishable. We will use the multi-input binding property of H, (single-input) binding property of signature scheme and the correctness of the constrained/punctured key's evaluation.

Recall, $\left\{m_j^*, \mathsf{vk}_j^*, \sigma_j^*, u_j^*, v_j^*\right\}_{j\in[n]}$ are computed/chosen by the challenger during setup. We will consider the following cases to show that the two programs are functionally identical:

- $\forall j$, $\mathsf{vk}_j = \mathsf{vk}_j^*$ and $m_i = m_i^*$ and $(\mathsf{vk}_i^*, m_i^*, \sigma_i)$ verifies: in this case, the two programs are identical. The first program outputs PRF evaluation using the key k, while the second program uses the vector of constrained keys K_i^*. Note that this key can evaluate at all points $(x_1, \ldots, x_n)$ where $x_i \in \{u_j^*\}_{j \in [n]}$, and in this program, the input to the constrained evaluation algorithm satisfies the constraint.

- $\forall j$, $\mathsf{vk}_j = \mathsf{vk}_j^*$ but either $m_i \neq m_i^*$ or $(\mathsf{vk}_i^*, m_i^*, \sigma_i)$ does not verify: In this case, both the programs reject. This is due to the binding property of the signature scheme. If $\mathsf{vk}_i = \mathsf{vk}_i^*$, then $\nexists\, m \neq m_i^*$ with a verifying signature.

- If there exists a t such that $\mathsf{vk}_t \notin \{\mathsf{vk}_j^*\}_{j \in [n]}$ (say $t^* \in [\ell]$ is the smallest such index), then using the multi-input binding property of the hash function, there does not exist a vk_{t^*} such that $H(\mathsf{hk}, \mathsf{vk}_{t^*}) = H(\mathsf{hk}, \mathsf{vk}_{t^*}^*) = u_{t^*}^*$. Therefore, the string $(\ H(\mathsf{hk}, \mathsf{vk}_j)\ )_j \notin \left(\left\{u_j^*\right\}_j\right)^{\ell}$. As a result, the punctured key can be used to perform the PRF evaluation at this point. □

□

Lemma 22. *Assuming the security of single-point binding signatures, for any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that for all λ, $|\mathsf{adv}_{\mathcal{A},3} - \mathsf{adv}_{\mathcal{A},4}| \leq \mathsf{negl}(\lambda)$.*

Proof. The only difference between the two experiments is the choice of vk_i^* and σ_i^*. In one case, the verification keys and signatures are chosen using $\mathsf{GenBind}_{sig}$, while in the other case, they are chosen using Gen_{sig} (that is, in non-binding mode). These two experiments are computationally indistinguishable, assuming the security of single-point binding signatures.

In particular, we define n intermediate hybrid experiments $\{\mathsf{Game}_{3,w}\}_{w \in [n]}$ between Game_3 and Game_4. In $\mathsf{Game}_{3,w}$, the first w verification keys are chosen using Gen_{sig}, while the remaining are chosen using $\mathsf{GenBind}_{sig}$. Suppose $|\mathsf{adv}_{\mathcal{A},3} - \mathsf{adv}_{\mathcal{A},4}|$ is non-negligible, then there exists an index $w \in [\ell]$ such that $|\mathsf{adv}_{\mathcal{A},3,w} - \mathsf{adv}_{\mathcal{A},3,w+1}|$ is non-negligible. We can use $\mathcal{A}$ to build a reduction algorithm $\mathcal{B}$ that breaks the security of single-point binding signatures. The reduction algorithm samples the first w (message, verification key, signature) tuples as in $\mathsf{Game}_{3,w}$. For the $(w+1)^{\text{th}}$ tuple, it chooses a uniformly random message m_{w+1}^* and sends it to the challenger. The challenger sends a verification key vk^* and σ^*. The reduction algorithm sets $\mathsf{vk}_{w+1}^* = \mathsf{vk}^*$ and $\sigma_{w+1}^* = \sigma^*$. The remaining (verification key, signature) pairs are chosen using $\mathsf{GenBind}_{sig}$. Once all verification keys are chosen, the reduction algorithm chooses hk as in Game_3, computes the crs and sends it to the adversary.

The honest user registration queries are answered using the verification keys, while the corruption queries are answered using the (message, signature) pairs. Note that if vk^* is in binding mode, then this corresponds to $\mathsf{Game}_{3,w}$, else it corresponds to $\mathsf{Game}_{3,w+1}$. Therefore, an adversary distinguishing between these two experiments can be used to break the single-point binding security of the signature scheme. □

Lemma 23. *For any adversary $\mathcal{A}$, $\mathsf{adv}_{\mathcal{A},4} = \mathsf{adv}_{\mathcal{A},5}$.*

Proof. This follows directly from the definition of the experiments, since the difference in the two experiments is syntactic. In Game_4, the challenger chooses m_j^* uniformly at random, and sets v_j^* as $m_j^* \oplus K_j^*$. In Game_5, it chooses v_j^* uniformly at random, and sets $m_j^* = v_j^* \oplus K_j^*$. The remaining experiments are identical. □

Lemma 24. *Assuming* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Punc}, \mathsf{EvalP}, \mathsf{Constr}, \mathsf{EvalC})$ *is an adaptively secure special constrained PRF, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ*,* $|\mathsf{adv}_{\mathcal{A},5}| \leq \mathsf{negl}(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ with non-negligible advantage in Game_5. We will use this adversary to build an algorithm $\mathcal{B}$ to break the adaptive security of the special constrained PRF scheme.

First, the algorithm $\mathcal{B}$ simulates the setup phase. It chooses $(\mathsf{sk}_j^*, \mathsf{vk}_j^*)$ for each $j \in [n]$, samples hk and computes $u_j^* = H(\mathsf{hk}, \mathsf{vk}_j^*)$. The reduction sets $S = \left\{u_j^*\right\}$, and queries for a punctured PRF key for set S. It receives K_S, chooses v_j^* at random, and sends an obfuscation of $\mathsf{KGenAlt}_{\mathsf{hk},\left\{u_j^*, v_j^*\right\},K_S}$ as the CRS.

Next, the reduction receives pre-challenge queries. For the i^{th} registration query, the challenger queries for a constrained key for (S, i, u_i^*). It receives K_i^* from the challenger, and sets $m_i^* = v_i^* \oplus K_i^*$. It computes a signature σ_i^* on m_i^* and sends $(\mathsf{vk}_i^*, (m_i^*, \sigma_i^*))$ to the adversary. The reduction algorithm also adds $(\mathsf{vk}_i^*, (m_i^*, \sigma_i^*), 0)$ to T.

When the adversary queries for corrupting a user, the reduction algorithm sends the corresponding secret key. And when the adversary queries to register a malicious user, the adversary sends pk, and the reduction algorithm adds $(\mathsf{pk}, \perp, 1)$ to T.

For responding to shared key queries, the reduction algorithm sends evaluation queries to the challenger.

Finally, the reduction receives the challenge set $(\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$. It computes $u_i = H(\mathsf{hk}, \mathsf{vk}_i)$ and sends $u_1 || \ldots || u_\ell$ to the challenger, and forwards the challenger's response to the adversary.

The post challenge queries are handled similar to the pre-challenge queries. Finally, the adversary sends a guess, which is forwarded to the PRF challenger. □ □

D Missing Proofs from Section 5

Here, we complete the proof of Lemma 14, by showing that the various hybrids are indistinguishable.

Claim 49. *For any* $y \in [Q]$*, and any adversary* $\mathcal{A}$ *making at most* Q *constrained key queries,* $\mathsf{adv}_{\mathcal{A},y,a} = \mathsf{adv}_{\mathcal{A},y,b}$.

Proof. The only differences in the two games are the following:

- In $\mathsf{Game}_{y,a}$, the challenger chooses all $\{h_j\}_{j\in[y]}$ uniformly at random from $\mathbb{G}$. In $\mathsf{Game}_{y,b}$, the challenger chooses $c_j \leftarrow \mathbb{Z}_p$ and sets $h_j = g^{c_j}$ for each $j \in [y]$. The distributions of these y group elements is identical in both the experiments.
- In $\mathsf{Game}_{y,a}$, the challenger chooses $v \leftarrow \mathbb{G}$, while in $\mathsf{Game}_{y,b}$ the challenger chooses $a \leftarrow \mathbb{Z}_p$ and sets $v = g^a$. The distribution of v is identical in both experiments.
- In both the games, all $e_{j,w}$ elements are uniformly random elements in $\mathbb{Z}_p$. Finally, we define $B = g^{e_{\mathsf{index}_{j+1},\mathsf{sym}_{j+1}}}$ and $T = v^{e_{\mathsf{index}_{j+1},\mathsf{sym}_{j+1}}}$.

□

Claim 50. *Fix any $y \in [Q]$. Assuming iO is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$ making Q constrained key queries, there exists a negligible function negl such that for all λ, $|\mathsf{adv}_{\mathcal{A},y,b} - \mathsf{adv}_{\mathcal{A},y,c}| \leq \mathsf{negl}(\lambda)$.*

Proof. The only difference in the two hybrids is the challenger's response to the last $Q-y$ constrained key queries. Fix any $q \in \{y+1, \ldots, Q\}$. We will show that the programs obfuscated in the two experiments (in response to the q^{th} query) are functionally identical, and have the same size. As a result, we can use the security of iO to argue that the q^{th} constrained key is indistinguishable in both the games. Therefore, we can conclude that $|\mathsf{adv}_{\mathcal{A},y,b} - \mathsf{adv}_{\mathcal{A},y,c}|$ is bounded by $(Q-y) \cdot \mathsf{adv}_{\mathsf{iO}}$, which is negligible if $\mathsf{adv}_{\mathsf{iO}}$ is negligible.

Proving that the programs are functionally identical: Take any input $x \in \Sigma^\ell$. Note that both the programs have list L_y which consists of the first y constrained key queries $\left((\mathsf{index}_j, \mathsf{sym}_j)\right)_{j \leq y}$. Additionally, the program in $\mathsf{Game}_{y,c}$ has $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$. We consider the following cases:

1. there exists a $j \in [y]$ such that $x_{\mathsf{index}_j} = \mathsf{sym}_j$. In this case, $\mathsf{ConstrainedKeyAlt}$ outputs $h_j^{\prod_l e_{l,x_l}}$. The program $\mathsf{ConstrainedKeyAlt}'$ has two cases:

   - if $x_{\mathsf{index}_{y+1}} = \mathsf{sym}_{y+1}$, then the program outputs $(g^{\prod_l e_{l,x_l}})^{c_j} = h_j^{\prod_l e_{l,x_j}}$. Here, we use the fact that A is set to $g^{e_{\mathsf{index}_{y+1},\mathsf{sym}_{y+1}}}$ and $h_j = g^{c_j}$.
   - if $x_{\mathsf{index}_{y+1}} \neq \mathsf{sym}_{y+1}$, then the program outputs $(g^{\prod_l e_{l,x_l}})^{c_j} = h_j^{\prod_l e_{l,x_j}}$. Here, we only use the fact that $h_j = g^{c_j}$.

2. there does not exist a $j \in [y]$ such that $x_{\mathsf{index}_j} = \mathsf{sym}_j$. In this case, the first program outputs $v^{\prod_l e_{l,x_l}}$. The second program again has two cases:

   - if $x_{\mathsf{index}_{y+1}} = \mathsf{sym}_{y+1}$, then the program outputs $(v^{\prod_l e_{l,x_l}})$. Here, we use the fact that T is set to $v^{e_{\mathsf{index}_{y+1},\mathsf{sym}_{y+1}}}$.
   - if $x_{\mathsf{index}_{y+1}} \neq \mathsf{sym}_{y+1}$, then the program outputs $(v^{\prod_l e_{l,x_l}})$. The computation is identical in both cases.

As a result, the two programs are functionally identical (assuming the constants are hardwired as described in $\mathsf{Game}_{y,b}$ and $\mathsf{Game}_{y,c}$). □

Claim 51. *Fix any $y \in [Q]$. Assuming DDH , for any PPT adversary $\mathcal{A}$ making Q constrained key queries, there exists a negligible function negl such that for all λ, $|\mathsf{adv}_{\mathcal{A},y,c} - \mathsf{adv}_{\mathcal{A},y,d}| \leq \mathsf{negl}(\lambda)$.*

Proof. The only difference in the two hybrids is that $A = g^a$, $v = g^b$ and $T = v^a$ in $\mathsf{Game}_{y,c}$, and in $\mathsf{Game}_{y,d}$, $A = g^a$, $v = g^b$ and $T \leftarrow \mathbb{G}$. If there exists an adversary that can distinguish between these two games with advantage ϵ, then there exists an algorithm that can break the DDH assumption. □

Claim 52. *For any $y \in [Q]$, and any adversary $\mathcal{A}$ making at most Q constrained key queries, $|\mathsf{adv}_{\mathcal{A},y,d} = \mathsf{adv}_{\mathcal{A},y,e}| \leq 1/p$.*

Proof. The only difference in the two hybrids is that $T \leftarrow \mathbb{G}$ in $\mathsf{Game}_{y,d}$, and in $\mathsf{Game}_{y,f}$, we first choose $h_{j+1} \leftarrow \mathbb{G}$ and set $T = h_{j+1}^a$. Since h_{j+1} is uniformly random, and a is non-zero with overwhelming probability, the two distributions are negligibly close. □

Claim 53. *Fix any $y \in [Q]$. Assuming iO is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$ making Q constrained key queries, there exists a negligible function negl such that for all λ, $|\mathsf{adv}_{\mathcal{A},y,e} - \mathsf{adv}_{\mathcal{A},y,f}| \leq \mathsf{negl}(\lambda)$.*

Proof. This proof is very similar to the proof of Claim 50. Again, the only difference in the two games is in the response to the last $Q - y$ key queries. We will show that for each of these queries, the programs obfuscated in the two experiments are functionally identical.

Fix any $q \in \{y+1, \ldots, Q\}$. Below we show that the q^{th} constrained key is functionally identical in both the experiments. Let $h_j = g^{c_j}$ for all $j \leq y$, $H_y = (h_j)_{j \leq y}$, $H_{y+1} = (h_j)_{j \leq y+1}$, $L_y = \left((\mathsf{index}_j, \mathsf{sym}_j)\right)_{j \leq y}$, $L_{y+1} = \left((\mathsf{index}_j, \mathsf{sym}_j)\right)_{j \leq y+1}$, $v = g^b$, $e_{\mathsf{index}_{y+1}, \mathsf{sym}_{y+1}} = a$, $A = g^a$, $T = h_{j+1}^a$. Finally, let $q^* = (\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$.

- In $\mathsf{Game}_{y,e}$, the challenger uses $P_1 \equiv \mathsf{ConstrainedKeyAlt}'_{y, L_y, q^*, \{c_j\}, g, v, A, T, (e_{j,w})_{(j,w) \neq q^*}, i, z}$.
- In $\mathsf{Game}_{y,f}$, the challenger uses $P_2 \equiv \mathsf{ConstrainedKeyAlt}_{y+1, L_{y+1}, H_{y+1}, v, (e_{j,w}), i, z}$.

Proving that the programs are functionally identical: Consider any input $x \in \Sigma^\ell$. As in the proof of Claim 50, we will have the following cases:

1. there exists a $j \in [y+1]$ such that $x_{\mathsf{index}_j} = \mathsf{sym}_j$. In this case, P_2 outputs $h_j^{\prod_l e_{l,x_l}}$. The program P_1 has the following cases:
   - if $x_{\mathsf{index}_{y+1}} = \mathsf{sym}_{y+1}$ and $j < y+1$, then the program outputs $(g^{\prod_l e_{l,x_l}})^{c_j} = h_j^{\prod_l e_{l,x_j}}$. Here, we use the fact that A is set to $g^{e_{\mathsf{index}_{y+1}, \mathsf{sym}_{y+1}}}$ and $h_j = g^{c_j}$.
   - if $x_{\mathsf{index}_{y+1}} = \mathsf{sym}_{y+1}$ and $j = y+1$, then the program executes Step 3(c). Note that this is equal to $(h_{j+1}^a)^{\prod_{l \neq \mathsf{index}_{y+1}} e_{l,x_l}} = (h_{j+1})^{\prod_l e_{l,x_l}}$. Here, we use the fact that $T = h_{j+1}^a$.
   - if $x_{\mathsf{index}_{y+1}} \neq \mathsf{sym}_{y+1}$ and $j < y+1$, then the program outputs $(g^{\prod_l e_{l,x_l}})^{c_j} = h_j^{\prod_l e_{l,x_j}}$. Here, we only use the fact that $h_j = g^{c_j}$.
2. there does not exist a $j \in [y+1]$ such that $x_{\mathsf{index}_j} = \mathsf{sym}_j$. In this case, P_2 outputs $v^{\prod_l e_{l,x_l}}$. The program P_1 again has two cases:
   - if $x_{\mathsf{index}_{y+1}} = \mathsf{sym}_{y+1}$, then the program outputs $(v^{\prod_l e_{l,x_l}})$. Here, we use the fact that T is set to $v^{e_{\mathsf{index}_{y+1}, \mathsf{sym}_{y+1}}}$.
   - if $x_{\mathsf{index}_{y+1}} \neq \mathsf{sym}_{y+1}$, then the program outputs $(v^{\prod_l e_{l,x_l}})$. The computation is identical in both cases.

□

Claim 54. *For any $y \in [Q]$, and any adversary $\mathcal{A}$ making at most Q constrained key queries, $\mathsf{adv}_{\mathcal{A},y,f} = \mathsf{adv}_{\mathcal{A},y,g}$.*

Proof. This proof is similar to the proof of Claim 49. The changes in the two experiments are syntactic, and therefore the distributions are identical.

□

From the above claims, it follows that for any PPT adversary $\mathcal{A}$, $|\mathsf{adv}_{\mathcal{A},y} - \mathsf{adv}_{\mathcal{A},y+1}| \leq \ell \cdot |\Sigma| \cdot (2\mathsf{adv}_{\mathsf{iO}} + \mathsf{adv}_{\mathsf{DDH}}) \leq \mathsf{negl}$.

Finally, we show that any adversary $\mathcal{A}$ has negligible advantage in Game_Q.

Claim 55. *For any adversary $\mathcal{A}$, there exists a negligible function negl such that for all λ, $\mathsf{adv}_{\mathcal{A},Q} \leq \mathsf{negl}(\lambda)$.*

Proof. First, let us recall Game_Q.

Game_Q: In this game, the challenger uses the program $\mathsf{ConstrainedKeyAlt}$ for all the constrained keys.

- **Setup Phase:** The challenger chooses $v \leftarrow \mathbb{G}$, $h_k \leftarrow \mathbb{G}$ for all $k \in [Q]$ and $e_{k,w} \leftarrow \mathbb{Z}_p$ for all $k \in [\ell], w \in \Sigma$. Let $H = (h_k)_{k\in[Q]}$.

  The challenger also maintains an ordered list L of $(\mathsf{index}, \mathsf{sym})$ pairs which is initially empty.

- **Pre-challenge queries:** Next, the challenger receives pre-challenge constrained key queries. Let $(\mathsf{index}_j, \mathsf{sym}_j)$ be the j^{th} constrained key query. The challenger adds $(\mathsf{index}_j, \mathsf{sym}_j)$ to L.

  Let $s = \min(Q, j) = j$, and let L_s (resp. H_s) denote the first s entries in L (resp. H). The challenger computes the constrained key
  $\mathsf{K}_j \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKeyAlt}_{s,L_s,H_s,v,(e_{k,w}),\mathsf{index}_j,\mathsf{sym}_j})$ and sends K_j to the adversary.

- **Challenge Phase:** Next, the adversary sends a challenge x^* such that $x^*_i \neq z$ for any pre-challenge key query (i, z). The challenger chooses $b \leftarrow \{0, 1\}$. If $b = 0$, the challenger computes $t = \prod_i e_{i,x^*_i}$ and sends v^t. If $b = 1$, the challenger sends a uniformly random group element in $\mathbb{G}$.

- **Post-challenge queries:** The post-challenge queries are handled similar to the pre-challenge queries.

- **Guess:** Finally, the adversary sends the guess b' and wins if $b = b'$.

We will define a new security game $\mathsf{Game}_{\text{end}}$, show that it is computationally indistinguishable from Game_Q, and finally show that any adversary has negligible advantage in $\mathsf{Game}_{\text{end}}$.

$\mathsf{Game}_{\text{end}}$: In this game, the challenger uses the program $\mathsf{ConstrainedKeyEnd}$ for all the constrained keys. The only difference between the programs $\mathsf{ConstrainedKeyAlt}$ and $\mathsf{ConstrainedKeyEnd}$ is that the latter does not have the Step 3(b), and therefore does not contain the constant v.

- **Setup Phase:** The challenger chooses $h_k \leftarrow \mathbb{G}$ for all $k \in [Q]$ and $e_{k,w} \leftarrow \mathbb{Z}_p$ for all $k \in [\ell], w \in \Sigma$. Let $H = (h_k)_{k\in[Q]}$.

  The challenger does not sample $v \leftarrow \mathbb{G}$.

  The challenger also maintains an ordered list L of $(\mathsf{index}, \mathsf{sym})$ pairs which is initially empty.

$\mathsf{ConstrainedKeyAlt}_{s,L_s,H_s,v,(e_{k,w}),i,z}$

Input: $x = (x_1, \ldots, x_\ell) \in \Sigma^\ell$
Constants: $s \in \ell \cdot |\Sigma|$
List $L_s = ((\mathsf{index}_k, \mathsf{sym}_k))_{k\in[s]}$
$H_s = (h_k)_{k\in[s]}$
Group element v,
Exponents $(e_{k,w})_{k,w}$,
Constraining index/symbol $i \in [\ell], z \in \Sigma$

1. If $x_i \neq z$ output $\bot$.
2. Compute $t = \left(\prod_{k\leq\ell} e_{k,x_k}\right)$.
3. Find the smallest $k \in [s]$ such that $x_{\mathsf{index}_k} = \mathsf{sym}_k$.
   (a) If such k exists, then output h_k^t.
   (b) Else output v^t.

Figure 31: Program $\mathsf{ConstrainedKeyAlt}$

- **Pre-challenge queries:** Next, the challenger receives pre-challenge constrained key queries. Let $(\mathsf{index}_j, \mathsf{sym}_j)$ be the j^{th} constrained key query. The challenger adds $(\mathsf{index}_j, \mathsf{sym}_j)$ to L.

  Let $s = \min(Q, j) = j$, and let L_s (resp. H_s) denote the first s entries in L (resp. H). The challenger computes the constrained key
  $\mathsf{K}_j \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKeyEnd}_{s,L_s,H_s,(e_{k,w}),\mathsf{index}_j,\mathsf{sym}_j})$ and sends K_j to the adversary.

- **Challenge Phase:** Next, the adversary sends a challenge x^* such that $x_i^* \neq z$ for any pre-challenge key query (i, z). The challenger chooses $b \leftarrow \{0, 1\}$.

  If $b = 0$, the challenger samples $v \leftarrow \mathbb{G}$, computes $t = \prod_i e_{i,x_i^*}$ and sends v^t. If $b = 1$, the challenger sends a uniformly random group element in $\mathbb{G}$.

- **Post-challenge queries:** The post-challenge queries are handled similar to the pre-challenge queries.

- **Guess:** Finally, the adversary sends the guess b' and wins if $b = b'$.

To show that Game_Q and $\mathsf{Game}_{\text{end}}$ are computationally indistinguishable, it suffices to show that the Q constrained keys are computationally indistinguishable. We will show that for each $j \in [Q]$, the programs $\mathsf{ConstrainedKeyAlt}_{j,L_j,H_j,v,(e_{k,w}),i,z}$ and $\mathsf{ConstrainedKeyEnd}_{j,L_j,H_j,(e_{k,w}),i,z}$ are functionally identical. Note that the only difference in the two programs is that $\mathsf{ConstrainedKeyEnd}$ does not have an 'else' condition in Step 3(b). However, note that $x_{\mathsf{index}_j} = \mathsf{sym}_j$, and as a result, Step 3(b) is never executed in $\mathsf{ConstrainedKeyAlt}$. As a result, both the programs are functionally identical, and hence their obfuscations are computationally indistinguishable.

Finally, we need to show that any adversary has negligible advantage in $\mathsf{Game}_{\text{end}}$. This follows from the fact that v is chosen uniformly at random in the challenge phase, and v is not used anywhere else in the experiment. As a result, the adversary receives a random group element in both cases (when $b = 0$ and $b = 1$). Therefore, the adversary has advantage 0 in $\mathsf{Game}_{\text{end}}$.

$\mathsf{ConstrainedKeyEnd}_{s,L_s,H_s,(e_{k,w}),i,z}$

Input: $x = (x_1, \ldots, x_\ell) \in \Sigma^\ell$
Constants: $s \in n \cdot |\Sigma|$ List $L_s = ((\mathsf{index}_k, \mathsf{sym}_k))_{k\in[s]}$
$H_s = (h_k)_{k\in[s]}$
Exponents $(e_{k,w})_{k,w}$,
Constraining index/symbol $i \in [\ell], z \in \Sigma$

1. If $x_i \neq z$ output $\perp$.
2. Compute $t = \left(\prod_{k\leq\ell} e_{k,x_k}\right)$.
3. Find the smallest $k \in [s]$ such that $x_{\mathsf{index}_k} = \mathsf{sym}_k$.
   (a) If such k exists, then output h_k^t.

Figure 32: Program ConstrainedKeyEnd

□