HiSE: Hierarchical (Threshold) Symmetric-key Encryption

Pousali Dey
Indian Statistical Institute
Kolkata, India
deypousali95@gmail.com

Pratyay Mukherjee
Supra Research
Kolkata, India
pratyay85@gmail.com

Swagata Sasmal
Indian Statistical Institute
Kolkata, India
swagata.sasmal@gmail.com

Rohit Sinha
Swirlds Labs
Fremont, USA
sinharo@gmail.com

ABSTRACT

Threshold symmetric encryption (TSE), introduced by Agrawal et al. [DiSE, CCS 2018], provides scalable and decentralized solution for symmetric encryption by ensuring that the secret-key stays distributed at all times. They avoid having a single point of attack or failure, while achieving the necessary security requirements. TSE was further improved by Christodorescu et al. [ATSE, CCS 2021] to support an amortization feature which enables a "more privileged" client to encrypt records in bulk by interacting only once with the key servers, while decryption must be performed individually for each record, potentially by a "less privileged" client. However, typical enterprises collect or generate data once and query it several times over its lifecycle in various data processing pipelines; i.e., enterprise workloads are often decryption heavy! ATSE does not meet the bar for this setting because of linear interaction / computation (in the number of records to be decrypted) – our experiments show that ATSE provides a sub-par throughput of a few hundred records / sec.

We observe that a large class of queries read a subsequence of records (e.g. a time window) from the database. With this access structure in mind, we build a new TSE scheme which allows for both encryption and decryption with flexible granularity, in that a client's interactions with the key servers is at most logarithmic in the number of records. Our idea is to employ a binary-tree access structure over the data, where *only one* interaction is needed to decrypt all ciphertexts within a sub-tree, and thus only log-many for any arbitrary size sub-sequence. Our scheme incorporates ideas from binary-tree encryption by Canetti et al. [Eurocrypt 2003] and its variants, and carefully merges that with Merkle-tree commitments to fit into the TSE setting. We formalize this notion as *hierarchical threshold symmetric-key encryption* (HiSE), and argue that our construction satisfies all essential TSE properties, such as correctness, privacy and authenticity with respect to our definition. Our analysis relies on a well-known XDH assumption and a new assumption, that we call ℓ-masked BDDH, over asymmetric bilinear pairing in the programmable random oracle model. We also show that our new assumption does hold in generic group model.

We provide an open-source implementation of HiSE. For practical parameters, we see 65× improvement in latency and throughput over ATSE. HiSE can decrypt over 6K records / sec on server-grade hardware, but the logarithmic overhead in HiSE's encryption (not decryption) only lets us encrypt up to 3K records / sec (about 3-4.5× slowdown) and incurs roughly 500 bytes of ciphertext expansion per record – while reducing this penalty is an important future work, we believe HiSE can offer an acceptable tradeoff in practice.

KEYWORDS

Threshold Cryptography, Distributed Encryption

1 INTRODUCTION

Consumer-facing applications, such as payments processing or ads, collect many billions of events everyday, in order to perform a variety of downstream analytics on them. Since this data is business sensitive and often containing personally-identifiable information, these applications protect the data at rest using key management systems (KMS) that are often backed by hardware security modules (HSMs), which are deployed on-premise or in the cloud (e.g. AWS KMS). While HSMs are inexpensive to procure, they require dedicated facilities with operational costs, have complex update procedures (e.g. with designated trusted personnel) and are susceptible to hardware side-channels [30, 32].

Threshold Key Management System. To address these shortcomings of HSMs, there is growing interest [31] in the use of threshold cryptosystems, where the secret key material is distributed across multiple (commodity) servers under a secret-sharing scheme, and *never* reconstructed during use in encryption or decryption queries; i.e., the security of the system rests on the assumption that the attacker is unable to compromise a significant (configured threshold) fraction of the servers. Since they use commodity machines, threshold cryptosystems can allow the KMS to scale to enterprise-level workloads at low costs. Due to these obvious benefits, threshold cryptosystems have made their way into an increasing number of commercial products – recent examples include Hashicorp Vault [3], Coinbase Custody [1], and the HSM replacements by Unbound Tech [2] – and is being standardized in an ongoing workstream by the U.S. National Institute of Standards and Technology (NIST) [16].

Threshold Symmetric-Key Encryption. Targeting the use case of enterprise data protection, we focus our attention on threshold symmetric-key encryption (TSE). In the symmetric-key setting, the application must interact with the KMS to encrypt or decrypt any data[1], which lets us ensure *authenticity* of the encrypted data and also enforce *fine-grained access control* policies. In contrast, public-key encryption schemes allow any actor to encrypt data,

[1] the KMS authenticates the application prior to responding to encryption or decryption queries, and can also maintain an audit trail of all queries.

which exposes the application to attacks such as data corruption or poisoning. Existing TSE schemes [5, 7, 20] follow a basic schemata:

- A setup phase establishes a threshold secret-sharing of the private key. This setup is either implemented as a ceremony wherein a trusted admin or group of admins provision a share to each KMS server, or as a distributed key generation protocol.
- To encrypt a record, the client application interacts with a threshold number of KMS servers, who evaluate a (variant of) distributed pseudo-random function (DPRF) using the above shares to derive a specific key that is bound to that record.
- To decrypt any record, the client application must again interact with a threshold number of KMS servers, using a component of the ciphertext as input to the DPRF function to derive the same key material (as in the encryption step above).

Limitations. The first TSE scheme in literature, DiSE [7], required the application to interact with the KMS servers for encrypting or decrypting *each* message, which posed a clear scaling bottleneck[2]. Payment processors are expected to handle several thousands of data records every second[3], whereas (the maliciously secure variant of) DiSE can process at most a few hundred operations per second, even when deployed in a LAN setting. Addressing this shortcoming of DiSE, ATSE [20] allowed the client to perform bulk encryption, where the client commits to a large chunk of records that it is encrypting, and only interacts once with the KMS servers based on that commitment – effectively, ATSE provides the same authenticity guarantee as DiSE. Nevertheless, decryption in ATSE still mandates interaction for each record. While this type of access control suffices in certain settings, in many other settings it becomes problematic: for example, enterprise workloads are often decryption-heavy, as data is created once but is queried and analyzed several times over its lifetime. For a threshold KMS to scale to enterprise workloads, this number of interactions is just not affordable – the interaction is not only expensive in terms of communication (specifically, bandwidth), but also the computation on both the client and server (see Table 1 and Section 7).

Our Key Observation. Our exploration in efficient decryption starts with an observation about a large class of queries in typical enterprise workloads, where a consumer-facing service stores event-level data as it collects them from its users, in a time-ordered sequence. We find that most large-data *queries operate on a subset of data within a time window.* For instance, a payment processor or ads platform may query for transactions in a specific one-hour window for computing various aggregate statistics. We can leverage this inherent structure or partial ordering in the access pattern structure to perform sub-linear number of interactions – ideally, and as achieved in HiSE, we can decrypt an entire, contiguous (sub-) sequence of records with only (at most) a logarithmic number of server interactions (in the number of records),[4] each interaction requiring constant work and bandwidth[5].

Objectives. To summarize our requirement, we seek a TSE scheme that has the following properties:

1. **Bulk Encryption**: when encrypting a large set of records, the KMS interaction (in terms of bandwidth and server computation) does not depend on the number or size of the records; i.e., the server performs constant (and concretely efficient) work. In HiSE, the client sends to the KMS servers a short commitment computed over the set of messages, and the servers return a common commitment-specific key to encrypt and locally authenticate all messages together. This lets us attain the same authenticity property as DiSE (and ATSE), but without the need for interaction on each individual record.
2. **Bulk Subsequence Decryption**: when decrypting any contiguous range of records from the entire dataset, the KMS interaction (in terms of bandwidth and server computation) is sub-linear, ideally logarithmic in the worst case, in the number of records.
3. **Authenticity and Access Control**: Any valid ciphertext can only be produced by interacting with a threshold number of KMS servers – specifically, a valid ciphertext must encrypt a message that is contained within the set of messages committed to by the encrypting client, within the interactive protocol. This property implies that the key material given by the KMS server(s) to the encrypting application is bound to the set of messages represented by the commitment. We formalize this property via a game-based definition in Section 5.[6] Dual to authenticity is the requirement of fine-grained access control: the decrypting client must only be able to decrypt messages for which the KMS server(s) issue the key material in the interactive protocol.

1.1 Contributions

Definition. We put forward and formalize the notion of hierarchical threshold symmetric-key encryption (HiSE), which enables bulk encryption and subsequence decryption in a threshold manner. We capture the various important properties using game-based definitions in Section 5.

Construction. We provide an efficient HiSE construction based on asymmetric bilinear pairing. We show that our construction meets our definitions assuming standard XDH assumption and a new assumption, called ℓ-masked BDDH, that we introduce here. In Appendix G we show that this new assumption holds in the generic

[2]As demonstrated in [20], even with optimizations such as batching requests to save on roundtrip communication, and using high-bandwidth and low-latency network links, DiSE provides orders-of-magnitude lower performance than what the workload requires. This is because DiSE imposes a heavy compute requirement on both the KMS servers (due to linear number of group operations for PRF and NIZK proofs) and the client (due to linear number of DPRF reconstructions and NIZK verifications).

[3]Visa processes 5K TPS on average, with a maximum capacity of 65K TPS [4].

[4]Note that, in the best case, the number of interactions may be as little as constant. For example, if all ciphertexts are exactly within a sub-access structure, then only one interaction is needed.

[5]A reader may observe that this requirement can be easily met by allowing the application to use a long-term symmetric key that encrypts large portions of the dataset, and protecting that key with the KMS, so a single KMS interaction will suffice. However, in this design, any application that needs to decrypt any record in the dataset must be given the long-term key, and there is no way for the KMS to enforce fine-grained access control as the application can decrypt any number of records. Worse, the compromise of *any* such application gives the attacker access to the entire dataset.

[6]We note that, like prior works [7, 20] our formalization also considers a "one-more type" definition – a malicious client may not produce more ciphertexts than what is accounted for by the honest servers.

group model. The concrete benefits of our HiSE scheme over prior works is characterized in Table 1.

Implementation and Evaluation. We provide an open-source implementation of our HiSE construction, made available at https://github.com/rsinha/hise. Our experiments indicate latency reduction between 15-65× and throughput improvement between 10-70×, compared to the ATSE decryption, when interacting with 6-24 KMS servers and decrypting hundreds to thousands of records in bulk. HiSE can decrypt over 6K records / sec on a single server-grade machine. Moreover, as we scale to larger workloads which decrypt a large number of records, both network latency and server computation becomes an increasingly insignificant fraction of the total decryption running time in HiSE, thus incentivizing threshold KMS deployments that have more geo-distributed servers – this is beneficial for increased availability and security. However, the improvement in decryption efficiency comes with a caveat: the logarithmic overhead in HiSE's encryption only lets us encrypt about 3K records / sec (about 3-4.5× slowdown compared to ATSE's encryption, yet 7 – 12× faster than "parallelized DiSE", as benchmarked as a baseline in ATSE [20]) and incurs roughly 500 bytes of ciphertext expansion per record. Nevertheless, we believe HiSE can offer an acceptable tradeoff in practice.

2 RELATED WORKS

Threshold Symmetric Encryption (TSE). Previous works in TSE by Agrawal et al. and Christodorescu et al. are closely related to our work. Agrawal et al. provided the first formal treatment of a TSE technique in DiSE [7] using a *distributed pseudorandom function* based construction. For each encryption and decryption, the scheme requires user interaction. In ATSE [20], the *flexible key derivation protocol* allows for encryption of a group of messages with just a single user interaction. In short, the user is required to commit to a group of messages and interactively derive a partial/whole key for the group. Individual encryption of each of the messages in the group can then be carried out locally. The threshold key derivation process uses a *constrained PRF evaluation* borrowing ideas from the works of Boneh and Waters [14] and Naor-Pinkas-Reingold's *DPRF* construction [34]. However, the decryption process in ATSE still requires the user to interact for each individual ciphertext. In this paper, we propose a scheme which reduces the number of interactions during decryption to logarithmic (in the number of ciphertexts) on average (and constant in the best case).

Recently Duc et al. (DiAE [24]) extended DiSE [7] using *encryptment* [23] instead of commit-then-encrypt technique. This facilitates computation of ciphertexts *before* interaction, and therefore requires lesser online memory. Using similar techniques to optimize on-line memory requirement for HiSE (and ATSE) is an interesting future direction.

HiBE, ABE etc. Our construction borrows idea from identity-based encryption (IBE) and its variants, like attribute-based encryption (ABE). In particular, our construction uses an encryption for binary-tree access structure. The primary idea comes from binary-tree encryption [18]. However, our construction is more similar to the recent work iTire [10], which gets rid of "key-delegation" property as well, though for a different purpose. Similar constructions are used for different HIBE [9, 12, 13, 26] and ABE schemes [11, 27, 35] in the literature. In particular, the work [9] constructed a multi-receiver IBE where ciphertexts are tagged to different parties, whereas our scheme uses a binary-tree node as a tag. Combining the Boneh et al. [12] HIBE and Baek et al.'s IBE [9], Chang et al [19] proposed a hierarchical designated decryption scheme that prevents ancestors from having access to all the messages intended for their descendants. Encryption can be performed under different *choices* so that decryption is allowed by a set of permitted ancestors.

However, all of these constructions constructed public-key primitives and therefore do not offer important features such as authenticity. From another perspective, our scheme can be thought of as a (threshold) symmetric version of ABE scheme for a specific binary-tree access structure.

For more related work on threshold cryptography [6, 17, 17, 22, 28, 33] and multi-party computation [29] we refer to ATSE [20] and DiSE [7].

3 TECHNICAL OVERVIEW

HiSE leverages the inherent hierarchy or partial ordering in the access pattern structure to perform sub-linear number of interactions. That is, we decrypt an entire range of records with at most a logarithmic number of server interactions (in the number of records), each interaction requiring constant work and bandwidth. In this paper, we do not attempt to optimize for random access patterns or point queries. Let us first elaborate on the motivation.

3.1 A motivating use case

We start from a similar use case as the one in ATSE [20], but we focus on the largely unexplored part of the data lifecycle: decryption. Our performance goals (and parameter settings, later in our experimental evaluation) are influenced by our observations in the data analytics pipelines of a payments processor, such as Visa [4]; that said, the following description applies generally to any modern enterprise serving a consumer-facing application (e.g., ads, payments, social network, etc.).

Our enterprise collects events at the source containing sensitive user data – these can be payment transactions or ads conversions, as examples. These event records will be later processed via a diverse set of data processing (ETL) pipelines, but first, the source application must store them with data-at-rest protections, by encrypting them with the assistance of a KMS.

We find the following architecture to be quite typical. At the source of data ingress, a designated application is responsible for one task: encrypting records as they arrive and forwarding it to a storage service. As it accepts all user data in the clear, we will refer to this application as a *privileged* client or encryptor – it is developed with simplicity and hardened with common security measures. The encrypted data is later accessed by different analytics workloads. In practice, as a large variety of teams within the enterprise are responsible for developing downstream analytics, security, regulatory, and compliance concerns typically require us to limit the amount of data they can access and also maintain an audit trail describing which records each query has accessed. Both access control and audit trail are implemented at the KMS layer.

Scheme	Operations	Encryption (client)	Encryption (server)	Decryption (client)	Decryption (server)
DiSE [7]	H	mt	$2m$	$2m$	$2m$
	$\mathbb{G}$	$6mt$ exp, $4mt$ add	$4m$ exp, m add	$6mt$ exp, $4mt$ add	$4m$ exp, m add
	P	0	0	0	0
ATSE [20]	H	$3m + t$	3	$3m$	$3m$
	$\mathbb{G}$	$6t$ exp, $4t$ add	4 $\mathbb{G}_1$ exp, 1 $\mathbb{G}_1$ add	$6mt$ $\mathbb{G}_T$ exp, $4mt$ $\mathbb{G}_T$ add	$4m$ $\mathbb{G}_T$ exp, m $\mathbb{G}_T$ add
	P	m	0	0	m
HiSE	H	$m \log m + 2m + t$	2	$3m + t$	3
	$\mathbb{G}$	$m \log m + 6t$ $\mathbb{G}_1$ exp, $4t$ $\mathbb{G}_1$ add	4 $\mathbb{G}_1$ exp, 1 $\mathbb{G}_1$ add	$10t$ $\mathbb{G}_1$ exp, $7t$ $\mathbb{G}_1$ add, m $\mathbb{G}_T$ add	6 $\mathbb{G}_1$ exp, 3 $\mathbb{G}_1$ add
	P	m	0	$2m$	0

Table 1: Computation for encrypting or decrypting m messages in a threshold t setting (with the maliciously-secure construction): H denotes hash operations (either hash to group or bitstring e.g. SHA256); $\mathbb{G}$ denotes group operations, and since ATSE and HiSE use pairing-based groups, we denote the specific group operations by $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$; P denote pairing operations.

Recall that the encryptor produces a continuous stream of records. A typical analytics query will request for access to some *subsequence* of this data; as an example, a metrics application will periodically compute aggregate statistics on, say, one minute interval of records. Such a query is implemented by issuing a decryption query to the KMS, which replies with sufficient key material to decrypt that subsequence, but no other record beyond that. Our goal in this paper is to make this interaction with the KMS – both bandwidth and compute – sublinear in the number and size of the records.

Observe that, by arranging data in a binary tree, any subsequence can be described by at most logarithmic number of descriptors, with each descriptor denoting a sub-tree. In the remainder of this paper, we consider the problem of efficient decryption (in constant time) of a sub-tree of records, with the idea that an algorithm for this problem then lets us efficiently decrypt subsequences with logarithmic complexity.

While having efficient decryption, we would like to preserve the property of efficient encryption from ATSE. That is, we want the encryptors to encrypt a large chunk of messages efficiently and privately with fine-grained decryption, but we also want authenticity: the encryptor must only produce legitimate ciphertexts, where the temporary key derived by encryptor through interaction is bound to the set of messages (the key servers hold a long-term key in a threshold fashion which is never reconstructed explicitly).

3.2 Our Construction: An Overview

DiSE Framework. In the overview, for ease of exposition we consider a toy example, in that only four messages $\boldsymbol{m} = (m_1, m_2, m_3, m_4)$ are considered. Let us start by recalling the basic framework of (a simplifed variant of) DDH-based DiSE [7] construction that uses a (DDH-based) DPRF by Naor, Pinkas and Reingold [34]. Both encryption and decryption use the same interaction pattern to derive a message-specific key (to be used for masking) k_i for each message m_i, such that $k_i = \mathcal{H}(\gamma_i)^{sk}$ where γ_i is the commitment to message m_i (and $\mathcal{H}$ is the hash function, modeled as random oracle). Then k_i is used to "mask" m_i. Clearly, here both encryption and decryption require to derive the k_i by interacting with the key servers who holds the DPRF key sk (possibly in a t out of n threshold structure). Message privacy is guaranteed by the pseudorandomness of k_i plus hiding of the commitment scheme, whereas authenticity (which guarantees that an encryption is only possible through a legitimate interaction with the key servers, and not otherwise) follows from the binding property of the commitment and the fact that the key k_i is bound to the message m_i as well via the DPRF (which works effectively as a distributed message authentication code).

ATSE: Asymmetry in Encryption and Decryption. In ATSE [20], instead of a DPRF, another primitive, called a flexible threshold key-derivation function (FTKD) is used – this enables the masking key k_i to be derived through a bilinear pairing as $k_i = e(\mathcal{H}_1(\delta), \mathcal{H}_2(\gamma_i))^{sk}$, where the commitment to m_i now consists of two parts δ and γ_i, among them δ is common among all messages in $\boldsymbol{m}$ – this is implemented by computing a Merkle-tree on $\boldsymbol{m}$ where δ is the root hash, and γ_i is the hash values corresponding to the unique path from root to the i-th leaf (each message m_i corresponds to leaf-i). Now, exploiting the bilinear property we note that the same masking key can be derived in multiple ways; in particular we are interested in two different ways: (i) during bulk encryption a common partial key $\mathcal{H}_1(\delta)^{sk}$ is derived interactively, and then each k_i is locally computed as $k_i = e(\mathcal{H}_1(\delta)^{sk}, \mathcal{H}_2(\gamma_i))$; (ii) during decryption k_i is derived directly as $e(\mathcal{H}_1(\delta), \mathcal{H}_2(\gamma_i))^{sk}$. The key point is: during decryption, deriving masking key k_i for message m_i does not allow derivation of another masking key k_j for another message m_j. So, in contrast to DiSE, ATSE provides support for a more fine-grained encryption and decryption. However, the decryption *always* has to happen individually. For example, if one needs to decrypt both m_1 and m_2, it must interact twice to derive both k_1 and k_2 – this scales up quickly when one needs to decrypt a large sub-sequence.

A näive extension of ATSE. One may think about a potential näive extension of ATSE to incorporate hierarchy in the decryption: for example, by allowing one to also derive partial keys $\mathcal{H}_1(\delta)^{sk}$ during decryption. This, however, allows one to decrypt *all* four messages that are committed to Merkle-root δ. This idea manifests that while it is possible to enable a hierarchy in the decryption using ATSE, the hierarchy does not go beyond a single level and therefore does not provide any additional utility beyond "all-or-nothing". In particular, it is not clear how to extend the core idea from ATSE to support multiple levels of decryption efficiently.

Our scheme: HiSE. Our main idea is to construct a binary-tree such that decryption can be done at *any* node. For our toy example, we consider a binary tree of depth $\log(4) = 2$ for 4 messages. In particular, we construct a Merkle-tree where each label at leaf-i is a commitment to message m_i using hash $\mathcal{H}^{\mathsf{mt}}$; for example, X_{01} is a

commitment of m_2, and so on. Each node of the tree is indexed by a binary-string ω that encodes the path from root (indexed by empty string ε) to that node, plus each node-ω is labeled with a hash value X_ω of the binary-tree. This is depicted in Figure 1 below.

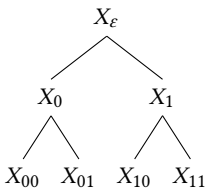

Figure 1: Each node ω is labeled with Merkle-hash X_ω. The leaves are commitments to messages: X_{00} is commitment to message m_1 and so on.

Now, our encryption follows an idea similar to BTE [18] and its variant iTire [10]. However, since both of them are public-key schemes, we made a number of crucial changes to ensure that our encryption stays symmetric *and* satisfies ciphertext authenticity. The first major change is: we now use two field elements as the long-term secret key $sk = (\alpha, \beta)$ – they are secret shared together as t out of n. Also $pp = g_1^\beta$ is made public.

Now, for each message m_i, the message-specific key k_i (or *masking key*) is computed, first by interacting with the (threshold) key-servers to derive $z = \mathcal{H}(X_\varepsilon)^\alpha$, and then locally computing $k_i = e(g_1^{r_i}, z)$ (where g_1 is generator of $\mathbb{G}_1$ in pairing $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$) which is unique to the m_i, as the randomness r_i is sampled uniquely for each message m_i. The ciphertext c_i contains log-many group elements:

$$(R_i = g_1^{r_i}, S_{\omega|_1}, S_{\omega|_2}, \ldots, S_\omega, E_i)$$

where E_i is the masked plaintext (m_i masked using k_i), and each $S_{\omega|_j}$ is computed as $\mathcal{H}(X_{\omega|_j})^{r_i}$.[7] Here ω is the binary expression of $i-1$ and the notation $\omega|_j$ refers to the first j bits of the string ω. For example, in our toy example, for m_1, m_2, m_3, m_4 the ciphertexts c_1, c_2, c_3, c_4 would look like:

$$(R_1 = g_1^{r_1}, S_{1,0} = \mathcal{H}(X_0)^{r_1}, S_{1,00} = \mathcal{H}(X_{00})^{r_1}, E_1)$$
$$(R_2 = g_1^{r_2}, S_{2,0} = \mathcal{H}(X_0)^{r_2}, S_{2,01} = \mathcal{H}(X_{01})^{r_2}, E_2)$$
$$(R_3 = g_1^{r_3}, S_{3,1} = \mathcal{H}(X_1)^{r_3}, S_{3,10} = \mathcal{H}(X_{10})^{r_3}, E_3)$$
$$(R_4 = g_1^{r_4}, S_{4,1} = \mathcal{H}(X_1)^{r_4}, S_{4,11} = \mathcal{H}(X_{11})^{r_4}, E_4)$$

Now, let us see how decryption works. Suppose that we want to decrypt m_1 and m_2, both of which are *exactly* covered under the sub-tree rooted at node-0 with label X_0 – so it is possible to decrypt them with a single interaction. In particular, the decryption interaction derives $\tilde{z} = \mathcal{H}(X_\varepsilon)^\alpha \mathcal{H}(X_0)^\beta$ in a threshold manner. Then it *locally* computes the masking keys for c_1, c_2 respectively as:

$$k_1 = e(R_1, \tilde{z}) \cdot e(pp, S_{1,0})^{-1} \text{ and } k_2 = e(R_2, \tilde{z}) \cdot e(pp, S_{2,0})^{-1}$$

A little calculation shows that these computations basically cancels out the components dependent on β, and leaves with the keys $k_1 = e(R_1, \mathcal{H}(X_\varepsilon)^\alpha)$ and $k_2 = e(R_2, \mathcal{H}(X_\varepsilon)^\alpha)$. Also, we note that, if we want to decrypt only c_1, but not c_2, that would also be possible by deriving $\tilde{z}' = \mathcal{H}(X_\varepsilon)^\alpha \mathcal{H}(X_{00})^\beta$ and then computing *only*

$$k_1 = e(R_1, \tilde{z}') \cdot e(pp, S_{1,00})^{-1}$$

Thereby a multi-leveled hierarchical decryption is possible. Now, let us argue about the security of the scheme next.

Privacy of HiSE. First note that, the value $\mathcal{H}(X_\varepsilon)^\alpha$ is just like a DPRF computation on the root commitment – this is very similar to ATSE's masking key and hence provides privacy. However, to argue that the masking key is indeed private we need to argue that from the decryption with respect to a specific node-ω, it should not be possible compute the masking key for any other ciphertexts that are not covered by the sub-tree rooted at that node-ω. Elaborating more through our example, the above decryption at node-0 (labeled X_0) should not allow computation of masking keys for c_3 or c_4. Intuitively, this can be seen from the fact that: to compute the masking key it seems essential to compute $\mathcal{H}(X_\varepsilon)^\alpha$; and since the value $\mathcal{H}(X_\varepsilon)^\alpha$ is multiplied with $\mathcal{H}(X_0)^\beta$ in $\tilde{z}$, [8] it should not be possible to "extract" $\mathcal{H}(X_\varepsilon)^\alpha$ from that – in fact, the decryption never has $\mathcal{H}(X_\varepsilon)^\alpha$ in the clear, instead it uses the S-values from the ciphertext to directly compute the message specific masking keys $e(R_i, \mathcal{H}(X_\varepsilon))^\alpha$. So, for c_3 or c_4, the S-values, included in the ciphertexts, are outside the path from root to that node: for example, c_3 has only $\mathcal{H}(X_1)^{r_3}$ and $\mathcal{H}(X_{10})^{r_3}$, but *not*, e.g. $\mathcal{H}(X_0)^{r_3}$ – this ensures that the decryption is not possible for c_3. In fact, trying to use $pp = g_1^\beta$ with the available S values in c_3, c_4 yields, for example:

$$e(pp, S_{3,1}) = e(R_3, \mathcal{H}(X_1)^\beta)$$

and attempting to divide $\tilde{z}$ with this leads to

$$e\left(R_3, \mathcal{H}(X_\varepsilon)^\alpha \cdot \left(\frac{\mathcal{H}(X_0)}{\mathcal{H}(X_1)}\right)^\beta\right)$$

which is not independent of (an unknown) β. Formalizing this, however, turns out to be trickier, and require using our new assumption, ℓ-masked BDDH along with XDH (see Section 4.1).

Authenticity of HiSE. The broad authenticity argument follows from ATSE. However, there are some crucial differences. First note that, in contrast to ATSE, DiSE our encryption is randomized. This would let the encryptor use the derived masking key multiple times to generate many ciphertexts for the same message. So, though we would obtain a property similar to plaintext integrity, we still would not have a stronger property akin to ciphertext integrity, in that the encryptor can produce only a fixed number (determined by the size of the Merkle-tree) of ciphertexts via one interaction.[9]

[7] We defer the exact description of how exactly E_i is computed until later in this section.

[8] This also shows why we need two components α and β. Giving g_1^α in pp like iTire/BTE would immediately break the privacy. Instead, we give out g_1^β and use β-dependent elements to "mask" $\mathcal{H}(X_\varepsilon)^\alpha$.

[9] Ciphertext integrity prevents a corrupt encryptor to produce *any* legitimate ciphertext locally that are unaccounted for – honest servers know exactly how many legitimate ciphertexts are being produced by one interaction. With only plaintext integrity one could locally produce *arbitrary many* legitimate ciphertexts encrypting the same messages without further interaction. In the enterprise KMS application ciphertext integrity helps in reducing the possibility of data duplication. See Remark 7.14 of ATSE [21] for more discussion.

To remedy this, we also commit to $R_i = g_1^{r_i}$ for each message m_i.[10] So for m_i, the masked value would look like

$$E_i = \mathcal{H}'(k_i) \oplus (m_i, \rho_i, \mathcal{H}''(R_i)).$$

Where $\mathcal{H}'$ maps to bit-string. During decryption one obtains (m_i, ρ_i, h_i) after demasking and then checks whether $X_\omega = \mathcal{H}^{\mathrm{mt}}(m_i, \rho_i, h_i)$ and $\mathcal{H}''(R_i) = h_i$, where ω is a binary expression of $(i-1)$.

Another issue, similar to ATSE, is that the root of the Merkle-tree X_ε does not contain information about the number N of messages committed to it. This can be exploited by a malicious encryptor by pretending to encrypt $N' < N$, such that the server would undercount the number of valid ciphertext that can be produced – in our definition (cf. Definition 5.5) this would constitute a valid forgery. To prevent this we instead compute z as $\mathcal{H}(N, X_\varepsilon)^\alpha$. So, such behavior would be caught during the decryption, in that an honest decryptor, who has now access to the Merkle-tree uses a correct N, and hence results in a different $z' \neq z$.

We do not discuss a few other issues here, which are common to all TSE schemes (DiSE, ATSE) in the literature, for example, need for a authenticated channel and inclusion of identity (j) of the encryptor in computing the masking key, which changes the scheme as $k = \mathcal{H}(j, N, X_\varepsilon)$. To illustrate we provide a simple flow for our toy HiSE scheme in Figure 2.

4 NOTATION AND PRELIMINARIES

We use $\mathbb{N}$ to denote the set of positive integers, and $[n]$ to denote the set $\{1, 2, \ldots, n\}$ (for $n \in \mathbb{N}$). We denote the security parameter by κ. We assume that, every algorithm takes κ as an implicit input and all definitions work for any sufficiently large choice of $\kappa \in \mathbb{N}$. We will omit mentioning the security parameter explicitly except a few places. Throughout the paper we use the symbol $\perp$ to denote invalidity; in particular, if any algorithm returns $\perp$ that means the algorithm failed or detected an error in the process.

We use negl to denote a negligible function; a function $f : \mathbb{N} \to \mathbb{N}$ is considered negligible is for every polynomial p, it holds that $f(n) < 1/p(n)$ for all large enough values of n. We use $\mathsf{D}(x) =: y$ or $y := \mathsf{D}(x)$ to denote the evaluation of a deterministic algorithm D on input x to produce output y. Often we use $x := \mathsf{var}$ to denote the assignment of a value var to the variable x. We write $\mathsf{R}(x) \to y$ or $y \leftarrow \mathsf{R}(x)$ to denote evaluation of a randomized algorithm R on input x to produce output y. R can be determinized as $\mathsf{R}(x; r) =: y$, where r is the explicit randomness used by R.

We model computationally bounded adversaries by probabilistic polynomial time (PPT) algorithms. Sometimes we say a particular problem is *computationally hard* to imply that for any PPT adversary, the probability of solving a random instance of that problem is bounded by $\mathrm{negl}(\kappa)$.

We denote a sequence of values or a tuple $(x_1, x_2, \ldots)$ by a standard vector notation $\boldsymbol{x}$, and its i-th element is denoted by $\boldsymbol{x}[i]$ or x_i. $|\boldsymbol{x}|$ denotes the number of elements in the vector x. For a tuple $\boldsymbol{x} = (x_1, \ldots, x_\ell)$, we write $\boldsymbol{x}^* = (x_1^*, \ldots, x_\ell^*)$ to denote another tuple for which $x_i^* \in \{x_i, \perp\}$ for $i \in [\ell]$. A list can be thought of as an ordered set; the i-th element of a list L is denoted by $L[i]$. Lists and vectors can be used interchangeably. Concatenation of two strings a and b is denoted by (a, b), or (a, b).

We denote $\omega \in \{0,1\}^{\leq \ell}$ to denote that ω is a bitstring with maximum length ℓ. For any integer $k \in \mathbb{N}$ we define its ℓ-digit binary expression as $\mathsf{Bin}(k, \ell) \in \{0,1\}^\ell$. For a binary string $\omega \in \{0,1\}^\ell$ for some integer ℓ, the truncated string with the first k bits of ω is denoted by $\omega|_k$. Also, for ω, the corresponding bit-string with most significant $\ell - 1$ bits are the same as ω and the last bit flipped is denoted by $\mathsf{LBF}(\omega)$.

We write $[j : x]$ to denote that the value x is private to party j. This is natirally extended to a set $[S : x]$ which means all parties $i \in S$ has x. For a protocol π, we write $[j : z'] \leftarrow \pi([i : (x, y)], [j : z], c)$ to denote that party i has two private inputs x and y; party j has one private input z; all the other parties have no private input; c is a common public input; and, after the execution, only j receives a private output z'. We write $[i : x_i]_{\forall i \in S}$ or more compactly $[\![\boldsymbol{x}]\!]_S$ to denote that each party $i \in S$ has a private value x_i. For a description on our communication model and protocol structure see Appendix A. For notations of security games and oracles we refer to Appendix B. For definition of Shamir's secret sharing see Appendix C.

4.1 Bilinear Pairing and our Assumptions

Our construction uses an asymmetric bilinear pairing. We consider three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ all are of prime order q. A bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable map which is *bilinear* and *non-degenerate*. We prove the security of our scheme under XDH and a new assumption, which we call *ℓ-masked BDDH*.

- *External Diffie-Hellman (XDH)*: Given uniform random generators $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and values g_2^a, g_2^b for uniform random $a, b \in \mathbb{Z}_q$, it is computationally hard to distinguish between g_2^{ab} and a uniform random $h \in \mathbb{G}_2$.
- *ℓ-masked BDDH*. Given uniform random generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2, g_T \in \mathbb{G}_T$ and elements $g_1^a, g_1^b, \{g_2^{d_i}, g_2^{c+bd_i}\}_{i \in [\ell]}$ for uniform random $a, b, c, d_i \leftarrow_\$ \mathbb{Z}_q$ it is computationally hard to distinguish g_T^{ac} from a uniform random element in $\mathbb{G}_T$. While ℓ is a parameter which is always bounded by a polynomial in κ. Note that, breaking this is easier than breaking BCDH, because one can always compute $e(g_1^a, g_2^{c+bd_i}) = g_T^{ac+abd_i}$, and breaking BCDH, compute $g_T^{abd_i}$. However, the other direction is not clear. Nevertheless, we show in Appendix G this assumption holds in the generic group model.

4.2 Message and Cipher trees

We consider a new data structure for accessing a tuple of plaintexts and ciphertexts with a labeled binary tree. We assume that the nodes of the binary-tree are indexed by a binary string with prefix-ordering. Assuming the root has depth 0, and the leaves have depth d, any node at depth $d' \in \{0, \ldots, d\}$ is indexed by a binary string $\omega \in \{0,1\}^{d'}$. For each node with index ω, its left-child is indexed by $\omega, 0$ and right child is indexed by $\omega, 1$, and this is done recursively starting from the root which is indexed by empty-string ε. Furthermore, each node is additionally labeled by a κ-bit string

[10] One may wonder we do not commit to r_i instead of $g_1^{r_i}$. We remark that it using r_i in the clear would cause a technical issue in the privacy proof, in that the reduction to ℓ-masked BDDH needs to implicitly set an unknown secret to r_i. For more details we refer to Appendix E.2

Group Encryption of (m_1, m_2, m_3, m_4) by encryptor Client-1

- Sample random values $\rho_1, \rho_2, \rho_3, \rho_4 \leftarrow_\$ \mathbb{Z}_q$ and $r_1, r_2, r_3, r_4 \leftarrow_\$ \mathbb{Z}_q$. Compute Merkle-tree $\boldsymbol{X}$ using hash $\mathcal{H}^{\text{mt}}$ on the set $((m_1, \rho_1, \mathcal{H}''(g_1^{r_1})), \ldots, (m_4, \rho_4, \mathcal{H}''(g_1^{r_4})))$ and compute the commitment X_ε to MT. (cf. Fig. 1).
- Send $(4, X_\varepsilon)$ to servers $\{2, 3, 4\}$. Each server returns $z_i := \mathcal{H}(1, 4, X_\varepsilon)^{\alpha_i}$.
- Compute $z := \prod_i z_i^{\lambda_i}$ by Lagrange using partial values.
- For $(m_1, \ldots, m_4)$, locally compute the ciphertexts as:
  $c_1 := (g_1^{r_1}, \mathcal{H}(X_0)^{r_1}, \mathcal{H}(X_{00})^{r_1}, (m_1, \rho_1, \mathcal{H}''(g_1^{r_1})) \oplus \mathcal{H}'(e(g_1^{r_1}, z)))$,
  $c_2 := (g_1^{r_2}, \mathcal{H}(X_0)^{r_2}, \mathcal{H}(X_{01})^{r_2}, (m_2, \rho_2, \mathcal{H}''(g_1^{r_2})) \oplus \mathcal{H}'(e(g_1^{r_2}, z)))$,
  $c_3 := (g_1^{r_3}, \mathcal{H}(X_1)^{r_3}, \mathcal{H}(X_{10})^{r_3}, (m_3, \rho_3, \mathcal{H}''(g_1^{r_3})) \oplus \mathcal{H}'(e(g_1^{r_3}, z)))$,
  $c_4 := (g_1^{r_4}, \mathcal{H}(X_1)^{r_4}, \mathcal{H}(X_{11})^{r_4}, (m_4, \rho_4, \mathcal{H}''(g_1^{r_4})) \oplus \mathcal{H}'(e(g_1^{r_4}, z)))$
- store $\widehat{C} := (1, \boldsymbol{X}, \varepsilon, (00, c_1), (01, c_2), (10, c_3), (11, c_4))$.

Group Decryption of $\widehat{C}_0$ by decryptor Client-7

- Parse $\widehat{C}_0$ as $(1, \boldsymbol{X}, 0, (00, c_1), (01, c_2))$. Let N denote the no. of leaves of $\boldsymbol{X}$. Here $N = 4$.
- Send $(1, 4, X_\varepsilon)$ and the value X_0 at node-0 to servers $\{4, 5, 6\}$. Each server i computes and returns $\tilde{z}_i = \mathcal{H}(1, 4, X_\varepsilon)^{\alpha_i} \mathcal{H}(X_0)^{\beta_i}$
- Compute $\tilde{z} = \prod_i \tilde{z}_i^{\lambda_i}$ by Lagrange using partial values.
- Decrypt ciphertexts c_1 and c_2 locally as follows:
  - Parse c_1 as $(R_1, S_{1,0}, S_{1,00}, E_1)$.
  - Compute $D := e(R_1, \tilde{z}) \cdot e(g_1^{\beta}, S_{1,0})^{-1}$
  - Compute $(m_1, \rho_1, h_1) := E_1 \oplus \mathcal{H}'(D)$.
  - Verify if $X_{00} = \mathcal{H}^{\text{mt}}(m_1, \rho_1, h_1)$ and if $\mathcal{H}''(R_1) = h_1$. If both succeed, then output m_1; else output $\perp$.
  - Perform the above steps for c_2.

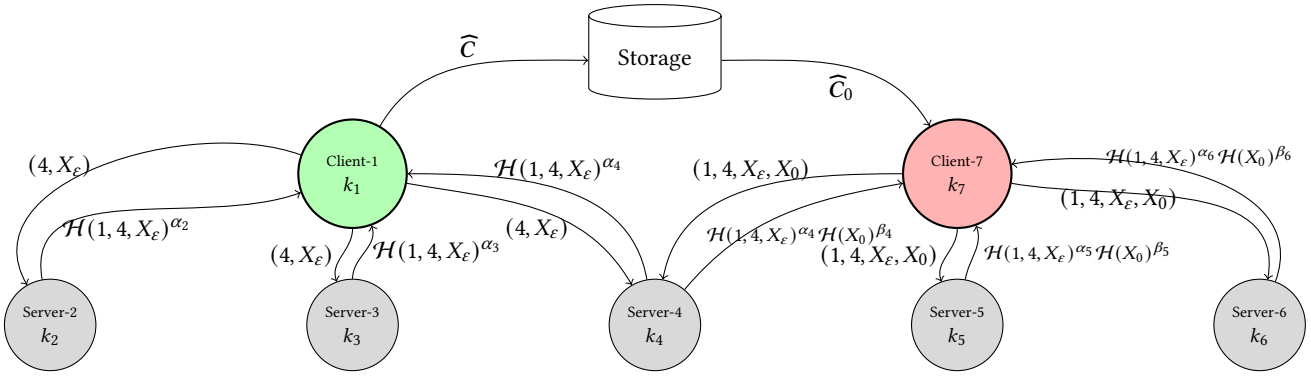

Figure 2: Flow of HiSE protocol for $n = 7$ and $t = 4$. Encryptor client (id-1) encrypts messages (m_1, m_2, m_3, m_4) in group using $t - 1$ servers $2, 3, 4$, and decryptor client (id-7) decrypts $c - 1, c_2$ in group $\widehat{C}_\omega$ for $\omega = 0$ using servers $4, 5, 6$ – both using a single interaction. Fig.1 illustrates the corresponding Merkle-tree structure.

or tags.[11] The tag at node ω is denoted by X_ω. The tree is denoted by $\boldsymbol{X} = (X_\varepsilon, X_0, X_1, X_{01}, \ldots, X_\omega, \ldots)$ which contains a sequence of tags starting from the root and is ordered according to increasing order of magnitude of ω – we simply assume ω can be computed efficiently from each tag X_ω.

Let $\boldsymbol{m} = (m_1, \ldots, m_N)$ be a tuple of messages. For simplicity we assume $N = 2^d$ for some integer d – this is without loss of generality. Then consider an associated binary tree $\boldsymbol{X}$, such that each leaf is associated with a message; therefore, m_i is associated with node $\mathsf{Bin}(i - 1)$. With respect to this binary tree we say all messages $m_1, \ldots, m_N$ are *covered* by the root of the tree. For any arbitrary node ω, let $(m'_1, \ldots m'_\ell) \subseteq (m_1, \ldots, m_N)$ be all messages that are associated with the leaves within the sub-tree rooted at ω, and these are the only such messages. Then we say the messages $(m'_1, \ldots, m'_\ell)$ are covered by node ω, and are denoted by $\boldsymbol{m}_\omega$. We can naturally extend this such that the set is covered by multiple nodes.

For any node ω, we call the tuple $(\boldsymbol{X}, \omega, ((\omega_1, m_1), \ldots, (\omega_\ell, m_\ell)))$ a message-tree, which is denoted by $\widehat{\boldsymbol{M}}_\omega$ or just $\widehat{\boldsymbol{M}}$ when the node is not specified. Note that, here we have $\ell = 2^{d-d'}$, where d' is the depth of node $\omega \in \{0, 1\}^{d'}$. For each pair (ω_i, m_i), ω_i denotes the index of the leaf-node, to which m_i corresponds to. For the same tree $\boldsymbol{X}$, consider two message-trees $\widehat{\boldsymbol{M}}_\omega$ and $\widehat{\boldsymbol{M}}_{\omega'}$ such that ω is a prefix of ω', and therefore appears in the sub-tree rooted at ω. We then say $\widehat{\boldsymbol{M}}_{\omega'}$ is a sub-message-tree of $\widehat{\boldsymbol{M}}_\omega$ and denote by the ordered relation $\widehat{\boldsymbol{M}}_{\omega'} \prec \widehat{\boldsymbol{M}}_\omega$ – this naturally extends to $\widehat{\boldsymbol{M}}_{\omega'} \preceq \widehat{\boldsymbol{M}}_\omega$ when ω' can possibly be the same as ω. If ω and ω' are such that no string is prefix of the other, then we can define a *message-forest* as $\widehat{\boldsymbol{M}}_{\omega,\omega'} := \widehat{\boldsymbol{M}}_\omega \cup \widehat{\boldsymbol{M}}_{\omega'}$.

Once encrypted, each message m_i yields a corresponding ciphertext c_i. The tuple $(\boldsymbol{X}, \omega, (c_1, \ldots, c_\ell))$ is called a cipher-tree and is denoted by $\widehat{C}_\omega$ or $\widehat{C}$. The message-tree notations described above naturally extend to this setting. For a message-tree $\widehat{\boldsymbol{M}}_\omega$, the messages that are associated with the leaves under the sub-tree rooted at node ω are compactly denoted by $\boldsymbol{m}_{\widehat{\boldsymbol{V}}_\omega}$. Similarly the ciphertext tuple under ω is compactly denoted by $\boldsymbol{c}_{\widehat{\boldsymbol{V}}_\omega}$. A message/cipher tree can be extended to include auxiliary information as, e.g. $\widehat{C} = (\text{aux}, \boldsymbol{X}, \omega, (\omega_1, c_1), \ldots)$. Looking ahead, in our constructions we use identity of an encryptor as aux within a produced cipher-tree.

For better understanding let us take a look at an example, provided in Figure 3. The entire message-tree is of depth 3, and has 8 associated messages $(m_1, \ldots, m_8)$. Within this, the message-tree

[11] Looking ahead, these tags will be the hash outputs of a Merkle-tree constructed based on the commitments of the messages.

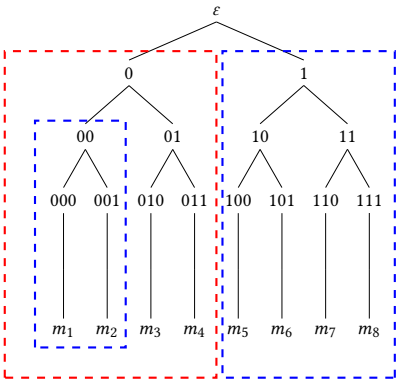

Figure 3: A plaintext message tuple $\boldsymbol{m}$ corresponds to a tree of depth 3. A few sub-trees are marked for exposition. We implicitly assume each node ω has a label X_ω.

rooted at node-01, for example, is written as $(\boldsymbol{X}, 01, (010, m_3), (011, m_4))$. Among these, the elements $(m_1, m_2, m_5, m_6, m_7, m_8)$ are covered by node-00 and node-1. So, $\widehat{\boldsymbol{M}}_{00,1}$ is a message forest (marked with dotted blue). The message tree $\widehat{\boldsymbol{M}}_{00}$ is a sub-message-tree of $\widehat{\boldsymbol{M}}_0$ (marked with dotted red), and hence $\widehat{\boldsymbol{M}}_{00} \prec \widehat{\boldsymbol{M}}_0$.

4.3 Merkle Tree Commitments

We will be using a variant of Merkle-tree commitments in our constructions. Below we directly present the construction along with syntax. We also discuss the security properties offered by Merkle-tree commitments, which we use to prove the security of our construction.

Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\kappa$ be a hash function (to be modeled as random oracles). Then for any integer $N = 2^d$ for an integer d,[12] a Merkle-tree commitment is defined as a triple of deterministic algorithms (MTCom, MTOpen, MTVer) with following descriptions.

$\mathsf{MTCom}(v_1, \ldots, v_N) \to \boldsymbol{X}$. On input N values $v_1, \ldots, v_N$ do as follows:

- **set** $d := \log(N)$
- **set** $L := N$ **and for** $i \in [N]$ **set** $x_{i,d} := v_i$.
- **for** $j \in [d]$ **do** :
  - **for** $i \in [L]$:
    - **set** $\omega := \mathsf{Bin}(i-1, d-j+1)$
    - **set** $X_\omega := x_{i,d-j+1}$.
    - **set** $x_{i,d-j} := \mathcal{H}(x_{2i-1,d-j+1}, x_{2i,d-j+1})$.
    - **set** $L := L/2$.
- **return** $\boldsymbol{X} := (X_\varepsilon, \ldots, X_{11\ldots1})$.

$\mathsf{MTOpen}(\boldsymbol{X}, \omega) \to (\overline{\boldsymbol{X}}_\omega, \boldsymbol{X}_\omega)$. On input a Merkle-tree $\boldsymbol{X}$ and any node ω do as follows:

- Let $\boldsymbol{X}_\omega$ denotes the sub-tree rooted at ω.
- Define $\overline{\boldsymbol{X}}_\omega$ as follows:
  - **set** $\overline{\boldsymbol{X}}_\omega := \{X_\varepsilon, X_\omega\}$.
  - **for** $j \in [|\omega|]$ **do** :
    - **set** $\overline{\boldsymbol{X}}_\omega := \overline{\boldsymbol{X}}_\omega \cup \{X_{\mathsf{LBF}(\omega|_j)}\}$
  - **return** $(\overline{\boldsymbol{X}}_\omega, \boldsymbol{X}_\omega)$.

$\mathsf{MTVer}(\omega, \overline{\boldsymbol{X}}_\omega, \boldsymbol{X}_\omega) \to 1/0$.

- Check the consistency of the sub-tree $\boldsymbol{X}_\omega$ with respect to the node X_ω as root.
- Check the consistency of root X_ε of the full-tree using X_ω and the siblings contained within $\overline{\boldsymbol{X}}_\omega$.
- If both checks pass, output 1, else output 0.

Correctness and Binding. The *correctness* property of MT commitments can be seen straightforwardly – correctness requires that if the commitment is done correctly then an opening would verify correctly; we do not formalize this. The *binding* holds when for any security parameter $\kappa \in \mathbb{N}$ and any tuple $\boldsymbol{v} := (v_1 \ldots, v_N) \in \{0,1\}^*$, any PPT adversary $\mathcal{A}$ can win the following security game with at most $\mathsf{negl}(\kappa)$ probability.

- Define a hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\kappa$ to be modeled as random oracles.
- **run** $(\omega, \overline{\boldsymbol{X}}_\omega, \boldsymbol{X}_\omega) \leftarrow \mathcal{A}^{\mathcal{H}}(\boldsymbol{v})$.
- **set** $\boldsymbol{X} := \mathsf{MTCom}(\boldsymbol{v})$.
- **run** $(\overline{\boldsymbol{X}}'_\omega, \boldsymbol{X}'_\omega) := \mathsf{MTOpen}(\boldsymbol{X}, \omega)$.
- **if** $(\mathsf{MTVer}(\omega, \overline{\boldsymbol{X}}_\omega, \boldsymbol{X}_\omega) = 1)$ **and** $(\mathsf{MTVer}(\omega, \overline{\boldsymbol{X}}'_\omega, \boldsymbol{X}'_\omega) = 1))$ **then return** 1; **else return** 0.

5 OUR DEFINITION: HIERARCHICAL THRESHOLD SYMMETRIC ENCRYPTION (HiSE)

Definition 5.1 (HiSE). A HiSE scheme consists of a tuple of algorithms/protocols (Setup, DistGrEnc, DistGrDec) with the following description.

- $\mathsf{Setup}(1^\kappa, 1^n, 1^t) \to (pp, [\![\boldsymbol{sk}]\!]_{[n]})$. This is a non-interactive algorithm[13] which takes as input the security parameter κ, the total number of parties n and a threshold value $t \le n$. It generates the public parameters pp and shares of secret key $[\![\boldsymbol{sk}]\!]_{[n]}$.
- $\mathsf{DistGrEnc}(pp, [\![\boldsymbol{sk}]\!]_{[n]}, [j : \boldsymbol{m}, S]) \to ([j : \widehat{\boldsymbol{C}}/\perp], [S : N, X_\varepsilon])$. This is an interactive protocol, in that the party j has a tuple $\boldsymbol{m}$ of messages and (identities of) a set of parties $S \subseteq [n]$ as input and every other party i participates with her key share sk_i. At the end of the protocol party j receives a cipher-tree $\widehat{\boldsymbol{C}}$ (or $\perp$ denoting failure) as the output, and every party in set S receives the root X_ε of a message-tree and size N of $\boldsymbol{m}$.
- $\mathsf{DistGrDec}(pp, [\![\boldsymbol{sk}]\!]_{[n]}, [j : \widehat{\boldsymbol{C}}, S]) \to ([j : \boldsymbol{m}/\perp], [S : N, X_\varepsilon, X_\omega])$. This is an interactive protocol, in that the party j has a cipher-tree $\widehat{\boldsymbol{C}}$ and (identities of) a set of parties $S \subseteq [n]$ as input and every other party i participates with her key share sk_i. At the end of the protocol party j receives a tuple of decrypted messages $\boldsymbol{m}$ (or $\perp$ denoting failure) as the output and every party in the set S receives the root X_ε and the size of messages N in the cipher tree $\widehat{\boldsymbol{C}}$.

They are required to satisfy the following consistency guarantee.

Consistency: For any $\kappa, n, t, N \in \mathbb{N}$ such that $t \le n$, all $([\![\boldsymbol{sk}]\!]_{[n]}, pp)$ output by $\mathsf{Setup}(1^\kappa, 1^n, 1^t)$, for any sequence of messages $\boldsymbol{m} = m_1 \ldots, m_N$, any $\omega \in \{0,1\}^{\le \lceil \log N \rceil}$, two sets $S, S' \subset [n]$ such that $|S|, |S'| \ge t$, and any two parties $j \in S, j' \in S'$,

[12] For simplicity we assume N to be power of 2. If not, we can simply use padding to ensure this holds. So, this is without loss of generality.

[13] However, in a full decentralized setting we can deploy a distributed key-generation (DKG) protocol to realize it interactively,.

if all the parties behave honestly, then there exists a negligible function negl for which the following probability is at least $1 - \mathsf{negl}(\kappa)$.

$$\Pr\Big[([j : \boldsymbol{m}_\omega/\perp], [S : N, X_\varepsilon, X_\omega]) \leftarrow \mathsf{DistGrDec}(\llbracket \boldsymbol{sk} \rrbracket_{[n]}, [j' : \widehat{C}_\omega, S']) \mid ([j : \widehat{C}/\perp], [S : N, X_\varepsilon]) \leftarrow \mathsf{DistGrEnc}(\llbracket \boldsymbol{sk} \rrbracket_{[n]}, [j : \boldsymbol{m}, S])\Big]$$

where the probability is over the random coin tosses of the parties involved in DistGrEnc and DistGrDec.

Let us now define the security of a HiSE scheme. Similar to prior works [7, 20] the overall security requirement is captured by three distinct properties: *correctness*, *message-privacy* and *authenticity*, where correctness and authenticity supports stronger versions. We defer the stronger definitions to Appendix D.1.

Definition 5.2 (Security of HiSE). A *H*iSE scheme is said to be secure if it satisfies *correctness (Def. 5.3), message-privacy (Def 5.4) and authenticity (Def 5.5).*

Correctness. Intuitively, a HiSE scheme is correct whenever a legitimately produced ciphertext (by executing a DistGrEnc protocol, possibly in presence of malicious parties), when decrypted (again, potentially in presence of malicious parties) yield either the actual message, or $\perp$ – in particular, the malicious parties can not make it successfully decrypt to something other than the actual message without getting detected.

Definition 5.3 (Correctness). A HiSE scheme is correct, if for any $\kappa, n, t \in \mathbb{N}$ and any PPT adversary $\mathcal{A}$, there exists a negligible function negl such that the game HiSE-Cor$_{\mathcal{A}}$, defined in Fig. 4, outputs 1 with probability at most $\mathsf{negl}(\kappa)$.

Game HiSE-Cor$_{\mathcal{A}}(1^\kappa, 1^n, 1^t)$:
- **run** $(\llbracket \boldsymbol{sk} \rrbracket_n, pp) \leftarrow \mathsf{Setup}(1^\kappa, n, t)$.
- **set** CHAL, OUT := 0.
- **run** $C \leftarrow \mathcal{A}(pp)$; **require** $C \subset [n]$ **and** $|C| < t$.
- **run** $\mathcal{A}^{O^{\text{hs-cor-en}}, O^{\text{hs-cor-dc}}, O^{\text{hs-cor-ch}}}(\{sk_i\}_{i \in C})$.
- **return** OUT.

Oracle $O^{\text{hs-cor-en}}(j, \boldsymbol{m}, S)$:
require $j \in S$.
- **run** $([j : \text{op}], \ldots) \leftarrow \mathsf{DistGrEnc}(pp, \llbracket \boldsymbol{sk} \rrbracket_S, [j : \boldsymbol{m}, S])$.
- **if** $j \notin C$ **then return** op.

Oracle $O^{\text{hs-cor-dc}}(j, \widehat{C}, S)$:
require: $j \in S$.
- **run** $([j : \text{op}], \ldots) \leftarrow \mathsf{DistGrDec}(pp, \llbracket \boldsymbol{sk} \rrbracket_S, [j : \widehat{C}, S])$.
- **if** $j \notin C$ **then return** op.

Oracle $O^{\text{hs-cor-ch}}(j, S, j', S', \boldsymbol{m} = (m_1 \ldots, m_N), \omega)$:
require $j \in S \setminus C$ **and** $j' \in S' \setminus C$ **and** $\omega \in \{0,1\}^{\lceil \log N \rceil}$ **and** CHAL = 0 **and** OUT = 0.
- **set** CHAL := 1.
- **run** $([j : \text{op}], \ldots) \leftarrow \mathsf{DistGrEnc}(pp, \llbracket \boldsymbol{sk} \rrbracket_S, [j : \boldsymbol{m}, S])$.
- **if** op = $\perp$ **then set** OUT := 0; **else set** $\widehat{C}$:= op **and do** :
  - **run** $([j' : \text{op}'], \ldots) \leftarrow \mathsf{DistGrDec}(pp, \llbracket \boldsymbol{sk} \rrbracket_S, [j' : \widehat{C}_\omega, S'])$.
  - **if** op$' = \boldsymbol{m}^*$ **then set** OUT := 0; **else set** OUT := 1.

Figure 4: The correctness game for HiSE.

Message privacy. Message privacy is naturally defined as an extension of ATSE message-privacy [20]. In particular, the adversary is given access to an "honest" decryption oracle $O^{\text{hs-mp-dc}}$, in which even the challenge cipher-tree $\widehat{C}^\star$ can be queried, but the decryption takes place at an honest party's disposal, and the result is not explicitly given to the adversary – this is similar to both DiSE [7] and ATSE [20]. Finally, just like ATSE, the adversary is provided a special decryption oracle $O^{\text{hs-mp-ch-dc}}$ which works specifically on the challenge cipher-tree, but only the part which is common to both the challenge message vectors – this captures the intuition that if the attacker gets to decrypt part of the cipher-tree on a node, the messages that are not covered by that node remains completely hidden. The definition is formally presented next.

Definition 5.4 (Message privacy). A HiSE scheme satisfies message privacy if for any integers $\kappa, n, t \in \mathbb{N}$ such that $n \geq t$ and any PPT adversary $\mathcal{A}$, there exists a negligible function negl such that:

$$\Big|\Pr\left[\text{HiSE-MsgPriv}_{\mathcal{A}}(1^\kappa, 1^n, 1^t, 0) = 1\right] - \Pr\left[\text{HiSE-MsgPriv}_{\mathcal{A}}(1^\kappa, 1^n, 1^t, 1) = 1\right]\Big| \leq \mathsf{negl}(\kappa),$$

where the security game HiSE-MsgPriv is described in Fig. 5.

Authenticity. The authenticity definition is a natural extension from DiSE and ATSE, but with an important difference. Here, it captures that an encryptor may produce exactly N valid ciphertexts by either an encryption query or a decryption query, but it has to be a "fresh" one with respect to the Merkle-tree root $X\varepsilon$ where the Merkle-tree has N leaves (and hence commits to N messages). Allowing decryption query is necessary here, because in our construction, via a decryption query for *any* node, one obtains sufficient information to encrypt all N messages that are committed through X_ε. However, any other encryption/decryption query for the same Merkle-tree with root X_ε would not give any additional information to produce more ciphertexts. This is captured by maintaining the list L_{root} which stores the roots of the Merkle-tree, and it is only counted (N times) via the counter ct when it appears for the first time in either $O^{\text{hs-au-en}}$ or $O^{\text{hs-au-dc}}$. Also, we crucially rely on the fact that in interactive protocols DistGrEnc and DistGrDec all parties in the set S receives X_ε and N (plus the node label X_ω in DistGrDec, which is not used).

Definition 5.5 (Authenticity). An HiSE scheme satisfies *authenticity* if for any integers κ, n, t, and any PPT adversary $\mathcal{A}$, the probability that the security game HiSE-Auth depicted at Fig. 6 outputs 1 is at most $\mathsf{negl}(\kappa)$.

6 OUR HISE CONSTRUCTION

We present our base HiSE construction in Figure 7. The construction is based on asymmetric bilinear pairing of Type-3 (no easy isomorphism). We show that our construction is secure (as per Definition D.1) from XDH and ℓ-masked BDDH over the bilinear group. Formally we present the following theorem, a proof of which is found in Appendix E.

Game HiSE-MsgPriv$_{\mathcal{A}}(1^\kappa, 1^n, 1^t, b)$:

- **set** CHAL $:= 0$.
- **set** $c^\star := \emptyset$.
- **set** $\mathcal{M} := \emptyset$.
- **run** $(\llbracket sk \rrbracket_n, pp) \leftarrow \mathsf{Setup}(1^\kappa, n, t)$.
- **run** $C \leftarrow \mathcal{A}(pp)$;
  require $C \subset [n]$ **and** $|C| < t$.
- **run** $b' \leftarrow \mathcal{A}^{O^{\text{hs-mp-en}}, O^{\text{hs-mp-dc}}, O^{\text{hs-mp-ch}}, O^{\text{hs-mp-ch-dc}}}(\{sk_i\}_{i \in C})$
- **return** b'.

Oracle $O^{\text{hs-mp-en}}(j, \boldsymbol{m}, S)$:
require $j \in S$.

- **run** $([j : \text{op}], \ldots) \leftarrow \mathsf{DistGrEnc}(pp, \llbracket sk \rrbracket_{[n]}, [j : \boldsymbol{m}, S])$.
- **if** $j \notin C$ **then return** op.

Oracle $O^{\text{hs-mp-dc}}(j, \widehat{C}, S)$:
require $j \in S \setminus C$.

- **run** $([j : \text{op}], \ldots) \leftarrow \mathsf{DistGrDec}(pp, \llbracket sk \rrbracket_{[n]}, [j : \widehat{C}, S])$.

Oracle $O^{\text{hs-mp-ch}}(j^\star, \boldsymbol{m}_0, \boldsymbol{m}_1, S^\star)$:
require $j^\star \in S \setminus C$ **and** $|\boldsymbol{m}_0| = |\boldsymbol{m}_1|$ **and** CHAL $= 0$.

- **set** CHAL $:= 1$.
- **for** $i \in [N]$: **if** $\boldsymbol{m}_0[i] = \boldsymbol{m}_1[i]$ **set** $\mathcal{M} := \mathcal{M} \cup \{\boldsymbol{m}_0[i]\}$.
- **run** $([j^\star : \text{op}], \ldots) \leftarrow \mathsf{DistGrEnc}(pp, \llbracket sk \rrbracket_n, [j^\star : \boldsymbol{m}_b, S^\star])$.
- **if** op $= \perp$ **return** $\perp$; **else do** :
  - **set** $\widehat{C}^\star := \text{op}$
  - **return** $\widehat{C}^\star$.

Oracle $O^{\text{hs-mp-ch-dc}}(j, \widehat{C}, S)$:
require $j \in S$ **and** CHAL $= 1$ **and** $\widehat{C} \preceq \widehat{C}^\star$ **and** $\boldsymbol{m}_{\widehat{C}} \in \mathcal{M}$.

- **run** $([j : \text{op}], \ldots) \leftarrow \mathsf{DistGrDec}(pp, \llbracket sk \rrbracket_{[n]}, [j : c, S])$.
- **if** $j \notin C$ **return** op.

Figure 5: The HiSE message-privacy game.

Theorem 6.1. *Our construction (Fig. 7) is a secure HiSE scheme assuming XDH and ℓ-masked BDDH over the underlying bilinear pairing in the programmable random oracle model.*

We defer the strong HiSE construction to Appendix D.2.

7 EXPERIMENTAL EVALUATION

In this section, we compare the strong versions[14] of HiSE with ATSE [20] on a few dimensions – latency, throughput, communication (bandwidth), and ciphertext expansion – while varying the number of servers n, threshold t, and the number of messages N in a group (to manifest the benefits of bulk encryption and decryption).

Implementation. We implement the HiSE scheme in Rust (roughly 1K LOC), using the BLS12-381 pairing-based curve implemented in [8]. For comparison, we also implement DiSE [7] and ATSE [20]. The code for all three schemes is open-sourced at https://github.com/rsinha/hise.

[14] As expalined in Remark 7.5 of ATSE [21], the stronger notion makes more sense in our application to enterprise data encryption.

Game HiSE-Auth$_{\mathcal{A}}(1^\kappa, 1^n, 1^t)$:

- **set** ct $:= 0$ **and** $L_{\text{ctxt}} := \emptyset$ **and** $\tau := 0$ **and** $L_{\text{root}} := \emptyset$.
- **set** SUCC $:= 0$ **and** CHAL $:= 0$.
- **run** $(\llbracket sk \rrbracket_n, pp) \leftarrow \mathsf{Setup}(1^\kappa, n, t)$.
- **run** $C \leftarrow \mathcal{A}(pp)$;
  require $C \subset [n]$ **and** $|C| < t$.
- **run** $\mathcal{A}^{O^{\text{hs-au-en}}, O^{\text{hs-au-dc}}, O^{\text{hs-au-t-dc}}, O^{\text{hs-au-ch}}}(\{sk_i\}_{i \in C})$.
- **return** SUCC.

Oracle $O^{\text{hs-au-en}}(j, \boldsymbol{m}, S)$:
require $j \in S$.

- **run** $([j : \widehat{C}/\perp], [S : X_\varepsilon, N]) \leftarrow \mathsf{DistGrEnc}(pp, \llbracket sk \rrbracket_{[n]}, [j : \boldsymbol{m}, S])$.
- **if** $j \notin C$ **then** $\tau := \tau + 1$ **and** $L_{\text{ctxt}}[\tau] := \{\text{op}\}$;
  else if $X_\varepsilon \notin L_{\text{root}}$ **then** :
  - **set** $L_{\text{root}} := L_{\text{root}} \cup X_\varepsilon$.
  - **set** ct $:=$ ct $+ N \cdot |S \setminus C|$.

Oracle $O^{\text{hs-au-dc}}(j, \widehat{C}_\omega, S)$:
require: $j \in S$.

- **run** $([j : \boldsymbol{m}/\perp], [S : X_\varepsilon, X_\omega, N]) \leftarrow \mathsf{DistGrDec}(pp, \llbracket sk \rrbracket_{[n]}, [j : \widehat{C}_\omega, S])$.
- **if** $j \in C$ **and** $X_\varepsilon \notin L_{\text{root}}$ **then** :
  - **set** $L_{\text{root}} := L_{\text{root}} \cup X_\varepsilon$.
  - **set** ct $:=$ ct $+ N \cdot |S \setminus C|$.

Oracle $O^{\text{hs-au-t-dc}}(j, i, \omega, S)$:
require: $j \in S \setminus C$ **and** $i \in [\tau]$

- **set** $\widehat{C} := L_{\text{ctxt}}[i]$.
- **run** $([j : \text{op}], \ldots) \leftarrow \mathsf{DistGrDec}(pp, \llbracket sk \rrbracket_{[n]}, [j : \widehat{C}_\omega, S])$.

Oracle $O^{\text{hs-au-ch}}(L_{\text{forge}})$:

- **set** $\ell := \lfloor \text{ct}/g \rfloor$, where $g := t - |C|$.
- **set** $\left((j_1, S_1, \widehat{C}_{\omega_1}), (j_2, S_2, \widehat{C}_{\omega_2}) \ldots)\right) := L_{\text{forge}}$;
  require CHAL $= 0$ **and** SUCC $= 0$ **and** $j_1, j_2, \ldots \notin C$ **and** $\forall\, i \neq i' : \widehat{C}_i \not\preceq \widehat{C}_{i'}$.
- **set** CHAL $:= 1$ **and** $L_{\text{succ}} := \emptyset$ **and set** $\gamma := |L_{\text{forge}}|$.
- **for** $\forall\, i \in [\gamma]$:
  - **run op** $\leftarrow \mathsf{DistGrDec}^\dagger(pp, \llbracket sk \rrbracket_n, [j_i : \widehat{C}_i, S_i])$.
  - $\forall\, k \in [|\textbf{op}|]$: **if** $\text{op}_k \neq \perp$ **then set** $L_{\text{succ}} := L_{\text{succ}} \cup \{\text{op}_k\}$;
- **if** $|L_{\text{succ}}| > \ell$ **then set** SUCC $:= 1$.

Figure 6: Description of the authenticity game.

Experimental Setup. While larger parameters for n and t only lead to larger improvements for HiSE, we nevertheless pick practical values of n between 2 and 24, modeling a reasonable enterprise deployment of a threshold KMS system. Experiments are run using two server-grade machines, each equipped with a 16-core Intel Xeon E5-2640 CPU @ 2.6 Ghz and 64 GB DDR4 RAM. We use the following setup for latency and throughput measurements, which, to our understanding, provides a fair comparison between the schemes. For the latency measurement, we run both the client and server(s) on the same machine to avoid including the network latency, thus computing a more accurate relative speedup. We report the end-to-end latency, which includes: 1) the server computation (all servers

Ingredients and Parameters

Public parameters:

- The security parameter κ.
- An efficiently computable Type-3 bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are multiplicative groups and each of prime order q; g_1 and g_2 are randomly chosen generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively.
- Descriptions of hash functions $\mathcal{H}_1 : \{0,1\}^* \to \{0,1\}^\kappa$; $\mathcal{H}_2 : \{0,1\}^* \to \mathbb{G}_2$; $\mathcal{H}_3 : \mathbb{G}_T \to \{0,1\}^{\mathsf{poly}(\kappa)}$; $\mathcal{H}_4 : \mathbb{G}_1 \to \{0,1\}^{\mathsf{poly}(\kappa)}$

Merkle-Tree Commitment. Consists of algorithms (MTCom, MTOpen, MTVer)

Construction

$\mathsf{Setup}(1^\kappa, 1^n, 1^t) \to (pp, [\![\boldsymbol{sk}]\!]_{[n]})$:

1. Choose $\alpha, \beta \leftarrow\!\!\$\ \mathbb{Z}_q$ and set $sk := (\alpha, \beta)$. Set $pp := g_1^\beta$.
2. Compute the secret shares $\{(\alpha_i, \beta_i)\}_{i\in[n]}$ of pair (α, β) using (q, n, t)-SSS; and set $sk_i := (\alpha_i, \beta_i)$.

$\mathsf{DistGrEnc}(pp, [\![\boldsymbol{sk}]\!]_{[n]}, [j : \boldsymbol{m}, S]) \to ([j : \widehat{C}/\perp], [S : N, X_\varepsilon])$: This is a two-round distributed protocol, that works as follows:

- **Pre-computation (local).** Let $(m_1 \ldots, m_N) := \boldsymbol{m} \in \{0,1\}^*$ and $N = 2^d$. Party-j chooses a randomnesses $\rho_k \leftarrow\!\!\$ \mathbb{Z}_q$ and $r_k \leftarrow\!\!\$ \mathbb{Z}_q$ for all $k \in [N]$ and then generates the Merkle-Tree commitment $\boldsymbol{X} := \mathsf{MTCom}(y_1, \ldots, y_N)$ where $y_k = \mathcal{H}_1(m_k, \rho_k, \mathcal{H}_4(g_1^{r_k}))$. Let X_ε be the root of $\boldsymbol{X}$.
- **Round 1.** Party-j sends (N, X_ε) to all parties in set S.
- **Round 2.** Each party $i \in S$ sends back $z_i := \mathcal{H}_2(j, N, X_\varepsilon)^{\alpha_i}$ to party-j.
- **Finalize (local).** Party-j, once receives at least $t-1$ responses z_i, computes $z := \prod_i z_i^{\lambda_i}$ for appropriately chosen λ_is. Then do as follows for all $k \in [N]$:
  - Let $\omega := \mathsf{Bin}(k-1, d)$ be the binary vector encoding (of length d bits).
  - Compute ciphertext $c_k := (g_1^{r_k}, \mathcal{H}_2(X_{\omega|_1})^{r_k}, \mathcal{H}_2(X_{\omega|_2})^{r_k}, \ldots, \mathcal{H}_2(X_\omega)^{r_k}, (m_k, \rho_k, \mathcal{H}_4(g_1^{r_k})) \oplus \mathcal{H}_3(e(g_1^{r_k}, z)))$
  - Set $\omega_k := \omega$.

  Output the cipher-tree $\widehat{C} := (j, \boldsymbol{X}, \varepsilon, (\omega_1, c_1), \ldots, (\omega_N, c_N))$.

$\mathsf{DistGrDec}(pp, [\![\boldsymbol{sk}]\!]_{[n]}, [j : \widehat{C}, S]) \to ([j : \boldsymbol{m}/\perp], [S : N, X_\varepsilon, X_\omega])$. : Party j parses $\widehat{C}$ as $(j', \boldsymbol{X}, \omega, (\omega_1, c_1), \ldots, (\omega_\ell, c_\ell))$, and let N be the number of leaves in $\boldsymbol{X}$. Then:

- **Round 1.** Party-j sends $(j', N, X_\varepsilon, X_\omega)$ to all parties $i \in S$.
- **Round 2.** Each party-i, on receiving $(j', N, X_\varepsilon, X_\omega)$, computes $\tilde{z}_i = \mathcal{H}_2(j', N, X_\varepsilon)^{\alpha_i} \mathcal{H}_2(X_\omega)^{\beta_i}$ and sends that back to party j.
- **Finalize (local).** On receiving $\tilde{z}_i$ from at least $t-1$ parties, it computes $\tilde{z} = \prod_i \tilde{z}_i^{\lambda_i}$ including its own locally computed value. Then for each $k \in [\ell]$ do as follows:
  - Let $\omega' := \omega_k$.
  - Parse c_k as $(R, S_{\omega'|_1}, S_{\omega'|_2}, \ldots, E)$.
  - Compute $D := e(R, \tilde{z}) \cdot e(pp, S_\omega)^{-1}$
  - Compute $(m_k, \rho_k, h_k) := E \oplus \mathcal{H}_3(D)$.
  - Verify if $X_{\omega'} = \mathcal{H}_1(m_k, \rho_k, h_k)$ and if $\mathcal{H}_4(R) = h_k$. If either fails set $m_k := \perp$.

  Finally verify the tree $\mathsf{MTVer}(\boldsymbol{X})$, if it fails then return $\perp$, else output $(m_1, \ldots, m_\ell)$.

Figure 7: Our HiSE construction

run in parallel) during which the client is waiting; 2) client's verification of responses from at t servers (i.e., verifying the NIZK proofs), and 3) client's Lagrange interpolation to compute the group key, and using the group key to encrypt all m messages in the group. We use a single threaded implementation for latency measurement, thus capturing the total amount of computation involved in processing the m messages. For the throughput measurement, we run all n KMS server processes on one machine, and the client process on the other machine – as both the ATSE and HiSE schemes are embarrassingly parallel[15], we let the client launch multiple concurrent threads for processing m messages in parallel, which uses all available CPU cores. Moreover, we use an asynchronous RPC call between the client and server, allowing both machines to operate at peak compute, thus measuring the number of messages that can be processed by all available CPU cores (with linear scaling).

Latency and Throughput Measurement. Tables 9 and 8 list the latency and throughput measurements. In general, the experiments support the expectations laid out when listing the algebraic operations in Table 1. A summary of our key findings are:

[15]In both ATSE and HiSE, after interacting with the servers and deriving the group key using Lagrange interpolation, the client can work on all m messages in parallel.

t	n	Latency (sec)						Throughput (messages / sec)					
		ATSE 1 msg	ATSE 100 msg	ATSE 10000 msg	HiSE 1 msg	HiSE 100 msg	HiSE 10000 msg	ATSE 1 msg	ATSE 100 msg	ATSE 10000 msg	HiSE 1 msg	HiSE 100 msg	HiSE 10000 msg
$n/3$	6	0.033	3.245	321.50	0.020	0.283	26.790	709.603	704.866	705.415	1304.25	5704.95	6010.98
	12	0.055	5.515	546.11	0.023	0.289	26.978	351.567	355.763	352.807	853.25	5683.88	6007.55
	18	0.078	7.859	774.79	0.028	0.290	26.631	236.889	235.753	234.892	639.98	5577.31	5996.60
	24	0.101	10.092	1001.64	0.034	0.298	26.649	178.014	176.713	174.847	497.66	5451.92	5989.70
$n/2$	6	0.043	4.382	435.76	0.018	0.281	26.689	447.499	469.188	453.500	1085.33	5753.81	5744.11
	12	0.078	7.824	773.38	0.028	0.292	27.134	236.734	235.214	230.184	639.88	5506.14	5688.60
	18	0.113	11.407	1111.68	0.038	0.304	27.048	154.294	153.583	154.677	440.40	5060.87	5471.84
	24	0.147	14.674	1452.99	0.049	0.330	26.882	118.861	116.818	117.119	341.14	5222.01	6002.05
$2n/3$	6	0.064	5.561	545.69	0.022	0.284	26.953	356.949	356.675	352.815	861.45	5710.81	5993.93
	12	0.101	10.088	999.44	0.043	0.326	27.485	177.952	175.480	175.321	496.72	5456.26	5984.03
	18	0.144	14.636	1454.70	0.048	0.314	27.115	118.463	117.505	117.843	350.42	5152.43	6005.81
	24	0.189	19.243	1913.32	0.061	0.330	27.019	88.556	88.604	87.696	278.14	4996.83	6013.53

Figure 8: Decryption latency and throughput metrics with 32 byte messages.

t	n	Latency (sec)						Throughput (messages / sec)					
		ATSE 1 msg	ATSE 100 msg	ATSE 10000 msg	HiSE 1 msg	HiSE 100 msg	HiSE 10000 msg	ATSE 1 msg	ATSE 100 msg	ATSE 10000 msg	HiSE 1 msg	HiSE 100 msg	HiSE 10000 msg
$n/3$	6	0.009	0.196	18.548	0.012	0.563	84.216	2120.57	8389.85	8700.05	1536.93	2895.89	1918.27
	12	0.011	0.193	18.376	0.012	0.555	84.258	1628.12	8367.38	8646.25	1518.66	2934.35	1916.98
	18	0.015	0.197	18.626	0.016	0.551	83.626	1142.36	8198.13	8658.69	1069.33	2893.78	1918.50
	24	0.019	0.200	18.443	0.020	0.566	83.726	894.92	8037.85	8603.37	866.97	2886.77	1909.47
$n/2$	6	0.009	0.195	18.496	0.010	0.555	84.018	2045.58	8462.63	8685.33	1900.68	2939.93	1914.85
	12	0.015	0.197	18.484	0.016	0.554	84.181	1160.34	8192.47	8685.98	1108.65	2915.91	1898.08
	18	0.021	0.207	18.574	0.022	0.567	84.263	807.37	7954.19	8697.87	783.65	2881.27	1912.41
	24	0.027	0.209	18.511	0.028	0.572	84.192	620.32	7700.20	8669.60	598.19	2853.89	1913.25
$2n/3$	6	0.011	0.196	18.586	0.012	0.555	84.180	1623.92	8360.25	8676.99	1448.37	2862.63	1924.12
	12	0.019	0.204	18.492	0.021	0.557	84.212	895.71	8005.88	8629.22	868.31	2886.84	1927.32
	18	0.027	0.208	18.578	0.028	0.568	84.686	620.95	7716.82	8622.98	602.49	2851.09	1927.92
	24	0.036	0.220	18.600	0.038	0.673	84.230	420.36	7406.07	8704.37	464.60	2800.98	1927.25

Figure 9: Encryption latency and throughput metrics with 32 byte messages.

- Though the server interaction is slightly more expensive in HiSE compared to ATSE, by amortizing it over m messages, we see an order of magnitude improvement in latency and throughput of the decryption procedure. Depending on the parameter settings, we observe between 10-65× improvement in throughput and 12-70× improvement in latency.
- The downside is that encryption has roughly 4.5× higher latency and lower throughput, due to the logarithmic number of additional group operations per message.
- We find an interesting tradeoff in the encryption procedure of HiSE. While larger values of m should further amortize the one-time server interaction, the $\log(m)$ overhead makes encryption more inefficient with larger m – this is not the case in decryption, which has constant number of operations for each message. We find a sweet-spot roughly around $m = 1000$.

Ciphertext Expansion. In addition to the slower encryption, another downside of HiSE is the ciphertext expansion due to the logarithmic number of group elements; that said, both ATSE and HiSE have log-size ciphertext already from the Merkle-tree opening proof (root-to-leaf path). For a message of size x bytes, the corresponding ciphertext has $x + 64 + 48\lceil log_2(m)\rceil$ bytes – we have 64 bytes from xor'd encoding of $\rho_k, \mathcal{H}_4(g_1^{r_k})$, and 48-byte $\lceil log_2(m)\rceil$ $\mathbb{G}_1$ elements. Concretely, for $m = 100$, we have $448 + x$ bytes; for m=10000, we have $784 + x$ bytes. In addition to the list of m ciphertexts, each group has a merkle tree of size $2m - 1$ hashes, where each hash has size 32 bytes (if using SHA-256).

Communication. One other advantage of HiSE is the reduction in bandwidth between the client and server(s). HiSE incurs constant communication overhead in the number of messages m; in contrast, ATSE transmits linear number of elements. For each interaction (i.e., for each invocation of DistGrEnc), ignoring the underlying TLS channel's overheads, the HiSE client receives 144^*t bytes, comprising one 48-byte group element and three 32-byte scalar values. In contrast, an ATSE client receives 480^*t^*m bytes. For $m = 10000$, that constitutes 4 orders of magnitude reduction in bandwidth.

8 CONCLUSION

Building on amortised threshold symmetric encryption (ATSE [20]), we develop new ideas for binary-tree-based hierarchical (fine-grained) decryption, while preserving the original security properties of privacy and authenticity. While our encryption suffers a 3-4.5× overhead, we observe between 10-65× improvement in latency and throughput during decryption. This improvement is beneficial in enterprise workloads, which are "read" (decryption) heavy.

REFERENCES

[1] Coinbase custody. https://www.coinbase.com/prime/custody.

[2] Dyadic Security. https://www.dyadicsec.com.

[3] Vault by HashiCorp. https://www.vaultproject.io/.

[4] Visa. https://usa.visa.com/.

[5] Shashank Agrawal, Wei Dai, Atul Luykx, Pratyay Mukherjee, and Peter Rindal. Paradise: Efficient threshold authenticated encryption in fully malicious model. In Takanori Isobe and Santanu Sarkar, editors, *Progress in Cryptology - INDOCRYPT 2022 - 23rd International Conference on Cryptology in India, Kolkata, India, December 11-14, 2022, Proceedings*, volume 13774 of *Lecture Notes in Computer Science*, pages 26–51. Springer, 2022.

[6] Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. PASTA: PASsword-based threshold authentication. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 2042–2059, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[7] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed symmetric-key encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1993–2010, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[8] arkworks contributors. `arkworks` zksnark ecosystem. https://arkworks.rs, 2022.

[9] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Universal designated verifier signature proof (or how to efficiently prove knowledge of a signature). In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 644–661, Chennai, India, December 4–8, 2005. Springer, Heidelberg, Germany.

[10] Leemon Baird, Pratyay Mukherjee, and Rohit Sinha. i-TiRE: Incremental timed-release encryption or how to use timed-release encryption on blockchains? In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 235–248, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[11] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE symposium on security and privacy (SP'07)*, pages 321–334. IEEE, 2007.

[12] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

[13] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[14] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany.

[15] Xavier Boyen. The uber-assumption family. In Steven D. Galbraith and Kenneth G. Paterson, editors, *Pairing-Based Cryptography – Pairing 2008*, pages 39–56, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[16] Luís Brandão and Rene Peralta. NIST First Call for Multi-Party Threshold Schemes. 2023. https://csrc.nist.gov/publications/detail/nistir/8214c/draft.

[17] Julian Brost, Christoph Egger, Russell W. F. Lai, Fritz Schmid, Dominique Schröder, and Markus Zoppelt. Threshold password-hardened encryption services. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 409–424, Virtual Event, USA, November 9–13, 2020. ACM Press.

[18] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.

[19] Shu-Hui Chang, Chuan-Ming Li, and Tzonelih Hwang. Identity-based hierarchical designated decryption. *J. Inf. Sci. Eng.*, 26(4):1243–1259, 2010.

[20] Mihai Christodorescu, Sivanarayana Gaddam, Pratyay Mukherjee, and Rohit Sinha. Amortized threshold symmetric-key encryption. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2758–2779, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[21] Mihai Christodorescu, Sivanarayana Gaddam, Pratyay Mukherjee, and Rohit Sinha. Amortized threshold symmetric-key encryption. Cryptology ePrint Archive, Report 2021/1176, 2021. https://eprint.iacr.org/2021/1176.

[22] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[23] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 155–186, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[24] Alexandre Duc, Robin Müller, and Damian Vizár. Diae: Re-rolling the dise. Cryptology ePrint Archive, Paper 2022/1275, 2022. https://eprint.iacr.org/2022/1275.

[25] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012: 13th International Conference in Cryptology in India*, volume 7668 of *Lecture Notes in Computer Science*, pages 60–79, Kolkata, India, December 9–12, 2012. Springer, Heidelberg, Germany.

[26] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566, Queenstown, New Zealand, December 1–5, 2002. Springer, Heidelberg, Germany.

[27] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 89–98, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. Available as Cryptology ePrint Archive Report 2006/309.

[28] Stanislaw Jarecki, Hugo Krawczyk, and Jason K. Resch. Updatable oblivious key management for storage systems. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 379–393, London, UK, November 11–15, 2019. ACM Press.

[29] Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl, Eduardo Soria-Vazquez, and Srinivas Vivek. Faster secure multi-party computation of AES and DES using lookup tables. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17: 15th International Conference on Applied Cryptography and Network Security*, volume 10355 of *Lecture Notes in Computer Science*, pages 229–249, Kanazawa, Japan, July 10–12, 2017. Springer, Heidelberg, Germany.

[30] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: exploiting speculative execution. *Commun. ACM*, 63(7):93–101, 2020.

[31] Yehuda Lindell. The UNBOUND Nextgen vHSM. https://www.fintechfutures.com/files/2020/09/vHSM-Whitepaper-v3.pdf.

[32] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg, and Raoul Strackx. Meltdown: reading kernel memory from user space. *Commun. ACM*, 63(6):46–56, 2020.

[33] Philip D. MacKenzie, Thomas Shrimpton, and Markus Jakobsson. Threshold password-authenticated key exchange. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 385–400, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany.

[34] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

[35] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 457–473, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[36] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 256–266, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

Appendix

A OUR COMMUNICATION MODEL

On our communication model and protocol structure. All our protocols are over *secure and authenticated* channel – they require two rounds of communication, in that a initiator party (often referred to as a client for this execution) sends messages to a number of other parties (referred to as the servers for this execution); each server computes on the message and then sends the responses back to the client in the second round; the client then combine the responses together to compute the final output. Importantly, the servers do not interact among themselves in an execution. However, we stress that our definitions and the protocol notations are flexible enough to accommodate protocols with different structures.

B SECURITY GAMES AND ORACLES.

While our formalization uses intuitive security games, to handle many cases, the descriptions often become cumbersome. Therefore, we use simple pseudo-code notation introduced in ATSE [20]. The exposition that follows is taken verbatim from ATSE.

Adversaries are formalized as probabilistic polynomial time (PPT) *stateful* algorithms. Adversarial queries are formalized via **interactive oracles**; they may run interactive protocols with the adversary. In particular, when a protocol, say $\pi(\cdots)$ is being executed inside an interactive oracle, the oracle computes and sends messages on behalf of the honest parties following the protocol specifications; the adversary controls the corrupted parties– such an execution may need multiple rounds of interactions between the oracle and the adversary. Occasionally, the oracle needs to execute an instance of a protocol $\pi(\cdots)$ internally, in that the codes of everyone are executed honestly by the oracle– such special executions are denoted by a *dagger* superscript, e.g. $\pi^{\dagger}(\cdots)$ and it is then treated like a non-interactive algorithm. We use standard **if – then – else** statements for branching and **for** to denote a loop. A branching within another is distinguished by indentations. The command **set** is used for updating/assigning/parsing variables, whereas **run** is used for executing an algorithm/protocol. The command **uniform** is used to qualify a variable, v (say) to denote a uniform random sample in the domain of v is drawn and assigned to v. Finally, **require** is used to impose conditions on a preceding set of variables; if the condition is satisfied, then the next step is executed, otherwise the experiment aborts at this step (for simplicity we keep the abortion implicit in the descriptions, they can be made explicit by using existing/new flags). All variables, including counters, flags and lists, that are initialized in the security game, are considered global, such that they can be accessed and modified by any oracle.

C BUILDING BLOCKS

In this section we present formal definitions of building blocks, such as Shamir's secret sharing trapdoor commitments and simulation sound non-interactive zero-knowledge proofs – they are borrowed, almost verbatim, from prior works [7, 20].

C.1 Shamir's Secret Sharing

We use Shamir's secret sharing for sharing a secret. We use Shamir's secret sharing scheme for sharing a secret.

Definition C.1 (Shamir's Secret Sharing). Let q be a prime and $\mathbb{Z}_q$ be the group of integers modulo q. A (n, t, q)-Shamir's secret sharing scheme is an efficient randomized algorithm SSS that on input three integers n, t, q and a secret s, where $0 < t \leq n < q$ and $s \in \mathbb{Z}_q$, outputs n shares $s_1, \ldots, s_n \in \mathbb{Z}_q$ such that the following two conditions hold for any set $S = \{i_1, \ldots, i_\ell\}$:

- if $|S| \geq t$, there exists fixed (i.e., independent of s) integers $\lambda_1, \ldots, \lambda_\ell \in \mathbb{Z}_q$ (a.k.a. Lagrange coefficients) such that $\sum_{j=1}^{\ell} \lambda_j s_{i_j} = s \mod q$;
- if $\ell < t$, the distribution of $(s_{i_1}, \ldots, s_{i_\ell})$ is uniformly random.

For sharing a secret $s = (a, b)$ using Shamir's secret sharing scheme, choose (t-1) pairs (a_1, b_1), …, (a_{t-1}, b_{t-1}) uniformly at random from $\mathbb{Z}_q \times \mathbb{Z}_q$. Let $f(x)$ be the $(t-1)$ degree polynomial $a + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_{t-1} \cdot x^{t-1}$ and $g(x)$ be the $(t-1)$ degree polynomial $b + b_1 \cdot x + b_2 \cdot x^2 + \ldots + b_{t-1} \cdot x^{t-1}$. For all $i \in [n]$, the i^{th} share is generated as $s_i = (f(i), g(i))$.

C.2 Commitment

Definition C.2. A (non-interactive) secure commitment scheme Σ consists of two PPT algorithms $(\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ which satisfy hiding and binding properties:

- $\mathsf{Setup}_{\mathsf{com}}(1^\kappa) \rightarrow pp_{\mathsf{com}}$: It takes the security parameter as input, and outputs some public parameters.
- $\mathsf{Com}(m, pp_{\mathsf{com}}; r) =: \alpha$: It takes a message m, public parameters pp_{com} and randomness r as inputs, and outputs a commitment α.

Hiding. A commitment scheme $\Sigma = (\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ is hiding if for all PPT adversaries $\mathcal{A}$, all messages m_0, m_1, there exists a negligible function negl such that for $pp_{\mathsf{com}} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^\kappa)$,

$$|\Pr[\mathcal{A}(pp_{\mathsf{com}}, \mathsf{Com}(m_0, pp_{\mathsf{com}}; r_0)) = 1] - \Pr[\mathcal{A}(pp_{\mathsf{com}}, \mathsf{Com}(m_1, pp_{\mathsf{com}}; r_1)) = 1]| \leq \mathsf{negl}(\kappa),$$

where the probability is over the randomness of $\mathsf{Setup}_{\mathsf{com}}$, random choice of r_0 and r_1, and the coin tosses of $\mathcal{A}$.

Binding. A commitment scheme $\Sigma = (\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ is binding if for all PPT adversaries $\mathcal{A}$, if $\mathcal{A}$ outputs m_0, m_1, r_0 and r_1 $((m_0, r_0) \neq (m_1, r_1))$ given $pp_{\mathsf{com}} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^\kappa)$, then there exists a negligible function negl such that

$$\Pr[\mathsf{Com}(m_0, pp_{\mathsf{com}}; r_0) = \mathsf{Com}(m_1, pp_{\mathsf{com}}; r_1)] \leq \mathsf{negl}(\kappa),$$

where the probability is over the randomness of $\mathsf{Setup}_{\mathsf{com}}$ and the coin tosses of $\mathcal{A}$.

Definition C.3 (Trapdoor (Non-interactive) Commitments.). Let $\Sigma = (\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ be a (non-interactive) commitment scheme. A trapdoor commitment scheme has two more PPT algorithms SimSetup and SimOpen:

- $\mathsf{SimSetup}(1^\kappa) \rightarrow (pp_{\mathsf{com}}, \tau_{\mathsf{com}})$: It takes the security parameter as input, and outputs public parameters pp_{com} and a trapdoor τ_{com}.
- $\mathsf{SimOpen}(pp_{\mathsf{com}}, \tau_{\mathsf{com}}, m', (m, r)) =: r'$: It takes the public parameters pp_{com}, the trapdoor τ_{com}, a message m' and a message-randomness pair (m, r), and outputs a randomness r'.

For every (m, r) and m', there exists a negligible function negl such that $pp_{\mathsf{com}} \approx_{stat} pp'_{\mathsf{com}}$, where $pp_{\mathsf{com}} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^\kappa)$ and $(pp'_{\mathsf{com}}, \tau_{\mathsf{com}}) \leftarrow \mathsf{SimSetup}(1^\kappa)$; and

$$\Pr\left[\mathsf{Com}(m, pp'_{\mathsf{com}}; r) = \mathsf{Com}(m', pp'_{\mathsf{com}}; r')\right] \geq 1 - \mathsf{negl}(\kappa),$$

where $r' := \mathsf{SimOpen}(pp'_{\mathsf{com}}, \tau_{\mathsf{com}}, m', (m, r))$ and $(pp'_{\mathsf{com}}, \tau_{\mathsf{com}}) \leftarrow \mathsf{SimSetup}(1^\kappa)$.

Remark C.4. *Clearly, a trapdoor commitment can be binding against PPT adversaries only.*

C.2.1 Concrete instantiations. Practical commitment schemes can be instantiated under various settings. Here we specicially use Pederson's commitment.

Pedersen commitment. A popular commitment scheme secure under DLOG is Pedersen commitment. Here, $\mathsf{Setup}_{\mathsf{com}}(1^\kappa)$ outputs the description of a (multiplicative) group G of prime order $p = \Theta(\kappa)$ (in which DLOG holds) and two randomly and independently chosen generators g, h. If $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$ is a collision-resistant hash function, then a commitment to a message m is given by $g^{\mathcal{H}(m)} \cdot h^r$, where $r \leftarrow_\$ \mathbb{Z}_p$. A trapdoor is simply the discrete log of h with respect to g. In other words, SimSetup picks a random generator g, a random integer a in $\mathbb{Z}_p^\star$ and sets h to be g^a. Given (m, r), m' and a, SimOpen outputs $[(\mathcal{H}(m) - \mathcal{H}(m'))/a] + r$. It is easy to check that commitment to m with randomness r is equal to the commitment to m' with randomness r'.

C.3 Non-interactive Zero-knowledge

Let $\mathfrak{R}$ be an efficiently computable binary relation. For pairs $(s, w) \in \mathfrak{R}$, we refer to s as the instance and w as the witness. If it is computationally hard to determine witness from a statement, then the relation is called a *hard relation*. For hard relations we define non-interactive zero-knowledge arguments of knowledge in the random oracle model based on the work of Faust et al. [25].

Definition C.5 (Non-interactive Zero-knowledge Argument of Knowledge). Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{\mathsf{poly}(\kappa)}$ be a hash function modeled as a random oracle. A secure NIZK for a binary hard relation $\mathfrak{R}$ consists of two PPT algorithms Prove and Verify with oracle access to $\mathcal{H}$ defined as follows:

- $\mathsf{Prove}^{\mathcal{H}}_{\mathfrak{R}}(s, w)$ takes as input a instance s and a witness w, and outputs a proof π if $(s, w) \in R$ and $\perp$ otherwise.
- $\mathsf{Verify}^{\mathcal{H}}_{\mathfrak{R}}(s, \pi)$ takes as input a instance s and a candidate proof π, and outputs a bit $b \in \{0, 1\}$ denoting acceptance or rejection.

These algorithms are required to satisfy the following properties:

- **Perfect completeness**: For any $(s, w) \in R$,

$$\Pr\left[\mathsf{Verify}^{\mathcal{H}}_{\mathfrak{R}}(s, \pi) = 1 \mid \pi \leftarrow \mathsf{Prove}^{\mathcal{H}}_{\mathfrak{R}}(s, w)\right] = 1.$$

- **Zero-knowledge**: There must exist a pair of PPT simulators $(\mathcal{S}_1, \mathcal{S}_2)$ such that for all PPT adversary $\mathcal{A}$,

$$\left|\Pr[\mathcal{A}^{\mathcal{H}, \mathsf{Prove}^{\mathcal{H}}_{\mathfrak{R}}}(1^\kappa) = 1] - \Pr[\mathcal{A}^{\mathcal{S}_1(\cdot), \mathcal{S}'_2(\cdot,\cdot)}(1^\kappa) = 1]\right| \leq \mathsf{negl}(\kappa)$$

  for some negligible function negl, where
  - $\mathcal{S}_1$ simulates the random oracle $\mathcal{H}$;
  - $\mathcal{S}'_2$ returns a simulated proof $\pi \leftarrow \mathcal{S}_2(s)$ on input (s, w) if $(s, w) \in R$ and $\perp$ otherwise;
  - $\mathcal{S}_1$ and $\mathcal{S}_2$ share states.

- **Argument of knowledge**: There must exist a PPT simulator $\mathcal{S}_1$ such that for all PPT adversary $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}^{\mathcal{A}}$ such that

$$\Pr\Big[(s, w) \notin R \text{ and } \mathsf{Verify}^{\mathcal{H}}_{\mathfrak{R}}(s, \pi) = 1 \mid (s, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_1(\cdot)}(1^\kappa); w \leftarrow \mathcal{E}^{\mathcal{A}}(s, \pi, Q)\Big] \leq \mathsf{negl}(\kappa)$$

  for some negligible function negl, where
  - $\mathcal{S}_1$ is like above;
  - Q is the list of (query, response) pairs obtained from $\mathcal{S}_1$.

Fiat-Shamir transform. Let (Prove, Verify) be a three-round public-coin honest-verifier zero-knowledge interactive proof system (a sigma protocol) with unique responses. Let $\mathcal{H}$ be a function with range equal to the space of the verifier's coins. In the random oracle model, the proof system $(\mathsf{Prove}^{\mathcal{H}}, \mathsf{Verify}^{\mathcal{H}})$ derived from (Prove, Verify) by applying the Fiat-Shamir transform satisfies the zero-knowledge and argument of knowledge properties defined above. See Definition 1, 2 and Theorem 1, 3 in Faust et al. [25] for more details. (They actually show that these properties hold even when adversary can ask for proofs of false instances.)

Specific Instances. NIZK proofs are only needed for our strongly secure construction (Fig. 12). In particular, we need Sigma NIZK proofs two different relations $\mathfrak{R}_a$ and $\mathfrak{R}_{a,b}$ as described in the construction. We describe the specific constructions in Appendix F

D STRONGLY SECURE HISE

D.1 Definitions

We provide the definitions for strongly secure HiSE.

Definition D.1 (Strong Security of HiSE). A *Hi*SE scheme is said to be strongly-secure if it satisfies *strong correctness, message-privacy (Def 5.4) and strong authenticity*. Strong-correctness is the same as correctness (Def. 5.3), except that the challenge oracle is replaced by $O^{\mathsf{hs\text{-}str\text{-}cor\text{-}ch}}$, described in Figure 10. Strong authenticity is the same as authenticity (Def. 5.5), except that the challenge oracle is replaced by $O^{\mathsf{hs\text{-}st\text{-}au\text{-}ch}}$, described in Figure 11.

Intuitively, a HiSE scheme is strongly-correct whenever a legitimately produced ciphertext (by executing a DistGrEnc protocol, possibly in presence of malicious parties), when decrypted *honestly*, it must output the actual message (even $\neq \perp$).

For strong authenticity, instead of an honest $\mathsf{DistGrDec}^\dagger$, DistGrDec is run within the for loop – this is the only difference. This captures that even if the adversary takes part in a decryption procedure (albeit initiated by an honest party), it can not produce more ciphertexts, than are accounted for, that decrypt successfully (in presence of the adversary within that procedure).

D.2 Strongly secure HiSE construction

In this section we provide our strongly secure HiSE construction, which is achieved by extending the basic HiSE construction (Fig. 7 by adding a trapdoor commitment and a non-interactive zero-knowledge proof of knowledge similar to DiSE and ATSE. However, due to difference in the encryption and decryption (unlike DiSE or ATSE), we need to deploy two slightly different non-interactive sigma protocols (see Appendix C and for definitions and Appendix F

Oracle $\mathcal{O}^{\text{hs-str-cor-ch}}(j, S, j', S', \boldsymbol{m} = (m_1 \ldots, m_N), \omega)$:

require $j \in S \setminus C$ **and** $j' \in S' \setminus C$ **and** $\omega \in \{0,1\}^{\lceil \log N \rceil}$ **and** CHAL $= 0$ **and** OUT $= 0$.

- **set** CHAL $:= 1$.
- **run** $([j : \text{op}], \ldots) \leftarrow \text{DistGrEnc}(pp, [\![sk]\!]_S, [j : \boldsymbol{m}, S])$.
- **if** op $= \perp$ **then set** OUT $:= 0$;
  else set $\widehat{C} :=$ op **and do** :
  - **run** $([j' : \text{op}'], \ldots) \leftarrow \text{DistGrDec}^{\dagger}(pp, [\![sk]\!]_S, [j' : \widehat{C}_\omega, S'])$.
  - **if** op$' = \boldsymbol{m}$ **then set** OUT $:= 0$; **else set** OUT $:= 1$.

Figure 10: The challenge oracle for the game strong-HiSE-Cor. The differences are marked by blue.

Oracle $\mathcal{O}^{\text{hs-st-au-ch}}(L_{\text{forge}})$:

- **set** $\ell := \lfloor \text{ct}/g \rfloor$, where $g := t - |C|$.
- **set** $\left((j_1, S_1, \widehat{C}_{\omega_1}), (j_2, S_2, \widehat{C}_{\omega_2}) \ldots)\right) := L_{\text{forge}}$;
  require CHAL $= 0$ **and** SUCC $= 0$ **and** $j_1, j_2, \ldots \notin C$ **and** $\forall\, i \neq i' : \widehat{C}_i \not\preceq \widehat{C}_{i'}$.
- **set** CHAL $:= 1$ **and** $L_{\text{succ}} := \emptyset$ **and set** $\gamma := |L_{\text{forge}}|$.
- **for** $\forall\, i \in [\gamma]$:
  - **run op** $\leftarrow \text{DistGrDec}(pp, [\![sk]\!]_n, [j_i : \widehat{C}_i, S_i])$.
  - $\forall\, k \in [|\textbf{op}|]$: **if** op$_k \neq \perp$ **then set** $L_{\text{succ}} := L_{\text{succ}} \cup \{\text{op}_k\}$;
- **if** $|L_{\text{succ}}| > \ell$ **then set** SUCC $:= 1$.

Figure 11: Challenge/forgery oracle for strong-HiSE-Auth.

for concrete constructions). We present the construction in Fig. 12 and highlight the changes from Fig. 7 using blue. We formalize the security in the following theorem. We show how the proof of the base construction can be extended in Appendix E.

Theorem D.2. *Our extended construction (Fig. 12) is a strongly secure HiSE scheme assuming XDH and ℓ-masked BDDH assumptions on the underlying bilinear pairing in the random oracle model plus the security of the underlying trapdoor commitment scheme and the NIZK proof systems.*

E MISSING PROOFS

We provide the proofs that are missing from main body here. We mainly focus on the analysis for our base construction (Theorem 6.1), and discuss extension to the strongly secure construction (Theorem D.2) within each separate proof. The proofs follow the overall structures of ATSE and DiSE, but differs significantly in some key aspects. We mostly focus on the key ideas here.

Consistency. For both the constructions consistency is straightforward to observe due to the properties of Shamir secret sharing. We omit the formal details.

E.1 Proof of (Strong)-Correctness

A HiSE scheme is said to be correct if whenever an honest party j initiates the encryption protocol DistGrEnc on an input $(\boldsymbol{m} := m_1, \ldots, m_N, S)$ to generate a ciphertree $\widehat{C} := (j, X, \varepsilon, (\omega_1, c_1), \ldots, \omega_N, c_N))$, then any other honest party j' on running the decryption protocol DistGrDec for any $\omega \in \{0,1\}^{\leq d}$ on $\widehat{C}_\omega \preceq \widehat{C}$ either recovers $\boldsymbol{m}$ or $\boldsymbol{m}^*$ with high probability, where $\boldsymbol{m}^*$ contains the set of plaintext messages from $\boldsymbol{m}$ where some or all can be $\perp$. In particular, even if a PPT adversary is involved in the execution via corrupting any fixed set of (up to) $t-1$ parties during the procedure, it is not feasible for the adversary to enforce j' to output another $\boldsymbol{m}'^*$ where $\boldsymbol{m}' \neq \boldsymbol{m}$ as a result of decrypting $\widehat{C}_\omega$. This property is achieved by our construction due to the **binding** of the underlying merkle tree commitment scheme. Recall that during decryption the initiating party interactively derives the value z. If instead an incorrect value z' is derived, the decryption yields messages in some $\boldsymbol{m}' \neq \boldsymbol{m}$. $\boldsymbol{m}'^*$ is output as the correct plaintext message only if $\mathcal{H}_1(m_k, r_k) = \mathcal{H}_1(m'_k, r'_k)$ for all k for which $\perp \neq m'_k \in \boldsymbol{m}'^*$, i.e., two different messages open to some commitments part of the subtree rooted at ω. This contradicts the binding nature of the Merkle tree commitment scheme. Furthermore, **strong-correctness** is ensured since an honest execution of the decryption protocol results in a **correct computation of the mask** $\mathcal{H}(j, N, X_\varepsilon)^\alpha$ – a proof for which is analogous to the DPRF correctness in DiSE and thus always outputs exactly $\boldsymbol{m}$ and never outputs $\boldsymbol{m}^*$ where decryption of some or all of the ciphertexts may result in $\perp$.

E.2 Proof for Message Privacy

The message-privacy proof relies on (i) XDH and (ii) ℓ-masked BDDH over the bilinear group in the random oracle model. Now, from Def. 5.4, we note that the objective is to show that, for any $\kappa, n, t \in \mathbb{N}$, the security games HiSE-MsgPriv$(1^\kappa, 1^n, 1^t, 0)$ (in short HiSE-MsgPriv(0)) and HiSE-MsgPriv$(1^\kappa, 1^n, 1^t, 1)$ are computationally indistinguishable. Similar to ATSE [20] we can show this via a number of computationally indistinguishable hybrids starting from HiSE-MsgPriv$(1^\kappa, 1^n, 1^t, 0)$ and eventually reaching HiSE-MsgPriv$(1^\kappa, 1^n, 1^t, 1)$ and use these computational assumptions to show that the consecutive hybrids are computationally indistinguishable – this requires constructing a reduction to the corresponding assumption, while simulating the oracles correctly. We briefly describe the key ideas here. The detailed proof can be extended following the footsteps of ATSE and DiSE.

First **we use XDH** to move to a hybrid Hyb_1 from game HiSE-MsgPriv(0) where the challenge oracle is simulated using g_2^ν for a uniform random $\nu \in \mathbb{Z}_q$ in place of $\mathcal{H}_2(j, X_\varepsilon)^\alpha$. For simplicity we just assume that exactly $t-1$ parties are corrupt (similar to ATSE proofs) – this can be extended to a general case following DiSE [7]. Furthermore, we assume that the adversary does not make any query to oracles $\mathcal{O}^{\text{hs-mp-en}}$ or $\mathcal{O}^{\text{hs-mp-dc}}$. It is again straightforward to extend the proof to handle that, by q-many hybrids to go to Hyb_1 using *XDH* at each step instead of only one hybrid, when q is the total number of queries to those oracles, where in Hyb_1 all such queries would be handled by encrypting with a uniform random element from $\mathbb{G}_T$. In this simplified case the XDH reduction works as follows:

- receive the XDH challenge $g_1, g_2, g_2^a, g_2^b, g_2^c$ where c is either ab or uniform random in $\mathbb{Z}_q$;
- program the RO query $\mathcal{H}_2(X_\varepsilon) = g_2^b$, and implicitly let $a = \alpha$, then $\mathcal{H}_2(j, X_\varepsilon)^\alpha = g_2^{ab}$;

Ingredients and Parameters

Public parameters:

- The security parameter κ.
- An efficiently computable Type-3 bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are multiplicative groups and each of prime order q; g_1 and g_2 are randomly chosen generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively.
- Descriptions of hash functions $\mathcal{H}_1 : \{0,1\}^* \to \{0,1\}^\kappa$; $\mathcal{H}_2 : \{0,1\}^* \to \mathbb{G}_2$; $\mathcal{H}_3 : \mathbb{G}_T \to \{0,1\}^{\mathsf{poly}(\kappa)}$; $\mathcal{H}_4 : \mathbb{G}_1 \to \{0,1\}^{\mathsf{poly}(\kappa)}$; $\mathcal{H}' : \{0,1\}^* \to \{0,1\}^{\mathsf{poly}(\kappa)}$.

Merkle-Tree Commitment. Consists of algorithms $(\mathsf{MTCom}, \mathsf{MTOpen}, \mathsf{MTVer})$

Trapdoor Commitments. Consists of algorithms $(\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$.

Hard Relations.

- $\mathfrak{R}_a$ which consists of public parameters $(pp_{\mathsf{com}}, \mathcal{H}_2)$, instance (γ, z, x), witness (a, ν) such that: $\gamma = \mathsf{Com}(a, pp_{\mathsf{com}}; \nu) \wedge z = \mathcal{H}_2(x)^a$.
- $\mathfrak{R}_{a,b}$ which consists of public parameters $(pp_{\mathsf{com}}, \mathcal{H}_2)$, instance $(\gamma_a, \gamma_b, z, x, y)$, witness (a, b, ν_a, ν_b) such that: $\gamma_a = \mathsf{Com}(a, pp_{\mathsf{com}}; \nu_a) \wedge \gamma_b = \mathsf{Com}(b, pp_{\mathsf{com}}; \nu_b) \wedge z = \mathcal{H}_2(x)^a \mathcal{H}_2(y)^b$.

Non-interactive argument of knowledge (Def. C.5). Consists of algorithms $(\mathsf{Prove}^{\mathcal{H}'}_{\mathfrak{R}}, \mathsf{Verify}^{\mathcal{H}'}_{\mathfrak{R}})$.

Construction

$\mathsf{Setup}(1^\kappa, 1^n, 1^t) \to (pp, [\![sk]\!]_{[n]})$:

1. Choose $\alpha, \beta \leftarrow\!\!\$\ \mathbb{Z}_q$.
2. Compute the secret shares $\{(\alpha_i, \beta_i)\}_{i\in[n]}$ of pair (α, β) using (q, n, t)-SSS.
3. Run $\mathsf{Setup}_{\mathsf{com}}(1^\kappa)$ to get pp_{com}. For all $i \in [n]$: compute $\gamma_{\alpha,i} := \mathsf{Com}(\alpha_i, pp_{\mathsf{com}}; \nu_{\alpha,i})$ and $\gamma_{\beta,i} := \mathsf{Com}(\beta_i, pp_{\mathsf{com}}; \nu_{\beta,i})$ by picking random $\nu_{\alpha,i}$ and $\nu_{\beta,i}$.
4. Set $sk_i := (\alpha_i, \beta_i, \nu_{\alpha,i}, \nu_{\beta,i})$
5. Set $pp := (g_1^\beta, \gamma_{\alpha,1}, \ldots, \gamma_{\alpha,n}, \gamma_{\beta,1}, \ldots, \gamma_{\beta,n}, pp_{\mathsf{com}})$.

$\mathsf{DistGrEnc}(pp, [\![sk]\!]_{[n]}, [j : \boldsymbol{m}, S]) \to ([j : \widehat{C}/\perp], [S : N, X_\varepsilon])$: This is a two-round distributed protocol, that works as follows:

- **Pre-computation (local).** Let $(m_1 \ldots, m_N) := \boldsymbol{m} \in \{0,1\}^*$ and $N = 2^d$. Party j chooses a randomnesses $\rho_k \leftarrow\!\!\$\ \mathbb{Z}_q$ and $r_k \leftarrow\!\!\$\ \mathbb{Z}_q$ for all $k \in [N]$ and then generates the Merkle-Tree commitment $\boldsymbol{X} := \mathsf{MTCom}(y_1, \ldots, y_N)$ where $y_k = \mathcal{H}_1(m_k, \rho_k, \mathcal{H}_4(g_1^{r_k}))$. Let X_ε be the root of $\boldsymbol{X}$.
- **Round 1.** Party-j sends (N, X_ε) to all parties in set S.
- **Round 2.** Each party $i \in S$ sends back to party-j: (z_i, π_i) where $z_i := \mathcal{H}_2(j, N, X_\varepsilon)^{\alpha_i}$ and $\pi_i \leftarrow \mathsf{Prove}^{\mathcal{H}'}_{\mathfrak{R}_\alpha}((\gamma_{\alpha,i}, z_i, (j, N, X_\varepsilon)), (\alpha_i, \nu_{\alpha,i}))$
- **Finalize (local).** Party-j, once receives at least $t-1$ responses (z_i, π_i) such that $\mathsf{Verify}^{\mathcal{H}'}_{\mathfrak{R}_\alpha}((\gamma_{\alpha,i}, z_i, (j, N, X_\varepsilon)), \pi_i)$ returns 1, then computes $z := \prod_i z_i^{\lambda_i}$ for appropriately chosen λ_is (otherwise output $\perp$). Then do as follows for all $k \in [N]$:
  - Let $\omega := \mathsf{Bin}(k-1, d)$ be the binary vector encoding (of length d bits).
  - Compute ciphertext $c_k := (g_1^{r_k}, \mathcal{H}_2(X_{\omega|_1})^{r_k}, \mathcal{H}_2(X_{\omega|_2})^{r_k}, \ldots, \mathcal{H}_2(X_\omega)^{r_k}), (m_k, \rho_k, \mathcal{H}_4(g_1^{r_k})) \oplus \mathcal{H}_3(e(g_1^{r_k}, z)))$
  - Set $\omega_k := \omega$.

  Output the cipher-tree $\widehat{C} := (j, \boldsymbol{X}, \varepsilon, (\omega_1, c_1), \ldots, (\omega_N, c_N))$.

$\mathsf{DistGrDec}(pp, [\![sk]\!]_S, [j : \widehat{C}, S]) \to ([j : \boldsymbol{m}/\perp], [S : N, X_\varepsilon, X_\omega])$: Parse $\widehat{C}$ as $(j, \boldsymbol{X}, \omega, (\omega_1, c_1), \ldots, (\omega_\ell, c_\ell))$ and let N be the number of leaves in $\boldsymbol{X}$, then do as follows:

- **Round 1.** Party-j sends $(j, N, X_\varepsilon, X_\omega)$ to all parties $i \in S$.
- **Round 2.** Each party-i, on receiving $(j, N, X_\varepsilon, X_\omega)$, computes $\tilde{z}_i = \mathcal{H}_2(j, N, X_\varepsilon)^{\alpha_i} \mathcal{H}_2(X_\omega)^{\beta_i}$ and $\tilde{\pi}_i \leftarrow \mathsf{Prove}^{\mathcal{H}'}_{\mathfrak{R}_{\alpha,\beta}}((\gamma_{\alpha,i}, \gamma_{\beta,i}, \tilde{z}_i), (j, N, X_\varepsilon), X_\omega), (\alpha_i, \beta_i, \nu_{\alpha,i}, \nu_{\beta,i})$ and sends $(\tilde{z}_i, \tilde{\pi}_i)$ back to party j.
- **Finalize (local).** On receiving $\tilde{z}_i$ from at least $t-1$ parties, such that $\mathsf{Verify}^{\mathcal{H}'}_{\mathfrak{R}_{\alpha,\beta}}((\gamma_{\alpha,i}, \gamma_{\beta,i}, \tilde{z}_i), (j, N, X_\varepsilon), X_\omega), \tilde{\pi}_i)$ returns 1 it computes $\tilde{z} = \prod_i \tilde{z}_i^{\lambda_i}$ (otherwise output 1) including its own locally computed value. Then for each $k \in [\ell]$ do as follows:
  - Let $\omega' := \omega_k$.
  - Parse c_k as $(R, S_{\omega'|_1}, S_{\omega'|_2}, \ldots, E)$.
  - Compute $D := e(R, \tilde{z}) \cdot e(pp, S_\omega)^{-1}$
  - Compute $(m_k, \rho_k, h_k) := E \oplus \mathcal{H}_3(D)$.
  - Verify if $X_{\omega'} = \mathcal{H}_1(m_k, \rho_k, h_k)$ and if $\mathcal{H}_4(R) = h_k$. If either fails set $m_k := \perp$.

  Finally verify the tree $\mathsf{MTVer}(\boldsymbol{X})$, if it fails then return $\perp$, else output $(m_1, \ldots, m_\ell)$.

Figure 12: Our strongly secure HiSE construction

- clearly for a random c, the hybrid Hyb_1 is simulated, and if $c = ab$ HiSE-MsgPriv(0) is simulated;

In the next hybrid Hyb_2, we use a $\tilde{z} = g_2^{\nu'}$ for uniform random ν' instead of $\tilde{z} = g_2^{\nu} \mathcal{H}_2(X_\omega)^\beta$ for each challenge-decryption oracle query (cf. Fig. 5). We show a reduction to argue that hybrids Hyb_1 and Hyb_2 are computationally indistinguishable. The reduction, on receiving $g_1^a, g_1^b, \{g_2^{d_i}, g_2^{c+bd_i}\}_{i\in[\ell]}$ for and a challenge $h \in \mathbb{G}_T$ from the challenger works as follows:

- Set $g_1^{\beta} := g_1^b$ and send g_1^b to the adversary as pp.
- Compute $g_T^c := e(g_1, g_2^{c+bd_i})/e(g_1^b, g_2^{d_i})$ and set $e(g_1^{\alpha}, \mathcal{H}(j, X_\varepsilon)) := g_T^c$. Random oracle query on $\mathcal{H}_2(j, X_\omega)$ is programmed as $g_2^{r_\alpha}$ for a uniform random r_α. Note that, this does not require knowledge of g_1^{α}.
- In the challenge oracle, on receiving two message vectors $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$, create the list $\mathcal{M}$ by appending messages m_i such that $\boldsymbol{m}_0[i] = \boldsymbol{m}_1[i] = m_i$.
- Now create the challenge ciphertree $\widehat{C}^{\star}$ as follows:
  - For every node ω such that $\widehat{\boldsymbol{M}}_\omega \subset \mathcal{M}$, and the parent ω' of ω such that $\widehat{\boldsymbol{M}}_{\omega'} \not\subset \mathcal{M}$ then each message m_i within that message tree is encrypted as: $R_i := g_1^{r_i}, \ldots, S_{i,\omega|_j} := g_2^{d_{\omega|_j} r_i}, \ldots, E_i = \mathcal{H}_3(g_T^{cr_i}) \oplus (m_i, \rho_i, \mathcal{H}_4(R_i))$ where r_i, ρ_i are sampled uniformly and random oracles are programmed $\mathcal{H}_2(X_{\omega|_j}) = g_2^{d_{\omega|_j}}$. Here we assume $d_{\omega|_j}$ is re-indexed from the given d_is.
  - For all other messages, encrypt as: $R_i := g_1^{r_i}, \ldots, S_{i,\omega|_j} := g_2^{\delta_{\omega,j} r_i}, \ldots, E_i := h^{\delta_i} \oplus (m_i, \rho_i, \mathcal{H}_4(R_i))$ where $g_1^{r_i} := g_1^{\delta_i a}$ for uniform random δ_i, $\delta_{\omega,j}$ are chosen uniformly at random (note from above that the random oracle is programmed as $\mathcal{H}_2(X_{\omega|_j}) = g_2^{\delta_{\omega,j}}$).
- The queries to $O^{\text{hs-mp-ch-dc}}$ for $\widehat{C}_\omega$ are simulated by responding $g_2^{c+bd_i} = g_2^{\alpha+\beta d_{\omega|_j}}$; where the random oracle queries are programmed as $\mathcal{H}_2(X_\omega) = g_2^{d_{\omega|_j}}$. Here it is important to note that, this a legitimate query must have a ω for which $\widehat{\boldsymbol{M}}_\omega \subseteq \mathcal{M}$. Hence the given $g_2^{d_{\omega|_j}}$ values can be used as Step E.2.

Now, observe that if $h = g_T^{ac}$, then the reduction simulates Hyb_1 because for any message $m_i \in \widehat{\boldsymbol{M}}_\omega$ such that $\widehat{\boldsymbol{M}}_\omega \subseteq \mathcal{M}$ and for the parent ω' of ω, $\widehat{\boldsymbol{M}}_{\omega'} \not\subseteq \mathcal{M}$: $g_T^{ac\delta_i} = e(g_1^{r_i}, \mathcal{H}_2(j, X_\varepsilon)^{\alpha})$ is used to mask the messages, whereas when h is uniform random the element h^{δ_i} is just another uniform random element in $\mathbb{G}_T$, so in that case reduction simulates Hyb_2.

Now, in Hyb_2, all messages that are distinct in $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ are information theoretically hidden. Same steps follow from HiSE-MsgPriv(1). This concludes the proof.

In the strong HiSE construction, the zero-knowledge proofs have to be replaced by simulated proofs. The rest of the proof stays virtually the same.

E.3 Proof for (Strong)-Authenticity.

We note that, an adversary $\mathcal{A}$ breaks authenticity if it produces at least one more ciphertext than ℓ in oracle $O^{\text{hs-au-ch}}$. We discuss the following "bad"events

- $\mathbf{E}_1$. This happens when $\mathcal{A}$ wins by sending a (j, N', X_ε) within $O^{\text{hs-au-en}}$ while X_ε is root of a message-tree with $N > N'$ messages. This happens only with negligible probability as, to win the game the corresponding cipher-tree $\widehat{C}$ must decrypt correctly. Since, in the decryption algorithm, N is chosen honestly from inspecting the ciphertree, this would lead to wrong decryption.
- $\mathbf{E}_2$. This happens when $\mathcal{A}$ is able to create more than one ciphertexts, c_1, c_2 for a single message m using from a single interaction, but using two different randomnesses r_1, r_2. This also happens with negligible probability, because the value $R = g_1^r$ is committed via $\mathcal{H}_4(R)$ to construct the message-tree and hence X_ε. In other words, except with negligible probability (when collision happens for hash function $\mathcal{H}_4$), X_ε uniquely fixes $R = g_1^r$.

Now, note that, in our construction each time the adversary becomes able to compute a fresh $\mathcal{H}_2(j, N, X_\varepsilon)^{\alpha}$ (that is for a fresh (j, N, X_ε) triple), N is added to ℓ – this is irrespective of the fact that whether the components of z are learned through encryption or decryption. Now, when $\mathbf{E}_1 \vee \mathbf{E}_2$ does not happen, that together implies that the only way to break the one-more security is to compute $\mathcal{H}_2(j, N, X_\varepsilon)^{\alpha}$ for a triple (j, N, X_ε) such that adversary has $< t$ partial values $\mathcal{H}_2(j, N, X_\varepsilon)^{\alpha_i}$. We can show that as long as XDH holds in $\mathbb{G}_2$, this holds. Let us outline a reduction for a simpler case when $|C| = t - 1$:

- Receive from the XDH challenger: $g_2^a, g_2^b, g_1, h \in \mathbb{G}_2$. Choose β and give $pp = g_1^{\beta}$ to the adversary.
- Guess a random oracle query on (j, N, X_ε) and program $\mathcal{H}_2(j, N, X_\varepsilon) = g_2^a$.
- For all other RO queries on $(j', N', X'_\varepsilon) \neq (j, N, X_\varepsilon)$, choose random r' to program $\mathcal{H}_2(j, N, X_\varepsilon) = g_2^{r'}$.
- Implicitly assume $\alpha = b$ and simulate encryption/decryption queries on $(j', N', X'_\varepsilon) \neq (j, N, X_\varepsilon)$ using g_2^b with known r'.
- Finally, when adversary responds with $> \ell$ valid ciphertexts, then look for the unique ciphertree $\widehat{C}$ corresponding to (j, N, X_ε). If not found, then abort. Otherwise, use $\mathcal{H}_3(e(R, h))$ to decrypt any ciphertext within $\widehat{C}$. If that succeeds, then conclude h as $g_2^{ab} = \mathcal{H}(X_\varepsilon)^{\alpha}$, else conclude h to be uniform in $\mathbb{G}_2$.

Now, if the hash functions used are collision resistant, then the probability of the reduction breaking XDH is $1/Q_{\mathcal{H}_2}$-th the probability of $\mathcal{A}$'s winning the authenticity game, where $Q_{\mathcal{H}_2}$ is the number of RO queries made to $\mathcal{H}_2$.

The case for $< t$ corruption and strong authenticity can be argued similar to [7, 20].

F SPECIFIC SIGMA NIZK PROOF SYSTEMS

Consider a group $\mathbb{G} = \langle g \rangle$ of prime order p where the discrete logarithm problem is hard. Let $\gamma_m = \mathsf{Com}(m, pp_{\mathsf{com}}; \nu_m) = g^m h^{\nu_m}$ denotes the Pedersen commitment of any $m \in \mathbb{Z}_p$ using the random string $\nu_m \leftarrow_{\$} \mathbb{Z}_p$. We provide the details of non-interactive sigma protocol for the following hard relation:

$$\mathfrak{R}_{a,b} = \{((\gamma_a, \gamma_b, z_{ab}, x, y), (a, b, \nu_a, \nu_b)) \; : \; \gamma_a = g^a h^{\nu_a} \wedge \gamma_b = g^b h^{\nu_b} \wedge z_{ab} = \mathcal{H}(x)^a \mathcal{H}(y)^b\}$$

Where $(\gamma_a, \gamma_b, z_{ab}, x, y)$ is the instance and (a, b, ν_a, ν_b) is the witness.

The non-interactive sigma protocol (with Fiat-Shamir), which is equipped with a hash function $\mathcal{H}' : \{0,1\}^* \to \mathbb{Z}_p$ (modeled as a andom oracle in the Fiat-Shamir heuristic) works as follows:

- The prover samples uniform random $\alpha_a, \alpha_{\nu_a}, \alpha_b, \alpha_{\nu_b}$ from $\mathbb{Z}_p$ and compute: $u_{\gamma_a} := g^{\alpha_a} h^{\alpha_{\nu_a}}$;

$u_{\gamma_b} := g^{\alpha_b} h^{\alpha_{\nu_b}}$;
$u_{z_{ab}} := \mathcal{H}(x)^{\alpha_a} \mathcal{H}(y)^{\alpha_b}$
- Then it computes $\beta := \mathcal{H}'(u_{\gamma_a}, u_{\gamma_b}, u_{z_{ab}})$
- Finally it computes:
$\delta_a := \alpha_a + a\beta$;
$\delta_{\nu_a} := \alpha_{\nu_a} + \nu_a \beta$;
$\delta_b := \alpha_b + b\beta$;
$\delta_{\nu_b} := \alpha_{\nu_b} + \nu_b \beta$;
- It then sends all of these computed values to the verifier.

- The verifier checkes the following equations:
$g^{\delta_a} h^{\delta_{\nu_a}} = \gamma_a^{\beta} u_{\gamma_a}$;
$g^{\delta_b} h^{\delta_{\nu_b}} = \gamma_b^{\beta} u_{\gamma_b}$;
$\mathcal{H}(x)^{\delta_a} \mathcal{H}(y)^{\delta_b} = z_{ab}^{\beta} u_{z_{ab}}$.
- It outputs 1 if and only if all of these equations satisfy, else outputs 0.

Next we outline how the sigma protocol satisfies completeness, knowledge soundness and zero-knowledge, and thereby a *non-interactive argument of knowledge* as per Definition C.5 in the random oracle model. We omit the details, which follow from the standard sigma protocol arguments.

Completeness. For an honest prover, the verification (by an honest verifier) always succeeds and the verifier accepts the proof with probability 1. Thus, the protocol satisfies *completeness.*

Knowledge Soundness. To prove *knowledge soundness* we construct an extractor $\mathcal{E}$, which rewinds the prover and provides two responses β and β', both uniform at random in $\mathbb{Z}_p$ for the same set of first round messages $u_{\gamma_a}, u_{\gamma_b}, u_{z_{ab}}$. It obtains two sets of responses in the third round from the prover: $\delta_a, \delta_{\nu_a}, \delta_b, \delta_{\nu_b}$ and $\delta'_a, \delta'_{\nu_a}, \delta'_b, \delta'_{\nu_b}$ It extracts the witnesses as follows:

- $a := (\delta_a - \delta'_a)(\beta - \beta')^{-1}$.
- $b := (\delta_b - \delta'_b)(\beta - \beta')^{-1}$.
- $\nu_a := (\delta_{\nu_a} - \delta'_{\nu_a})(\beta - \beta')^{-1}$.
- $\nu_b := (\delta_{\nu_b} - \delta'_{\nu_b})(\beta - \beta')^{-1}$.

Zero-knowledge. For an honest verifier, which chooses β uniformly at random from $\mathbb{Z}_p$, we construct a simulator as follows:

- Choose $\delta_a, \delta_b, \delta_{\nu_a}, \delta_{\nu_b} \beta$ uniformly at random from $\mathbb{Z}_p$ and then compute $u_{\gamma_a}, u_{\gamma_b}, u_{z_{ab}}$ from the four verification equations.
- Finally program the random oracle $\mathcal{H}'(u_{\gamma_a}, u_{\gamma_b}, u_{z_{ab}}) = \beta$.

Similarly one can construct a sigma protocol for the relation:

$$\mathfrak{R}_a = \{((\gamma_a, z_a, x), (a, \nu_a)) \ : \ \gamma_a = g^a h^{\nu_a} \wedge z_a = \mathcal{H}(x)^a\}$$

G PROOF OF ℓ-MASKED BDDH IN THE GENERIC GROUP MODEL

Boneh, Boyen and Goh extend the Generic Group Model defined by Shoup [36] to include the class of assumptions in pairs of groups having an efficiently computable bilinear pairing. They define the Generalized Diffie-Hellman problem in [12] and give a proof of its security for generic bilinear groups.

Let $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T = \langle g_T \rangle$ be cyclic groups of order p and let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be an efficiently computable bilinear map. We state below a slight variant of the generalized decisional Diffie Hellman Problem [12], as defined by Boyen in [15] to account for groups equipped with an asymmetric bilinear pairing.

Definition G.1 ((P,Q,R,f) - Decisional Diffie-Hellman Problem). Let $P = (P_1, \ldots, P_{s_1}) \in \mathbf{F}_p[X_1, X_2, \ldots, X_n]^{s_1}$, $Q = (Q_1, \ldots, Q_{s_2}) \in \mathbf{F}_p[X_1, X_2, \ldots, X_n]^{s_2}$ and $R = (R_1, \ldots, R_{s_3}) \in \mathbf{F}_p[X_1, X_2, \ldots, X_n]^{s_3}$ be tuples of n-variate polynomials over $\mathbf{F}_p$. For a polynomial $f \in \mathbf{F}_p[X_1, X_2, \ldots, X_n]$ and a random vector $(x_1, \ldots, x_n) \in \mathbf{F}_p^n$ define the vector

$$H(x_1, \ldots, x_n) = (g_1^{P(x_1,\ldots,x_n)}, g_2^{Q(x_1,\ldots,x_n)}, g_T^{R(x_1,\ldots,x_n)}) \in \mathbb{G}_1^{s_1} \times \mathbb{G}_2^{s_2} \times \mathbb{G}_T^{s_3} \tag{G.1}$$

The (P, Q, R, f)- *Decisional Diffie Hellman* problem for the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ is defined as:
Given the vector $H(x_1, \ldots, x_n) = (g_1^{P(x_1,\ldots,x_n)}, g_2^{Q(x_1,\ldots,x_n)}, g_T^{R(x_1,\ldots,x_n)})$, distinguish $g_T^{f(x_1,\ldots,x_n)}$ from a random element of $\mathbb{G}_T$.

The advantage of an algorithm $\mathcal{A}$ in solving the the above decisional problem is given by

$$Adv_{\mathcal{A}}^{(P,Q,R,f)\text{-}DDH} = \Big| \Pr\left[\mathcal{A}(H(x_1, \ldots, x_n), g_T^{f(x_1,\ldots,x_n)}) = 1\right] - \Pr\left[\mathcal{A}(H(x_1, \ldots, x_n), g_T^{\nu}) = 1 : \nu \leftarrow_{\$} \mathbf{F}_p\right] \Big| \tag{G.2}$$

We now state the notion of independence given in [12] for a set of polynomials (P, Q, R, f).

Definition G.2. Let $P \in \mathbf{F}_p[X_1, X_2, \ldots, X_n]^{s_1}$, $Q \in \mathbf{F}_p[X_1, X_2, \ldots, X_n]^{s_2}$ and $R \in \mathbf{F}_p[X_1, X_2, \ldots, X_n]^{s_3}$ be three tuples of n-variate polynomials over $\mathbf{F}_p$ having $P_1 = Q_1 = R_1 = 1$. A polynomial $f \in F_p[X_1, X_2, \ldots, X_n]$ is said to be dependent on the sets (P, Q, R) if there exist constants $\{u_{i,j}\}_{i,j}$, $\{v_{i,j}\}_{i,j}$, $\{w_{i,j}\}_{i,j}$, $\{t_k\}_k \in \mathbf{F}_p$ such that

$$f = \sum_i \sum_j u_{i,j} P_i Q_j + \sum_i \sum_j v_{i,j} Q_i Q_j + \sum_i \sum_j w_{i,j} P_i P_j + \sum_k t_k R_k \tag{G.3}$$

f is independent of (P, Q, R) if it is not dependent on the sets (P, Q, R).

The coefficients $\{v_{i,j}\}_{i,j}$, $\{w_{i,j}\}_{i,j}$ will be zero if efficiently computable isomorphisms $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ and $\psi^{-1} : \mathbb{G}_1 \to \mathbb{G}_2$ are not known.

An adversary, in the Generic Group Model(GGM), is provided with only random encodings of some group elements. For group operations and pairing computations, the adversary is given access to oracles. This limits the adversary from exploiting any structural property of the actual group representation. The security proof of an assumption in the GGM is structured as follows:

An algorithm $\mathcal{B}$ simulates the interaction of the adversary $\mathcal{A}$ with the group operation oracles. That is, $\mathcal{B}$ answers the queries made by $\mathcal{A}$ without the knowledge of the encodings, the actual assignment $(x_1, \ldots, x_n)$ and the bit b that were used to generate the random (P, Q, R, f)−DDH instance. Information about the bit b could be leaked to $\mathcal{A}$ in case it is not provided with a correct simulation. The simulation fails if the algorithm $\mathcal{B}$ while simulating the oracles ends up assigning different encodings to the same group

element. The advantage of the adversary is then bounded using the probability of the simulation failing. The details of this can be found in [12, 15, 36].

Let $\sigma_1, \sigma_2, \sigma_T : \mathbf{F}_p \to \{0,1\}^*$ be three random encodings (injective maps) corresponding to the groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ respectively. Then we have the following theorem from [12, 15]:

Theorem G.3. *Let $P \in \mathbf{F}_p[X_1, X_2, \ldots, X_n]^{s_1}, Q \in \mathbf{F}_p[X_1, X_2, \ldots, X_n]^{s_2}$ and $R \in \mathbf{F}_p[X_1, X_2, \ldots, X_n]^{s_3}$ be three tuples of n-variate polynomials over $\mathbf{F}_p$ having $P_1 = Q_1 = R_1 = 1$. Let $f \in F_p[X_1, X_2, \ldots, X_n]$ be independent of (P, Q, R). Then for any algorithm $\mathcal{A}$ that makes at most q oracle queries in total, we have:*

$$Adv_{\mathcal{A}}^{(P,Q,R,f)\text{-}DDH} = \left| \Pr\left[\begin{array}{c|c} \mathcal{A}(\sigma_1(P(x_1,\ldots,x_n)), & (x_1,\ldots,x_n) \leftarrow_{\$} \mathbf{F}_p, \\ \sigma_2(Q(x_1,\ldots,x_n)), & b \leftarrow_{\$} \{0,1\}, \\ \sigma_T(R(x_1,\ldots,x_n)), & v_b \leftarrow f(x_1,\ldots,x_n), \\ \sigma_T(v_0), \sigma_T(v_1)) = b & v_{1-b} \leftarrow_{\$} \mathbf{F}_p \end{array} \right] - \frac{1}{2} \right| \leq \frac{(q+2n+2)^2 d}{2p} \quad \text{(G.4)}$$

where $d = max(d_{PQ} = d_P + d_Q, d_R, d_f)$. d_S denoting the maximum of the total degrees of the polynomials in the set S.

Following the framework of BBG, for our assumption we have: $n = l + 3$, $d = 3$ and $P, Q, R, f \in \mathbf{F}_p[a, b, c, d_1, \ldots, d_l]$ where

- $P = (P_1, P_2, P_3) = (1, a, b)$
- $Q = (Q_1, \ldots Q_{2l+1}) = (1, d_1, \ldots, d_\ell, c + bd_1, \ldots c + bd_\ell)$
- $R = (R_1) = (1)$
- $f = ac$

So $PQ = (1, a, b, \{d_i\}_{i=1}^{\ell}, \{ad_i\}_{i=1}^{\ell}, \{bd_i\}_{i=1}^{\ell}, \{c+bd_i\}_{i=1}^{\ell}, \{ac+abd_i\}_{i=1}^{\ell}, \{bc+b^2d_i\}_{i=1}^{\ell})$.
We now state and prove the following lemma to be able to use theorem G.3.

Lemma G.4. *$f = ac$ is independent of the sets (P, Q, R).*

Proof. Suppose, to the contrary, that f is dependent on (P, Q, R) in the sense of definition G.2. Then f can be written as $f = \sum_i \sum_j u_{i,j} P_i Q_j + \sum_k t_k R_k$
In other words:

$$ac = u_{0,0} + \sum_{i=1}^{l} u_{0,i} d_i + \sum_{i=1}^{l} u_{0,l+i}(c + bd_i) + u_{1,0}a + \sum_{i=1}^{l} u_{1,i} ad_i + \sum_{i=1}^{l} u_{1,l+i}(ac + abd_i) + u_{2,0}b + \sum_{i=1}^{l} u_{2,i} bd_i + \sum_{i=1}^{l} u_{2,l+i}(bc + b^2 d_i) + t_0$$

Equivalently:

$$0 = ac(\sum_{i=1}^{l} u_{1,l+i} - 1) + (u_{0,0} + t_0) + \sum_{i=1}^{l} u_{0,i} d_i + \sum_{i=1}^{l} u_{0,l+i}(c + bd_i) + u_{1,0}a + \sum_{i=1}^{l} u_{1,i} ad_i + \sum_{i=1}^{l} u_{1,l+i} abd_i + u_{2,0}b + \sum_{i=1}^{l} u_{2,i} bd_i + \sum_{i=1}^{l} u_{2,l+i}(bc + b^2 d_i)$$

On equating the coefficients to 0, we arrive at a contradiction of the form $\sum_{i=1}^{l} u_{1,l+i} = 1$ and $u_{1,l+i} = 0$ for all $i \in [l]$, from the coefficients of the monomials $ac, abd_1, \ldots, abd_l$.
So our assumption is wrong and f must be independent of (P, Q, R). □

Having proven f independent of (P, Q, R), our assumption fits the (P, Q, R, f) Decisional Diffie-Hellman framework G.1 given by BBG and thus can be proven secure in GGM by directly invoking the above theorem.
From theorem G.3 we have,

$$Adv_{\mathcal{A}}^{\ell-masked\ BDDH} \leq \frac{3(q+2l+8)^2}{2p} \quad \text{(G.5)}$$