HiSE: Hierarchical (Threshold) Symmetric-key Encryption

Pousali Dey[1], Pratyay Mukherjee[1], Swagata Sasmal[1] and Rohit Sinha[2]

[1] Indian Statistical Institute, Kolkata, India

[2] Swirlds Labs, USA

Abstract. Threshold symmetric encryption (TSE) [DiSE, CCS 2018], provides a practical decentralized solution for symmetric encryption by distributing the secret-key *at all times*, thus avoiding a single point of attack or failure. TSE was further enhanced [ATSE, CCS 2021] by an amortization which enables a "more privileged" client to encrypt bulk records by interacting only once with the key servers, while decryption must be performed individually for each record, potentially by a "less privileged" client. However, a typical enterprise generates data once and queries it several times for various data analysis; 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.

Our work starts with an observation that a large and useful class of analytics queries access some time-windowed sequence of database records (e.g. log entries or user transactions). Can we offer faster decryption for such access patterns, without compromising the benefits of prior schemes?

To that end, we build a new TSE scheme that 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 structure, where *one* interaction is needed to decrypt all ciphertexts in a sub-tree, and thus only log-many for any arbitrary sub-sequence. Our scheme incorporates ideas from binary-tree encryption by Canetti et al. [Eurocrypt 2003] and carefully combines that with Merkle-tree commitments. We show that our scheme satisfies all essential TSE properties, such as correctness, privacy and authenticity for our notion, formalized as *hierarchical threshold symmetric-key encryption* (HiSE). 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 holds in the generic group model.

Our extensive implementation shows 10-65× improvement in latency and throughput over ATSE. HiSE can decrypt over 6K records/sec on server-grade hardware, but the logarithmic overhead in encryption (not decryption) only lets us encrypt up to 3K records/sec ($\sim$ 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 offers an acceptable practical trade-off in practice.

Keywords: threshold cryptography; threshold symmetric encryption

E-mail: deypousali95@gmail.com (Pousali Dey), pratyay85@gmail.com (Pratyay Mukherjee), swagata.sasmal@gmail.com (Swagata Sasmal), sinharo@gmail.com (Rohit Sinha)

1 Introduction

A large enterprise collects billions of events everyday, and later performs a variety of analytics on them. Since this data is business-sensitive these enterprises protect the data at rest using key management systems (KMS) that are often backed by hardware security modules (HSMs). However, HSMs require dedicated facilities with operational costs, have complex update procedures (e.g. with designated trusted personnel) and are susceptible to hardware side-channels [KHF+20, LSG+20].

Threshold Key Management System To address these shortcomings of HSMs, there is growing interest [Lin] 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 [vaub], Coinbase Custody [coi], and the HSM replacements by Unbound Tech [dya] – and is being standardized in an ongoing workstream by the U.S. National Institute of Standards and Technology (NIST) [BP23].

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, which exposes the application to attacks such as data corruption or poisoning. Existing TSE schemes [AMMR18, CGMS21a, ADL+22] 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 provisions 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 [AMMR18], required the application to interact with the KMS servers for encrypting or decrypting *each* message, which posed a clear scaling bottleneck[2]. Real-time applications, such as payment processing, are expected to handle over 50K events 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 [CGMS21a]

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

[2] As demonstrated in [CGMS21a], 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 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 [Visa, Visb].

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 it may be acceptable in certain settings, data-intensive workloads cannot tolerate this penalty on decryption; for example, enterprise analytics workloads are decryption-heavy, as data is created once but is queried and analyzed several times over its lifetime. In our experience attempting to deploy ATSE for an analytics workflow, we find the slow decryption to be a major bottleneck – 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 experimental data in Section 8).

Our Key Observation Our exploration in efficient decryption starts with an observation (folklore in industry) about typical enterprise analytics workloads, wherein the event data is stored as they arrive, in a *time-ordered sequence*, and later processed in chunks (e.g. by the minute or hour). That is, a typical query operates on a sequence of data records within a time window [med]. The importance of supporting such queries is evident in the rising demand for time-series database products [ris, KKD+23], which facilitate SQL-like queries on time-ordered event streams. For instance, a payment processor or ads platform may query for transactions in a specific one-hour window for computing various aggregate statistics [SOL+23, spl, ris], or a security organization may compute metrics over recent events for real-time decision making [spl]. This naturally results in large subsequences of read accesses. We can leverage this inherent structure 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/bandwidth.[5]

Objectives We seek a TSE scheme with the following properties:

1. **Bulk Encryption**: when encrypting a large set of records, the KMS interaction (in bandwidth and server computation) does not depend on the number of the records; i.e., the server performs constant (and concretely efficient) work. In HiSE, the client sends to the servers a short commitment computed over the message-set, 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 per record interaction.
2. **Bulk Subsequence Decryption**: when decrypting any contiguous range of records, the KMS interaction (in bandwidth and server computation) is sub-linear / 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

[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.

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.

Scheme	Operations	Encryption Client	Encryption Server	Decryption Client	Decryption Server
DiSE [AMMR18]	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 [CGMS21a]	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 (this paper)	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

via a game-based definition in Section 6.[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) provide the key material.

1.1 Contributions

Definition We introduce and formalize the notion of hierarchical threshold symmetric-key encryption (HiSE), which enables bulk encryption and subsequence decryption in group via a threshold symmetric cryptosystem. We capture the various security properties using game-based definitions in Section 6.

Construction We provide an efficient HiSE construction based on asymmetric bilinear pairing. Technically, our construction builds on ideas from binary-tree public-key encryptions [CHK03]. However, the main challenge was to convert the public-key primitive to symmetric-key one while maintaining the fine-grained access control, which necessitates novel ideas (see the exposition in Section 3) – this requires a new assumption (which holds in generic group) called ℓ-masked BDDH, in addition to well-known XDH over pairing. The concrete performance 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 12-70× and throughput improvement between 10-65×, compared to the ATSE decryption, when interacting with 6-24 KMS servers. 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 (4.5× slowdown compared to ATSE's encryption, yet $7-12\times$ faster than "parallelized DiSE", as benchmarked as a baseline in ATSE [CGMS21a]) and incurs roughly 500 bytes of ciphertext expansion per record. Nevertheless, we believe HiSE offers an acceptable tradeoff in practice.

[6] We note that, like prior works [AMMR18, CGMS21a] 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.

2 Related Works

Threshold Symmetric Encryption (TSE) Agrawal et al. provided the first formal treatment of a TSE technique in DiSE [AMMR18] using a *distributed pseudorandom function* based construction. For each encryption and decryption, the scheme requires interaction with the KMS servers. To address this shortcoming, Christodorescu et al. propose ATSE [CGMS21a], wherein the *flexible key derivation protocol* enables encryption of a group of messages with a single interaction. In short, the client commits to a set of messages and interactively derives 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 [BW13] and Naor-Pinkas-Reingold's *DPRF* construction [NPR99]. However, the decryption process in ATSE still requires the client 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, and even constant in some cases. Mukherjee [Muk20] extended the TSE notion to an adaptive corruption setting and proposed a construction that appears to incur about 2x computational overhead. Recently Duc et al. (DiAE [DMV22]) extended DiSE [AMMR18] using *encryptment* [DGRW18] 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 ideas 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 (public-key) encryption [CHK03] (BTE). We build on their construction making it work in the symmetric key setting. In BTE, each node has a corresponding secret key, which can be used to generate secret keys for its child nodes. We use this hierarchical structure on a set of (the commitments of) the messages, where the tree is essentially the corresponding Merkle tree (MT) – each leaf is corresponding to a message, whereas an inner node corresponds to a sequence of messages. We encrypt at the root and decrypt at any intermediate node using the corresponding secret keys. The decryption capability of a node increases with its height in the tree and the secret key at each node decrypts ciphertexts corresponding to the subtree rooted at that node and *nothing beyond* it. However, our construction is more similar to the recent work iTire [BMS22], which gets rid of the "key-delegation" property as well , as it is sufficient to ensure that a key at an inner-node decrypts the ciphertexts under that node. Similar constructions are used for different HIBE [BF01, BSS05, BBG05, GS02] and ABE schemes [SW05, GPSW06, BSW07] in the literature. In particular, the work [BSS05] 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. [BBG05] HIBE and Baek et al.'s IBE [BSS05], Chang et al [CLH10] 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 these work built public-key primitives and 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.

Threshold PKE A large body of work considers threshold public-key encryption [CG99, SDFY94, DP08, Fra90, NPR99, SG98] for various applications, such as oblivious key management [JKR19]. In our setting, one potential approach could be to combine threshold

PKE along with threshold signatures (such as BLS [BLS01]) to obtain a TSE scheme, for example, by locally encrypting and then using threshold signing in the encryption procedure, and during decryption first verify locally, and then use threshold decryption. Nevertheless, it is unclear how to perform fine-grained batched decryption without revealing all messages. This is because threshold decryption for each individual ciphertext would make both the server's computation and communication linear in the number of ciphertexts.

For more work on threshold cryptography [JKR19, DF90, BEL+20, AMMM18, MSJ02] and multi-party computation [KOR+17], we refer to ATSE [CGMS21a] and DiSE [AMMR18].

3 Overview

We start from a similar observation as the one in ATSE [CGMS21a], but we focus on the largely understudied part of the data life-cycle: decryption. More concretely, our work occurs in service of *windowed queries* [Amaa] – i.e., queries that process events within a window (e.g. a time window). Such queries are commonplace in data analytics pipelines of modern enterprises [VGTM23], promoting cloud providers to offer relevant services [Amaa, Amac].

Windowed queries result in sub-sequence access patterns, as they require decryption of a contiguous range of records within an ordered dataset. HiSE leverages the inherent hierarchy or partial ordering in the access pattern structure to perform sub-linear number of KMS interactions (each interaction requiring constant computation and bandwidth). 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 motivating use case.

3.1 Use Case: Windowed Queries

Our research draws inspiration from the data processing pipelines that we observed in major payments processors, like Visa [Visc]; that said, we found such data processing architectures to be broadly adopted by enterprises that handle sensitive data.

Data Storage Enterprises gather source events containing sensitive user data, such as payment transactions or ad conversions. Before these records are processed by various data (ETL) pipelines, the source application must first ensure data-at-rest protection by encrypting them with the aid of a Key Management Service (KMS). To that end, the following architectural pattern is quite common. At the data ingress point, a dedicated application performs a single function: it encrypts records upon arrival and forwards them to a storage service. Because this application handles plaintext user data, we refer to it as a "privileged client" or *encryptor*; it is intentionally designed for simplicity and fortified with standard security measures. Subsequently, different analytics workloads access the encrypted data. In practice, since numerous teams within the organization develop these downstream analytics, concerns related to security, regulation, and compliance typically compel us to restrict the data they can access. It is also necessary to maintain an audit trail that details which records have been accessed by each query. The implementation of both the access control and the audit trail occurs at the KMS layer.

Data Access Recall that, the encryptor outputs a continuous stream of records. A typical analytics query will need to access a specific subsequence of this data stream. For instance, a metrics application might periodically compute aggregate statistics over a one-minute interval of records. To accomplish this, the application sends a decryption request to the KMS. The KMS responds with just enough key material to decrypt the

specified subsequence, ensuring no other records can be accessed. The primary goal of our work is to make this KMS interaction – regarding both bandwidth and computation – sublinear in relation to the number and size of the records being requested.

Note that by structuring data as a binary tree, any subsequence can be defined by a logarithmic number of descriptors at most, with each descriptor representing a sub-tree. For the rest of this paper, we will focus on the problem of decrypting a full sub-tree of records efficiently (specifically, in constant time). The underlying principle is that an effective algorithm for this sub-problem will allow us to decrypt any arbitrary subsequence with logarithmic complexity. In addition to efficient decryption, we must also preserve the efficient encryption capabilities inherited from the ATSE scheme. This means that while encryptors need to process large chunks of messages efficiently and privately, we also demand *authenticity*. Specifically, the encryptor must be restricted to producing only legitimate ciphertexts. This is achieved by ensuring that the temporary key, derived by the encryptor for a session, is cryptographically bound to the exact set of messages being encrypted. (This process is anchored by a long-term key, which key servers manage in a threshold fashion so that it is never explicitly reconstructed in a single location.)

Integration into Enterprise Data Processing Conceptually, integrating HiSE into a modern data platform involves deploying the "privileged encryptor" as a dedicated, scalable ingestion service. This service would consume real-time data streams (e.g., from Apache Kafka), buffer incoming events into micro-batches, and then execute the HiSE group encryption protocol once per batch before persisting the resulting cipher-trees to a data lake or object store like Amazon S3 [Amab]. By handling encryption at the boundary between the operational and analytical planes, HiSE secures data at rest without impeding the low-latency requirements of real-time systems. Downstream analytics clients, such as Apache Spark [Apa] jobs, would then interact with the KMS to retrieve keys for specific sub-trees of data, enabling secure and performant queries on the historical dataset.

3.2 Our Construction: Technical Insights

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 [AMMR18] construction that uses a (DDH-based) DPRF by Naor, Pinkas and Reingold [NPR99] (NPR-DPRF). 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 a 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 hold 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 [CGMS21a], 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

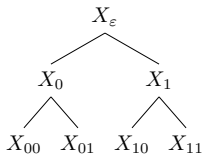

Figure 1: The Merkle-tree is denoted by X. Each node ω is labeled with Merkle-hash X_ω. Leaves correspond to messages.

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 number of records.

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 ATSE to support decryption of subsets.

Our scheme: HiSE Our main idea is to construct a binary-tree such that decryption can be done at *any* node. For our illustrative 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.

Now, our encryption follows an idea similar to BTE [CHK03] and its variant iTire [BMS22]. However, since both of them are public-key schemes, we made several modifications to ensure that our encryption is symmetric *and* satisfies ciphertext authenticity. As a first modification, we now use two field elements as the long-term secret key $sk = (\alpha, \beta)$ – both are secret shared together under a t-out-of-n secret sharing scheme; moreover, only $pp = g_1^\beta$ (not g_1^α) is made public. So, public encryption capability like earlier works [CHK03, BMS22] is removed, and looking ahead, β is used for restricted decryption.

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 $k_i = e(g_1^{r_i}, z)$ — g_1 is generator of $\mathbb{G}_1$ in pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ — which is unique to the m_i, as the randomness r_i is sampled uniquely for each 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

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 $\boldsymbol{X}$, as seen in Fig. 1, on the vector of values $\{(m_1, \rho_1, \mathcal{H}''(g_1^{r_1})), \ldots, (m_4, \rho_4, \mathcal{H}''(g_1^{r_4}))\}$, whose root is X_ε.
- Send $(4, X_\varepsilon)$ to all servers. Say servers $\{2, 3, 4\}$ reply. Each server i replies with $z_i := \mathcal{H}(1, 4, X_\varepsilon)^{\alpha_i}$, where 1 denotes encryptor id and 4 the total number of messages.
- Compute key material $z := \prod_i z_i^{\lambda_i}$ via Lagrange interpolation.
- For $(m_1, \ldots, m_4)$, locally compute the ciphertext components 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 ciphertext $(1, \boldsymbol{X}, \varepsilon, (c_1, c_2, c_3, c_4))$.

Decryption of (c_1, c_2) by decryptor Client-7

- Retrieve $(1, \boldsymbol{X}, 0, (c_1, c_2))$, where 1 is the encryptor id, and 0 denotes the root of Merkle subtree covering leaves 00 and 01.
- Send $(1, 4, X_\varepsilon, X_0)$ to all servers. Say servers $\{4, 5, 6\}$ reply. Each server i computes and returns $\tilde{z}_i = \mathcal{H}(1, 4, X_\varepsilon)^{\alpha_i} \mathcal{H}(X_0)^{\beta_i}$.
- Derive key material $\tilde{z} = \prod_i \tilde{z}_i^{\lambda_i}$ via Lagrange interpolation.
- Decrypt ciphertext $c_1 = (R_1, S_{1,0}, S_{1,00}, E_1)$ locally as follows:
  - 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 $X_{00} = \mathcal{H}^{\mathsf{mt}}(m_1, \rho_1, h_1)$ and $\mathcal{H}''(R_1) = h_1$.
- Perform similar steps as above for decrypting c_2.
- Check if $X_0 = \mathcal{H}^{\mathsf{mt}}(X_{00}, X_{01})$ for Merkle tree consistency.
- Output $[m_1, m_2]$ if all verifications pass; else output $\perp$.

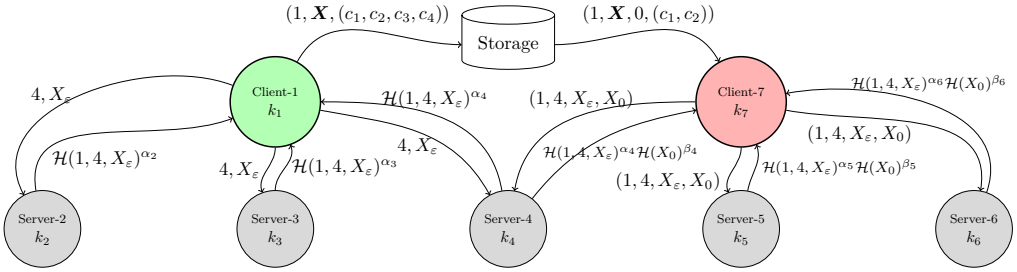

Figure 2: Flow of HiSE protocol for $n = 7$, $N = 4$ and $t = 4$. Encryptor client-1 encrypts messages (m_1, m_2, m_3, m_4) in bulk using $t - 1$ servers $2, 3, 4$, and decryptor client (id-7) decrypts (c_1, c_2) using servers $4, 5, 6$ – both using a single interaction. Using (m_1, m_2, m_3, m_4), client-1 generates a Merkle tree $\boldsymbol{X}$. For encryption, it sends the root X_ε to the servers and receives in return the value z_i as response from server-i. The encryptor client after generating the ciphertexts sends $(1, \boldsymbol{X}, \varepsilon, (c_1, c_2, c_3, c_4))$ to the storage (1 is the encryptor's id, $\boldsymbol{X}$ is the Merkle tree and c_1, c_2, c_3, c_4 are the ciphertexts). The decryptor client-7, in order to decrypt the ciphertexts (c_1, c_2), retrieves $(1, \boldsymbol{X}, 0, (c_1, c_2))$ from the storage and sends $(1, 4, X_\varepsilon, X_0)$ to the servers (1 is the encryptor's id, X_0 is the Merkle hash at node 0 and 4 is the total number of messages (number of leaves in $\boldsymbol{X}$)). The decryptor receives $\tilde{z}_i$ from server-i, and derives per-message keys (in addition to performing validity checks) locally. (We elide some construction-specific details for simplicity.)

$\mathcal{H}(X_{\omega|_j})^{r_i}$.[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:

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

[7] We defer the exact description for computation of E_i until later in this section.

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.

Privacy of HiSE First note that, the value $\mathcal{H}(X_\varepsilon)^\alpha$ is just like NPR-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 to 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 $e(R_i, \mathcal{H}(X_\varepsilon))^\alpha$, which are target group elements. 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 requires using our new assumption: ℓ-masked BDDH along with XDH (see Section 5.2).[9]

Authenticity of HiSE Recall from the TSE literature that, authenticity in this context captures the property of a protocol that ensures that a client must have to interact to generate valid ciphertexts. However, in contrast with DiSE, in our setting (similar to ATSE) a single interaction allows generation of multiple ciphertexts – so more care is needed to handle such granularity (cf. Def. 6).

The broad authenticity argument follows from ATSE. However, there are some crucial differences. First note that, in contrast to ATSE and 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

[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] For example, setting $\ell = 1$, our new assumption states that, given uniform random generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2, g_T \in \mathbb{G}_T$, and elements $g_1^a, g_1^b, g_2^{d_1}, g_2^{c+bd_1}$ for uniform random $a, b, c, d_1 \in \mathbb{Z}_q$, g_T^{ac} is computationally indistinguishable from a uniform random element in $\mathbb{G}_T$.

size of the Merkle-tree) of ciphertexts via one interaction.[10] To remedy this, we also commit to $R_i = g_1^{r_i}$ for each message m_i.[11] 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 $X_\omega = \mathcal{H}^{\mathsf{mt}}(m_i, \rho_i, h_i)$ 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 6) 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 entire 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 an authenticated channel and inclusion of identity (j) of the encryptor in computing the masking key, which changes the scheme as $z = \mathcal{H}(j, N, X_\varepsilon)^\alpha$. To illustrate the workings of our HiSE construction, we provide a sample execution of a simplified example in Figure 2.

3.3 Ramifications

Latency vs. Throughput Trade-off of Batching A primary ramification of this integration model is the necessity to batch messages before encryption, which introduces a fundamental and tunable trade-off between data freshness and decryption efficiency. The size of the batch is a critical, application-level parameter. Using large batches maximizes the amortization benefits of HiSE (in our experiments, we use a batch size of 1K, c.f. Section 8); the logarithmic cost of KMS interactions is spread across thousands of records, making subsequence decryption extremely performant and cost-effective. However, this comes at the cost of increased end-to-end data latency, as records must be buffered for a period before being written to storage, making them unavailable for immediate analysis. Conversely, using smaller micro-batches significantly reduces this latency, but the efficiency gains of amortization are diminished, leading to lower overall throughput for both encryption and decryption operations. This allows system architects to balance competing business requirements.

Storage Overhead vs. Network Efficiency Furthermore, HiSE introduces a clear trade-off between storage costs and network efficiency. The protocol incurs a notable ciphertext expansion of approximately 500 bytes per record, a consequence of embedding the logarithmic-many group elements required for the hierarchical decryption capability. For petabyte-scale data lakes, this represents a non-trivial increase in recurring storage costs. This storage overhead, however, is effectively counterbalanced by a significant, orders-of-magnitude reduction in network bandwidth between the analytics client and the KMS during decryption. While prior schemes required communication linear in the number of records, HiSE's logarithmic interaction drastically cuts network traffic for the read-heavy analytical queries that dominate these application domains [Visc, coi]. This

[10] Ciphertext integrity prevents a corrupt encryptor from producing *any* legitimate ciphertext locally that is 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. See Remark 7.14 of ATSE [CGMS21b] for more discussion.

[11] 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 the proof of message privacy in Section 7.

is a crucial advantage in distributed environments where network latency is a primary performance bottleneck.

4 Roadmap

In Section 5 we provide notations and preliminaries, including several building blocks and communication model. We only provide the preliminaries needed for the base construction. For the strong construction, additional preliminaries are given in Appendix C. In Section 6 we provide formal game-based definitions of our notion HiSE. In Section 7 we describe our main (base) HiSE construction and also provide overview of the analysis. Full analysis of security properties are deferred to Appendix A and Appendix B. Section 8 describes our implementation benchmarking details. We provide concluding remarks plus some future directions in Section 9. The stronger variant of HiSE is defined in Appendix D. In Appendix D.1 we provide the strongly secure HiSE construction. The security properties are proven in subsequent subsections. Appendix E describes the specific instantiations of the zero-knowledge proofs required for our strongly-secure HiSE construction. Finally, in Appendix F we provide a detailed proof of our new (ℓ-masked BDDH) assumption in the generic group model and in Appendix G we provide a reduction from our newly introduced TXDH assumption to well-known XDH.

5 Notation and Preliminaries

5.1 Basic Notations

$\mathbb{N}$ denotes the set of positive integers, and $[n]$ 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 deterministically expressed 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. We say a problem is *computationally hard* to imply that for any PPT adversary, the probability of solving a random instance of that problem is bounded by $\mathsf{negl}(\kappa)$.

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 $a \parallel b$ 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 naturally 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 cyclic group $\mathbb{G}$ of order q, and generated by g, and for a matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times m}$ we write $g^{\boldsymbol{A}}$ to denote the following matrix in $\mathbb{G}^{n \times m}$:

$$\begin{bmatrix} g^{a_{11}} & \dots & g^{a_{1n}} \\ \vdots & \ddots & \vdots \\ g^{a_{m1}} & \dots & g^{a_{mn}} \end{bmatrix}$$

where a_{ij} is the $(i, j)-$th element of matrix $\boldsymbol{A}$.

5.2 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*. Following [ADL+22] we also define a generalization of XDH, which we call Tensor-XDH (similar to Tensor-DDH).

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 $h \in \mathbb{G}_2$. This is a well-known assumption, used in many pairing-based cryptosystems.

Tensor External Diffie-hellman(TXDH). For integers α, β, let $\mathcal{D}_{\alpha,\beta}^{\otimes}$ denote a matrix distribution of $\mathbf{A} = (\boldsymbol{s}, 1) \otimes \boldsymbol{r} \in \mathbb{Z}_q^{(\alpha+1)\times\beta}$ such that $(\boldsymbol{s}, \boldsymbol{r}) \xleftarrow{\$} \mathbb{Z}_q^{\alpha} \times \mathbb{Z}_q^{\beta}$. Given uniform random generators $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and value $g_2^{\mathbf{A}}$ for uniform random $\mathbf{A} \in \mathcal{D}_{\alpha,\beta}^{\otimes}$, it is computationally hard to distinguish between $[g_2^{\mathbf{A}} | g_2^{\mathbf{A}\cdot\boldsymbol{u}}]$ and $[g_2^{\mathbf{A}} | g_2^{\boldsymbol{z}}]$ for uniform random $\boldsymbol{u} \in \mathbb{Z}_q^{\beta}$ and $\boldsymbol{z} \in \mathbb{Z}_q^{\alpha+1}$. Similar to [ADL+22] we show that this reduces to XDH in Appendix G.

ℓ-masked BDDH – A New Assumption. 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 $\mathbb{G}_T$. Here ℓ is a parameter bounded by a polynomial in κ.

Note that, breaking this is easier than breaking BCDH, because given a BCDH solver one can always compute $e(g_1^a, g_2^{c+bd_i}) = g_T^{ac+abd_i}$, and use the BCDH solver with input $g_1^a, g_1^b, g_2^{d_i}$ to compute $g_T^{abd_i}$, and subsequently obtain g_T^{ac} – the other direction is unclear. We show that this assumption holds in the generic group model in Appendix F.

Next, 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 [CGMS21a, AMMR18]. .

5.3 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 $(\mathsf{MTCom}, \mathsf{MTOpen}, \mathsf{MTVer})$ 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** $d := \log(N)$
- **for** $i \in [N]$ **do** :
  - **set** $x_{i,d} := v_i$.
  - **set** $\omega := \mathsf{Bin}(i-1, d)$
  - **set** $X_\omega := x_{i,d}$.
- **set** $L := \frac{N}{2}$
- **for** $j \in [d]$ **do** :
  - **for** $i \in [L]$:
    - **set** $\omega := \mathsf{Bin}(i-1, d-j)$
    - **set** $x_{i,d-j} := \mathcal{H}(x_{2i-1,d-j+1}, x_{2i,d-j+1})$.
    - **set** $X_\omega := x_{i,d-j}$.
  - **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.

[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.

5.4 Shamir's Secret Sharing Scheme

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

Definition 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 $\sum_{j=1}^{\ell} \lambda_j s_{i_j} = s \bmod q$;
- 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 $s_i = (f(i), g(i))$.

5.5 Message and Cipher trees

We introduce a new tree data structure for accessing a tuple of plaintexts/ciphertexts. Suppose that the nodes of the binary-tree are indexed by a binary string with prefix-ordering such that 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 or tags.[13] 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 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 extend this so 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, denoted by $\widehat{\boldsymbol{M}}_\omega$ (or just $\widehat{\boldsymbol{M}}$). 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 $\widehat{\boldsymbol{C}}_\omega$ or $\widehat{\boldsymbol{C}}$. The message-tree

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

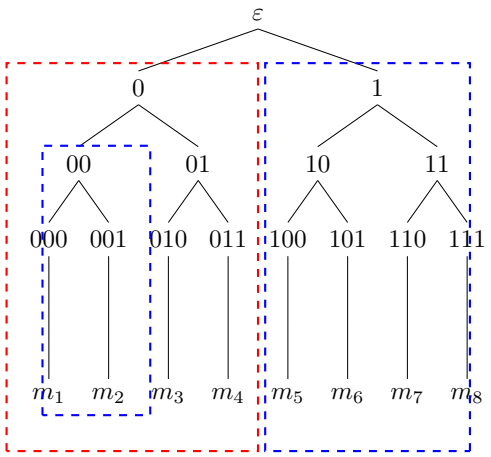

Figure 3: As an example, a plaintext message tuple $\boldsymbol{m}$ of length 8 corresponds to a tree of depth 3. We implicitly assume each node ω has a label X_ω.

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. $\widehat{\boldsymbol{C}} = (\mathsf{aux}, \boldsymbol{X}, \omega, (\omega_1, c_1), \ldots)$. Looking ahead, our schemes 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 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 $\widehat{\boldsymbol{M}}_{00} \prec \widehat{\boldsymbol{M}}_0$.

5.6 Security games & 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 [CGMS21a]. 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.

5.7 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 combines 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.

6 Our Definition: Hierarchical Threshold Symmetric Encryption (HiSE)

Definition 2 (HiSE)**.** A HiSE scheme consists of a tuple $(\mathsf{Setup}, \mathsf{DistGrEnc}, \mathsf{DistGrDec})$ with the following description.

- $\mathsf{Setup}(1^\kappa, 1^n, 1^t) \to (pp, [\![\boldsymbol{sk}]\!]_{[n]})$. A non-interactive algorithm[14] which takes as input the security parameter κ, the total number of parties n and a threshold value $t \leq n$. 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 $\boldsymbol{m}$.

- $\mathsf{DistGrDec}(pp, [\![\boldsymbol{sk}]\!]_{[n]}, [j : \widehat{\boldsymbol{C}}, S]) \to ([j : \boldsymbol{m}/\perp], [S : N, X_\varepsilon, X_\omega])$. 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 key share sk_i. On completion, 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.

*C*onsistency: For any $\kappa, n, t, N \in \mathbb{N}$ such that $t \leq n$, 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 $\omega \in \{0,1\}^{\leq \lceil \log N \rceil}$, sets $S, S' \subset [n]$ ($|S|, |S'| \geq t$), and any $j \in S, j' \in S'$, if all the parties behave honestly, then there exists a negligible function negl for which the following probability is at

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

least $1 - \mathsf{negl}(\kappa)$.

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

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 [AMMR18, CGMS21a] the overall security requirement is captured by three distinct properties: *correctness*, *message-privacy* and *authenticity*, where correctness and authenticity supports stronger versions. We defer defining the **stronger variants** to Appendix D.1.

Definition 3 (Security of HiSE)**.** A HiSE scheme is said to be secure if it satisfies *correctness (Def. 4), message-privacy (Def 5) and authenticity (Def 6).*

Correctness Intuitively, a HiSE scheme is correct whenever a legitimately produced ciphertext (by executing a DistGrEnc protocol), when decrypted (potentially in presence of malicious parties in either execution) 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 4 (Correctness)**.** A 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 below, outputs 1 with probability at most $\mathsf{negl}(\kappa)$.

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

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

Oracle $\mathcal{O}^{\mathsf{hs\text{-}cor\text{-}en}}(j, \boldsymbol{m}, S)$:

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

Oracle $\mathcal{O}^{\mathsf{hs\text{-}cor\text{-}dc}}(j, \widehat{\boldsymbol{C}}, S)$:

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

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

- **require** $j \in S \setminus C$ **and** $j' \in S' \setminus C$ **and** $\omega \in \{0,1\}^{\lceil \log N \rceil}$ **and** $\mathsf{CH} = 0$ **and** $\mathsf{OUT} = 0$; **set** $\mathsf{CH} := 1$; **run** $([j : \mathsf{op}], \ldots) \leftarrow \mathsf{DistGrEnc}(pp, \llbracket sk \rrbracket_S, [j : \boldsymbol{m}, S])$.
- **if** $\mathsf{op} = \perp$ **then set** $\mathsf{OUT} := 0$;

  else set $\widehat{\boldsymbol{C}} := \mathsf{op}$ **and do** :
  - **run** $([j' : \mathsf{op}'], \ldots) \leftarrow \mathsf{DistGrDec}(pp, \llbracket sk \rrbracket_S, [j' : \widehat{\boldsymbol{C}}_\omega, S'])$.
  - **if** $\mathsf{op}' = \boldsymbol{m}^*$ [15] **then set** $\mathsf{OUT} := 0$; **else set** $\mathsf{OUT} := 1$.

[15] m^* contains the set of plaintext messages from m where some or all can be $\perp$

Message privacy Message privacy is naturally defined as an extension of ATSE message-privacy [CGMS21a]. In particular, the adversary is given access to an "honest" decryption oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}dc}}$, in which even the challenge cipher-tree $\widehat{\boldsymbol{C}}^\star$ 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 [AMMR18] and ATSE [CGMS21a]. Finally, just like ATSE, the adversary is provided a special decryption oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}$ 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 (Message privacy)**.** HiSE 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 below.

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

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

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

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

Oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}dc}}(j, \widehat{\boldsymbol{C}}, S)$:

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

Oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch}}(j^\star, \boldsymbol{m}_0, \boldsymbol{m}_1, S^\star)$:

- **require** $j^\star \in S \setminus C$ **and** $|\boldsymbol{m}_0| = |\boldsymbol{m}_1|$ **and** $\mathsf{CH} = 0$; **set** $\mathsf{CH} := 1$.
- **for** $i \in [N]$: **if** $\boldsymbol{m}_0[i] = \boldsymbol{m}_1[i]$ **set** $\mathcal{M} := \mathcal{M} \cup \{\boldsymbol{m}_0[i]\}$.
- **run** $([j^\star : \mathsf{op}], \ldots) \leftarrow \mathsf{DistGrEnc}(pp, \llbracket \boldsymbol{sk} \rrbracket_n, [j^\star : \boldsymbol{m}_b, S^\star])$.
- **if** $\mathsf{op} = \perp$ **return** $\perp$; **else do** :
  - **set** $\widehat{\boldsymbol{C}}^\star := \mathsf{op}$; **return** $\widehat{\boldsymbol{C}}^\star$.

Oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}(j, \widehat{\boldsymbol{C}}, S)$:[16]

[16] In oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}$, the challenger receives query on $\widehat{\boldsymbol{C}} \preceq \widehat{\boldsymbol{C}}^\star$, for which it knows the corresponding message $m_{\widehat{\boldsymbol{C}}}$, because $\mathsf{CH} = 1$.

- **require** $j \in S$ **and** $\mathsf{CH} = 1$ **and** $\widehat{\boldsymbol{C}} \preceq \widehat{\boldsymbol{C}}^{\star}$ **and** $\boldsymbol{m}_{\widehat{\boldsymbol{C}}} \in \mathcal{M}$.
- **run** $([j : \mathsf{op}], \ldots) \leftarrow \mathsf{DistGrDec}(pp, [\![\boldsymbol{sk}]\!]_{[n]}, [j : c, S])$.
- **if** $j \notin C$ **return** op.

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). Similar to DiSE and ATSE, allowing a 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 $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}en}}$ or $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}dc}}$. Also, we crucially rely on the fact that in interactive protocols $\mathsf{DistGrEnc}$ and $\mathsf{DistGrDec}$ all parties in the set S receives X_ε and N (plus the node label X_ω in $\mathsf{DistGrDec}$, which is not used).

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

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

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

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

- **require** $j \in S$.
- **run** $([j : \mathsf{op}(= \widehat{\boldsymbol{C}}/\perp)], [S : X_\varepsilon, N]) \leftarrow \mathsf{DistGrEnc}(pp, [\![\boldsymbol{sk}]\!]_{[n]}, [j : \boldsymbol{m}, S])$.
- **if** $j \notin C$ **then** $\tau := \tau + 1$ **and** $L_{\mathsf{ctxt}}[\tau] := \{\mathsf{op}\}$;

  else if $X_\varepsilon \notin L_{\mathsf{root}}$ **then** :

  - **set** $L_{\mathsf{root}} := L_{\mathsf{root}} \cup X_\varepsilon$; **set** $\mathsf{ct} := \mathsf{ct} + N \cdot |S \setminus C|$.

Oracle $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}dc}}(j, \widehat{\boldsymbol{C}}_\omega, S)$:

- **require**: $j \in S$.
- **run** $([j : \boldsymbol{m}/\perp], [S : X_\varepsilon, X_\omega, N]) \leftarrow \mathsf{DistGrDec}(pp, [\![\boldsymbol{sk}]\!]_{[n]}, [j : \widehat{\boldsymbol{C}}_\omega, S])$.
- **if** $j \in C$ **and** $X_\varepsilon \notin L_{\mathsf{root}}$ **then** :

  - **set** $L_{\mathsf{root}} := L_{\mathsf{root}} \cup X_\varepsilon$; **set** $\mathsf{ct} := \mathsf{ct} + N \cdot |S \setminus C|$.

Oracle $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}t\text{-}dc}}(j, i, \omega, S)$:

– **require**: $j \in S \setminus C$ **and** $i \in [\tau]$; **set** $\widehat{\boldsymbol{C}} := L_{\mathsf{ctxt}}[i]$.

– **run** $([j : \mathsf{op}], \ldots) \leftarrow \mathsf{DistGrDec}(pp, [\![\boldsymbol{sk}]\!]_{[n]}, [j : \widehat{\boldsymbol{C}}_\omega, S])$.

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

– **set** $\ell := \lfloor \mathsf{ct}/g \rfloor$, where $g := t - |C|$.

– **set** $\Big((j_1, S_1, \widehat{\boldsymbol{C}}_{\omega_1}), (j_2, S_2, \widehat{\boldsymbol{C}}_{\omega_2}) \ldots)\Big) := L_{\mathsf{forge}}$.

– **require** $\mathsf{CH} = 0$ **and** $\mathsf{SUCC} = 0$ **and** $j_1, j_2, \ldots \notin C$ **and** $\forall\ i \neq i' :\ \widehat{\boldsymbol{C}}_i \not\preceq \widehat{\boldsymbol{C}}_{i'}$.

– **set** $\mathsf{CH} := 1$ **and** $L_{\mathsf{succ}} := \emptyset$ **and set** $\gamma := |L_{\mathsf{forge}}|$.

– **for** $\forall\ i\ \in [\gamma]$:

  – **run op** $\leftarrow \mathsf{DistGrDec}^{\dagger}$[17]$(pp, [\![\boldsymbol{sk}]\!]_n, [j_i : \widehat{\boldsymbol{C}}_i, S_i])$.

  – $\forall\ k \in [|\mathbf{op}|]$: **if** $\mathsf{op}_k \neq \perp$ **then set** $L_{\mathsf{succ}} := L_{\mathsf{succ}} \cup \{\mathsf{op}_k\}$;

– **if** $|L_{\mathsf{succ}}| > \ell$ **then set** $\mathsf{SUCC} := 1$.

7 Our HiSE construction

We present our HiSE construction below. The construction is based on asymmetric bilinear pairing of Type-3 (no easy isomorphism).

Ingredients and Parameters

Public parameters:

– 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 $(\mathsf{MTCom}, \mathsf{MTOpen}, \mathsf{MTVer})$

Our HiSE Construction

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

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

– $\mathsf{DistGrEnc}(pp, [\![\boldsymbol{sk}]\!]_{[n]}, [j : \boldsymbol{m}, S]) \to ([j : \widehat{\boldsymbol{C}}/\perp], [S : N, X_\varepsilon])$: 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 MT 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.

[17] † is used to denote an honest execution of the protocol

- **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).
  - $c_k := (g_1^{r_k}, \mathcal{H}_2(X_{\omega|_1})^{r_k}, \mathcal{H}_2(X_{\omega|_2})^{r_k}, \ldots, \mathcal{H}_2(X_\omega)^{r_k}, (m_k, \rho_k, \mathcal{H}_4(g_1^{r_k})) \oplus \mathcal{H}_3(e(g_1^{r_k}, z)))$;
  - Set $\omega_k := \omega$.

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

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

  Verify $\mathsf{MTVer}(\boldsymbol{X})$, return $(m_1, \ldots, m_\ell)$ if succeeds, else $\perp$.

We show that our construction is secure (as per Definition 3) from XDH and ℓ-masked BDDH over the bilinear group. Formally we present the following theorem.

Theorem 1. *Our construction is a secure HiSE scheme assuming TXDH and ℓ-masked BDDH over the underlying bilinear pairing in the programmable random oracle model.*

To prove the theorem we show correctness, message-privacy and authenticity below.

7.1 Correctness

A HiSE scheme is said to be correct if whenever an honest party j initiates the encryption protocol $\mathsf{DistGrEnc}$ on an input $(\boldsymbol{m} := m_1, \ldots, m_N, S)$ to generate a ciphertree $\widehat{\boldsymbol{C}} := (j, X, \varepsilon, (\omega_1, c_1), \ldots, \omega_N, c_N))$, then any other honest party j' on running the decryption protocol $\mathsf{DistGrDec}$ for any $\omega \in \{0,1\}^{\leq d}$ on $\widehat{\boldsymbol{C}}_\omega \preceq \widehat{\boldsymbol{C}}$ 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 $\widehat{\boldsymbol{C}}_\omega$. 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 of the Merkle tree commitment scheme.

7.2 Message Privacy (Sketch)

Here we provide a brief sketch of the proof. The full proof is deferred to Appendix A. The message-privacy proof relies on (i) TXDH and (ii) ℓ-masked BDDH over the bilinear group in the random oracle model. Now, from Def. 5, 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 [CGMS21a] we can show this via a number of computationally indistinguishable hybrids starting from HiSE-MsgPriv(0) and eventually reaching HiSE-MsgPriv(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.

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 $\mathcal{H}_2(j, X_\varepsilon)^\alpha$. In this sketch, for simplicity, we just assume that exactly $t-1$ parties are corrupt (similar to ATSE proofs). Furthermore, we assume that the adversary does not make any query to oracles $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}en}}$ or $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}dc}}$. In this simplified case, even XDH suffices. The 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 $\mathcal{H}_2(j, X_\varepsilon)^\alpha = g_2^{ab}$;
- 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. 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 $\widehat{\boldsymbol{C}}^\star$ as follows:
  - For every node ω such that $\widehat{\boldsymbol{M}}_\omega \subset \mathcal{M}$, and the parent ω' of ω such that $\widehat{\boldsymbol{M}}_{\omega'} \subset \mathcal{M}$ 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 $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}$ for $\widehat{\boldsymbol{C}}_\omega$ 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 above.

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).

7.3 Authenticity (Sketch)

Here we provide a brief sketch of the proof. The full proof is deferred to Appendix B. We note that, an adversary $\mathcal{A}$ breaks authenticity if it produces at least one more ciphertext than ℓ in oracle $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}ch}}$. We discuss the following "bad"events

- $\mathbf{E}_1$. This happens when $\mathcal{A}$ wins by sending a (j, N', X_ε) within $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}en}}$ 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 $\widehat{\boldsymbol{C}}$ 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 (the full proof requires TXDH) 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 $\widehat{\boldsymbol{C}}$ corresponding to (j, N, X_ε). If not found, then abort. Otherwise, use $\mathcal{H}_3(e(R, h))$ to decrypt any ciphertext within $\widehat{\boldsymbol{C}}$. 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 $\mathcal{H}_2$.

8 Experimental Evaluation

Here we compare the strong versions[18] of HiSE with ATSE [CGMS21a] on latency, throughput, communication (bandwidth), and ciphertext expansion. We study these performance metrics while varying the number of servers n, threshold t, and the number of records N in a group to manifest the benefits of bulk encryption and decryption. While larger values of n (and t) demonstrate greater improvements for HiSE, we nevertheless restrict our study to reasonable values of n between 2 and 24; we find that a practical enterprise deployment of a threshold KMS system only uses a handful of servers [vaua].

Implementation We implement the HiSE scheme in Rust (roughly 1500 LOC), using the BLS12-381 pairing-based curve implemented in [ac22]. For apples-to-apples comparison, we also implement ATSE [CGMS21a] using the same curve library. The code for both schemes is open-sourced at `https://github.com/rsinha/hise`.

8.1 Experimental Setup

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 deploy both machines on a LAN network, where the round-trip latency was roughly 5 ms. We use the following experimental setup, which, to our understanding, provides a fair characterization and comparison between the HiSE and ATSE schemes.

Latency Measurement To measure latency, we run both the client and server(s) processes on the same machine, with sufficient hardware parallelism, to avoid including the network latency. This produces a more fair reporting of relative speedup which accounts only for the difference in computational complexity. On that note, we use a single threaded implementation for each client or server process, thus capturing the total amount of computation involved in processing the N records. In our reporting, we always provide the end-to-end latency, which includes the following steps: 1) client-side operations to create the request; 2) server-side computation (all servers run concurrently) during which the client is waiting; 2) client-side computation including the verification of responses using the associated NIZK proofs (only the first t correct responses), Lagrange interpolation to compute the group key, and using the group key to encrypt or decrypt all N records in the group.

Throughput Measurement The goal of the throughput measurement to study the linear scaling attributes of our system; i.e., given server-grade machines with sufficient hardware parallelism, what is the throughput for encryption and decryption operations? To that end, 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[19], we let the client launch multiple concurrent threads for processing N records 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 CPU utilization, thus measuring peak throughput. This also shows the linear scaling of our system.

[18] As explained in Remark 7.5 of ATSE [CGMS21b], the malicious security offered by the strong version is suited to our application to enterprise data encryption.

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

t	n	**Latency** (sec)						**Throughput** (records / 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 4: Decryption latency and throughput metrics with 1 K records.

t	n	**Latency** (sec)						**Throughput** (records / 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 5: Encryption latency and throughput metrics with 1 K records.

n	**Latency** (sec)					
	HiSE (best) 1 of 10^6	HiSE (best) 10^2 of 10^6	HiSE (best) 10^4 of 10^6	HiSE (worst) 1 of 10^6	HiSE (worst) 10^2 of 10^6	HiSE (worst) 10^4 of 10^6
6	0.018	0.281	26.689	0.018	0.335	26.743
12	0.028	0.292	27.134	0.028	0.402	27.242
18	0.038	0.304	27.048	0.038	0.466	27.210
24	0.049	0.330	26.882	0.049	0.546	27.098

Figure 6: Best case and worst case decryption latency in HiSE, when decrypting a subset of records (for 1 K records and $t = n/2$).

8.2 Latency and Throughput Measurement

Tables 4 and 5 list the latency and throughput measurements, with varying parameter choices, while encrypting and decrypting a group of records, of varying group sizes. While we report only for 1 K records, we find that the performance is only slightly impacted for larger records, as we only perform very lightweight symmetric key operations (hash and xor) over the record bytes.

The experiments support the expectations laid out via enumeration in Table 1. Our key findings are:

– Though the server interaction is slightly more expensive in HiSE compared to ATSE, by amortizing it over N records, we see an order of magnitude improvement in latency

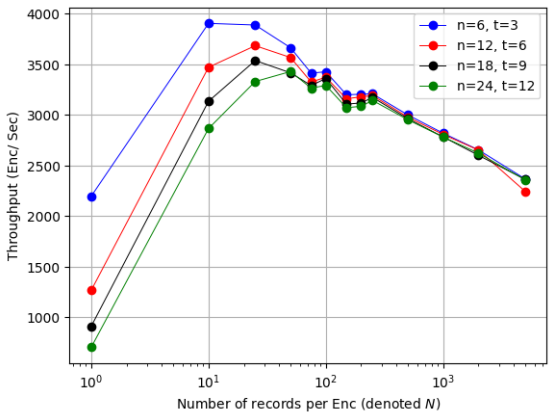

Figure 7: Effect of N on Encryption Throughput

and throughput of the decryption. We observe between 10-65× gain in throughput and 12-70× in latency in different settings.
– The downside is that encryption has roughly 4.5× higher latency and lower throughput, due to the log-many more group operations per record.
– We find an interesting tradeoff in HiSE's encryption. While larger values of N should further amortize the one-time server interaction, the $\log(N)$ overhead in group operations makes encryption more inefficient beyond a certain value of N – this is not the case in decryption, which has constant number of operations for each record. In our findings, as shown in Figure 7, we find a sweet-spot roughly around $N = 100$, where encrypting larger groups than 100 records (each 1 K) results in lower encryption throughput.

We also study the effects when decrypting a subset of messages that were encrypted together as a group. Recall that, in the worst case, the decryption procedure incurs log-many (in the number of records being decrypted) interactions with the server(s) – that said, this work is amortized across all records being decrypted. As a result, the running time for decryption depends on the range of indices being decrypted – i.e., how many subtrees cover the range – with the best case being a single subtree and worst case being a logarithmic number. These scenarios are captured in the measurements reported in Table 6. Note that the difference between best and worst case becomes negligible when the fraction of the group's messages being decrypted is larger. For instance, when decrypting a range of 10^2 out of 10^6 records, the worst case latency is 19% larger than the best case (0.335 vs. 0.281), but 0.2% larger when decrypting 10^4 records.

8.3 Bandwidth Costs

HiSE facilitates reduction in bandwidth between the client and server(s), incurring constant communication overhead in N in a group. In contrast, ATSE transmits linear number of elements. For each interaction (i.e., for each invocation of DistGrEnc), ignoring the underlying TLS overheads, the HiSE client receives a total of $192*n$ bytes, comprising two 48-byte group element and three 32-byte scalar values from each server[20]. In contrast, an ATSE client receives $480*n*N$ bytes. For $N = 10^4$, that constitutes 10^4× reduction in bandwidth – a significant saving in cloud deployments.

[20]Recall that the server's response contains one group element for the derived key, and the associated NIZK proof has one (blinded) group element and two (blinded) scalars.

8.4 Storage Costs

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

9 Conclusion

Building on amortised threshold symmetric encryption (ATSE [CGMS21a]), 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 4.5× overhead plus a $\log(N)$ blow-up in ciphertext size, we observe between 10-65× improvement in latency and throughput during decryption. This improvement is beneficial in enterprise workloads, which are "read" / decryption heavy. Nevertheless, we leave open the possibility for an even more efficient scheme, in particular one that supports constant size ciphertext. Furthermore, we leave exploring other access structures beyond binary-trees for this setting.

References

[ac22] arkworks contributors. `arkworks` zksnark ecosystem. https://arkworks.rs, 2022.

[ADL+22] 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. doi:10.1007/978-3-031-22912-1_2.

[Amaa] Amazon kinesis data analytics. https://docs.aws.amazon.com/kinesisanalytics/latest/dev/windowed-sql.html.

[Amab] Amazon s3: Object storage built to retrieve any amount of data from anywhere. https://aws.amazon.com/s3/.

[Amac] Amazon Timestream. https://aws.amazon.com/timestream/.

[AMMM18] 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*, pages 2042–2059. ACM Press, October 2018. doi:10.1145/3243734.3243839.

[AMMR18] 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*, pages 1993–2010. ACM Press, October 2018. doi:10.1145/3243734.3243774.

[Apa] Apache spark: Unified engine for large-scale data analytics. https://spark.apache.org.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456, May 2005. doi:10.1007/11426639_26.

[BEL+20] 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*, pages 409–424. ACM Press, November 2020. doi:10.1145/3372297.3417266.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229, August 2001. doi:10.1007/3-540-44647-8_13.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532, December 2001. doi:10.1007/3-540-45682-1_30.

[BMS22] 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*, pages 235–248. ACM Press, November 2022. doi:10.1145/3548606.3560704.

[Boy08] 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.

[BP23] Luís Brandão and Rene Peralta. NIST First Call for Multi-Party Threshold Schemes. 2023. https://csrc.nist.gov/publications/detail/nistir/8214c/draft.

[BSS05] 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, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 644–661, December 2005. doi:10.1007/11593447_35.

[BSW07] 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.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300, December 2013. doi:10.1007/978-3-642-42045-0_15.

[CG99] R. Canetti and S. Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. *J. Stern, editor, EUROCRYPT'99*, 1592 of LNCS, pages 90–106,Springer, Heidelberg, May 1999.

[CGMS21a] Mihai Christodorescu, Sivanarayana Gaddam, Pratyay Mukherjee, and Rohit Sinha. Amortized threshold symmetric-key encryption. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2758–2779. ACM Press, November 2021. doi:10.1145/3460120.3485256.

[CGMS21b] 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.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 255–271, May 2003. doi:10.1007/3-540-39200-9_16.

[CLH10] Shu-Hui Chang, Chuan-Ming Li, and Tzonelih Hwang. Identity-based hierarchical designated decryption. *J. Inf. Sci. Eng.*, 26(4):1243–1259, 2010. URL: http://www.iis.sinica.edu.tw/page/jise/2010/201007_06.html.

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

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315, August 1990. doi:10.1007/0-387-34805-0_28.

[DGRW18] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 155–186, August 2018. doi:10.1007/978-3-319-96884-1_6.

[DMV22] 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.

[DP08] C. Delerablée and D. Pointcheval. Dynamic threshold public-key encryption. *CRYPTO 2008, volume 5157 of LNCS, pages 317–334. Springer, Heidelberg*, Aug 2008.

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

[FKMV12] 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, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79, December 2012. doi:10.1007/978-3-642-34931-7_5.

[Fra90] Yair Frankel. A practical protocol for large group oriented networks. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89*, pages 56–61, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg. URL: https://rdcu.be/eGhBJ.

[GPSW06] 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*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309. doi:10.1145/1180405.1180418.

[GS02] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 548–566, December 2002. doi:10.1007/3-540-36178-2_34.

[JKR19] 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*, pages 379–393. ACM Press, November 2019. doi:10.1145/3319535.3363196.

[KHF+20] 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. doi:10.1145/3399742.

[KKD+23] Abdelouahab Khelifati, Mourad Khayati, Anton Dignös, Djellel Difallah, and Philippe Cudré-Mauroux. Tsm-bench: Benchmarking time series database systems for monitoring applications. *Proc. VLDB Endow.*, 16(11):3363–3376, July 2023. doi:10.14778/3611479.3611532.

[KOR+17] 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 2017*, volume 10355 of *LNCS*, pages 229–249, July 2017. doi:10.1007/978-3-319-61204-1_12.

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

[LSG+20] 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. doi:10.1145/3357033.

[med] Time window queries and aggregates. https://tinyurl.com/mrypcyze.

[MSJ02] Philip D. MacKenzie, Thomas Shrimpton, and Markus Jakobsson. Threshold password-authenticated key exchange. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 385–400, August 2002. doi:10.1007/3-540-45708-9_25.

[Muk20] Pratyay Mukherjee. Adaptively secure threshold symmetric-key encryption. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 465–487, December 2020. doi:10.1007/978-3-030-65277-7_21.

[NPR99] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 327–346, May 1999. doi:10.1007/3-540-48910-X_23.

[ris] Time series data is everywhere. https://kx.com/time-series-database/.

[SDFY94] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. *26th ACM STOC, pages 522–533,ACM Press*, May 1994.

[SG98] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *EUROCRYPT'98, volume 1403 of LNCS, pages 1–16. Springer, Heidelberg*, May / June 1998.

[Sho97] 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.

[SOL+23] Chunhui Shen, Qianyu Ouyang, Feibo Li, Zhipeng Liu, Longcheng Zhu, Yujie Zou, Qing Su, Tianhuan Yu, Yi Yi, Jianhong Hu, et al. Lindorm tsdb: A cloud-native time-series database for large-scale monitoring systems. *Proceedings of the VLDB Endowment*, 16(12):3715–3727, 2023.

[spl] Time series databases (tsdbs) explained. https://www.splunk.com/en_us/blog/learn/time-series-databases.html.

[SW05] 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.

[vaua] Vault Architecture. https://www.vaultproject.io/docs/internals/architecture.html.

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

[VGTM23] Juliane Verwiebe, Philipp M. Grulich, Jonas Traub, and Volker Markl. Survey of window types for aggregation in stream processing systems. *The VLDB Journal*, 32(5):985–1011, February 2023. doi:10.1007/s00778-022-00778-6.

[Visa] Beyond transactions: The social value of payments technology innovation. https://corporate.visa.com/content/dam/VCOM/regional/na/us/sites/documents/veei-beyond-transactions.pdf.

[Visb] Fact Sheet - Visa. https://usa.visa.com/dam/VCOM/download/corporate/media/visanet-technology/aboutvisafactsheet.pdf. [Online; posted June-2018].

[Visc] Message level encryption. https://developer.visa.com/pages/encryption_guide.

Appendix

A Message Privacy of HiSE (in detail)

We start by defining a number of hybrids for any PPT adversary $\mathcal{A}$.

Hyb_0**:** This is the original HiSE-MsgPriv(0) game.

Hyb_1**:** This hybrid is identical to Hyb_0, except for how the challenge ciphertree is generated. When the adversary submits its challenge query $(j^*, \boldsymbol{m}_0, \boldsymbol{m}_1, S^*)$, the challenger proceeds as follows:

- It computes the Merkle root X^*_ε for the message vector $\boldsymbol{m}_0$.
- Instead of computing the key $z^* = \mathcal{H}_2(j^*, N^*, X^*_\varepsilon)^\alpha$, it samples a uniformly random element $z^* \leftarrow_s \mathbb{G}_2$.
- It then constructs the challenge ciphertree $\widehat{\boldsymbol{C}}^*$ using this random z^*.
- All other oracle queries are handled exactly as in Hyb_0.

The indistinguishability of Hyb_0 and Hyb_1 is based on the (Q,t)-*TXDH assumption*, where Q is the total number of encryption oracle queries made by the adversary. The value z^* is effectively evaluation of an instance of NPR-DPRF on the input $(j^*, N^*, X^*_\varepsilon)$. The TXDH assumption guarantees that this output is computationally indistinguishable from a random group element, even given the adversary's view of other DPRF evaluations from its queries to $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}en}}$. A reduction is provided in Lemma 1.

Hyb_2**:** This hybrid modifies the challenge decryption oracle, $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}$. When queried on a valid sub-tree $\widehat{\boldsymbol{C}}_\omega \leq \widehat{\boldsymbol{C}}^*$ (where all messages in the sub-tree are in $\mathcal{M}$), the challenger proceeds as follows:

- Instead of computing the decryption key $\tilde{z} = z^* \cdot \mathcal{H}_2(X_\omega)^\beta$ (where z^* is random from Hyb_1), it samples a fresh, uniformly random element $\tilde{z} \leftarrow_s \mathbb{G}_2$.
- It uses this new random $\tilde{z}$ to perform the final decryption steps for the client.
- All other oracle queries are handled as in Hyb_1.

The indistinguishability of Hyb_1 and Hyb_2 is based on the ℓ-*masked BDDH assumption*. In Hyb_1, the value z^* is already uniformly random. The adversary is given $pp = g_1^\beta$ and, through a query to $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}$, observes the interaction that leads to the reconstruction of $\tilde{z} = z^* \cdot \mathcal{H}_2(X_\omega)^\beta$. The ℓ-masked BDDH assumption implies that the resulting masking key for any ciphertext c_k outside the sub-tree $\widehat{\boldsymbol{C}}_\omega$, which is of the form $e(R_k, z^*)$, remains pseudorandom. The term $\mathcal{H}_2(X_\omega)^\beta$ acts as a random mask on z^*, and the adversary cannot leverage the decryption of $\widehat{\boldsymbol{C}}_\omega$ to "unmask" z^* and gain information about other parts of the ciphertext. A formal reduction is provided in Lemma 2.

Hyb_3**:** This hybrid is identical to Hyb_2, except that in the challenge oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch}}$, the challenger now encrypts the message vector $\boldsymbol{m}_1$ instead of $\boldsymbol{m}_0$. In Hyb_2, the challenge ciphertree $\widehat{\boldsymbol{C}}^*$ is generated using a key z^* that is uniformly random and independent of all other values in the game. The encryption of each encryption plaintext value $\tilde{\mu}_k$ is performed via an XOR with a hash of a value derived from this random key, i.e., $E_k = \tilde{\mu}_k \oplus \mathcal{H}_3(e(g_1^{r_k}, z^*))$. Since z^* is uniform (and the pairing is non-degenerate), $e(g_1^{r_k}, z^*)$ is uniform in $\mathbb{G}_T$. As $\mathcal{H}_3$ is modeled as a random oracle, the output is pseudorandom. This means the encryption is effectively working as a one-time pad. The adversary's view is therefore information-theoretically independent of the choice between $\boldsymbol{m}_0$ and $\boldsymbol{m}_1$.

Finally, we can reverse the sequence of modifications to go from hybrid Hyb_4 to Hyb_5, which is essentially the same as HiSE-MsgPriv(1) game.

Hyb_4**:** This hybrid is identical to Hyb_3, but we revert the change made in Hyb_2. The challenge decryption oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}$ now uses the correct computation $\tilde{z} = z^* \cdot \mathcal{H}_2(X_\omega)^\beta$, where z^* is still random. By the same argument as for $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$, this hybrid is indistinguishable from Hyb_3, based on the ℓ-masked BDDH assumption.

Hyb_5**:** This hybrid is identical to Hyb_4, but we revert the change made in Hyb_1. The challenge oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch}}$ now uses the correctly computed key $z^* = \mathcal{H}_2(j^*, N^*, X_\varepsilon^*)^\alpha$ to encrypt $\boldsymbol{m}_1$. By the same argument as for $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$, this hybrid is indistinguishable from Hyb_4, based on the TXDH assumption. Hybrid Hyb_5 is identical to the HiSE-MsgPriv(1) experiment.

Now to complete the proof we show the following two lemmas to formally show the reductions between $\mathsf{Hyb}_0, \mathsf{Hyb}_1$ (and similarly, $\mathsf{Hyb}_4, \mathsf{Hyb}_5$) and $\mathsf{Hyb}_1, \mathsf{Hyb}_2$ (and similarly, $\mathsf{Hyb}_3, \mathsf{Hyb}_4$) successively.

Lemma 1 ($\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$)**.** *If there exists a PPT adversary $\mathcal{A}$ that can distinguish between Hyb_0 and Hyb_1 with non-negligible advantage ε, we can construct a PPT algorithm $\mathcal{B}$ that breaks the (Q, τ)-TXDH assumption with advantage ε/Q, where Q is an upper bound on $\mathcal{A}$'s queries.*

Proof. $\mathcal{B}$ receives a TXDH challenge instance $(g_2^A, g_2^{\boldsymbol{b}})$, where $A \in \mathbb{Z}_q^{(\tau+1)\times Q}$ (with $\tau = t - |C| - 1$), and $\boldsymbol{b}$ is either $A \cdot \boldsymbol{u}$ (real) or a random vector (random). $\mathcal{B}$ guesses that the k-th encryption query will be the challenge. It simulates the hybrids for $\mathcal{A}$. For non-challenge queries $i \neq k$, $\mathcal{B}$ can compute the correct key shares using its knowledge of A and setting $u_i = 1$. For the k-th (challenge) query, it uses the vector $\boldsymbol{b}$ from its TXDH instance to construct the server responses. If $\boldsymbol{b}$ is from the real distribution, $\mathcal{B}$ perfectly simulates Hyb_0. If $\boldsymbol{b}$ is from the random distribution, $\mathcal{B}$ perfectly simulates Hyb_1. $\mathcal{A}$'s ability to distinguish the hybrids allows $\mathcal{B}$ to distinguish the TXDH distributions, thus breaking the assumption. □

Lemma 2 ($\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$)**.** *If there exists a PPT adversary $\mathcal{A}$ that can distinguish between Hyb_1 and Hyb_2 with non-negligible probability, we can construct a PPT algorithm $\mathcal{B}$ that breaks the ℓ-masked BDDH assumption with non-negligible probability.*

Proof. $\mathcal{B}$ receives an ℓ-masked BDDH challenge $(g_1^a, g_1^b, \{g_2^{d_i}, g_2^{c+bd_i}\}_{i\in[l]}, h \in \mathbb{G}_T)$. $\mathcal{B}$'s goal is to determine if $h = e(g_1^a, g_2^c)$ or if h is random. $\mathcal{B}$ simulates the HiSE environment for $\mathcal{A}$ by implicitly setting the master secrets $\alpha := a$ and $\beta := b$. It sets the public parameter $pp := g_1^b$. It programs the random oracle $\mathcal{H}_2$ such that for the challenge root X_ε^*, $\mathcal{H}_2(j^*, N^*, X_\varepsilon^*) := g_2^c$, and for any node ω_i that can be queried in $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}$, it sets $\mathcal{H}_2(X_{\omega_i}) := g_2^{d_i}$. When simulating the challenge encryption, for any message not in $\mathcal{M}$, the masking key involves $e(g_1^{r_k}, (g_2^c)^a) = e(g_1^a, g_2^c)^{r_k}$. $\mathcal{B}$ can compute this using its challenge value h. When simulating a challenge decryption query for node ω_i, it needs to provide shares for $\tilde{z} = (g_2^c)^a (g_2^{d_i})^b = g_2^{ac+bd_i}$. It can do this using the challenge term $g_2^{c+bd_i}$. If

$h = e(g_1^a, g_2^c)$, $\mathcal{B}$ has perfectly simulated Hyb_1 (where z^* is random and $\tilde{z}$ is computed from it). If h is random, $\mathcal{B}$ has perfectly simulated Hyb_2 (where both z^* and $\tilde{z}$ are effectively random). Thus, $\mathcal{A}$'s success in distinguishing the hybrids allows $\mathcal{B}$ to break the ℓ-masked BDDH assumption. □

This concludes the proof.

B Authenticity of HiSE (in detail)

This section provides the detailed proof of authenticity for the base HiSE construction. It relies on the binding property of the Merkle tree commitment, the collision resistance of the hash functions, and the TXDH assumption. Let us start by defining a sequence of hybrids for any PPT adversary $\mathcal{A}$:

Hyb_0: This is the original HiSE-Auth game.

Hyb_1: This hybrid is identical to Hyb_0, except that the challenger performs an additional check within the $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}ch}}$ oracle. When verifying a forged ciphertree $\widehat{\boldsymbol{C}}_i = (j, X, \omega_i, \dots)$, the challenger checks if the number of leaves N in the Merkle tree X is consistent with the value N' used by the adversary during any key derivation interactions for the root X_ε. If there is a mismatch for any ciphertree, the game aborts and the adversary loses.

One can see that the hybrids Hyb_0 and Hyb_1 are computationally indistinguishable. If an adversary could succeed in Hyb_0 but fails in Hyb_1, it means it created a valid forgery where the number of messages N in the Merkle tree X does not match the number N' it declared to the servers during an interaction for the root X_ε. For the forgery to be valid, the decryption must succeed. However, an honest decryptor in the $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}ch}}$ oracle will parse N from the provided tree X and use it to compute the key $z = \mathcal{H}_2(j, N, X_\varepsilon)^\alpha$. If $N \neq N'$, this key will not match the key $z' = \mathcal{H}_2(j, N', X_\varepsilon)^\alpha$ that the adversary derived. In the random oracle model, $\mathcal{H}_2(j, N, X_\varepsilon)$ and $\mathcal{H}_2(j, N', X_\varepsilon)$ are independent and uniform random, so decryption will fail except with negligible probability. Therefore, the probability that $\mathcal{A}$ comes up with a successful forgery of this type is negligible.

Hyb_2: This hybrid is identical to Hyb_1, except with an additional check in the $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}ch}}$ oracle. When verifying a forged ciphertext $c_k = (R, S_{\dots}, E)$ within a ciphertree, the challenger checks if the value R is consistent with the value committed to in the corresponding leaf of the Merkle tree X. Recall that a leaf is computed as $y_k = \mathcal{H}_1(m_k, \rho_k, \mathcal{H}_4(R))$. Therefore, if the adversary produces a forgery where the R in the ciphertext body differs from the one implicitly committed to in the Merkle tree leaf, the game aborts and the adversary loses.

We argue that Hyb_1 and Hyb_2 are computationally indistinguishable. If an adversary succeeds in Hyb_1 but fails in Hyb_2, it must have produced a valid forgery where, for some message m_k, the ciphertext component R_k is different from an R'_k such that the leaf value is $y_k = \mathcal{H}_1(m_k, \rho_k, \mathcal{H}_4(R'_k))$. For the decryption to succeed, the check $\mathcal{H}_4(R_k) = h_k$ must pass, where h_k is recovered from the decrypted values. The Merkle tree verification also requires that $X_{\omega_k} = \mathcal{H}_1(m_k, \rho_k, h_k)$. This implies that $\mathcal{H}_1(m_k, \rho_k, \mathcal{H}_4(R_k)) = \mathcal{H}_1(m_k, \rho_k, \mathcal{H}_4(R'_k))$, which requires a collision in the hash function $\mathcal{H}_4$ – the probability of this happening is negligible as $\mathcal{H}_4$ is a random oracle.

In Hyb_2, any successful forgery must involve the adversary computing a valid key $z^* = \mathcal{H}_2(j^*, N^*, X^*_\varepsilon)^\alpha$ for a fresh tuple $(j^*, N^*, X^*_\varepsilon)$ without making a "sufficient" number of queries to honest servers. A tuple is considered "fresh" if the adversary has not obtained t or more key shares for it from in total via the $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}en}}$ or $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}dc}}$ oracles. The ability to compute z^* with fewer than t total shares essentially constitutes a break of the

pseudorandomness of the underlying NPR-DPRF. We now show that this is computationally hard under the TXDH assumption.

Lemma 3. *The probability of an adversary winning in* Hyb_2 *is negligible, assuming the TXDH assumption holds.*

Proof. We construct a reduction $\mathcal{B}$ that uses an authenticity adversary $\mathcal{A}$ to break the TXDH assumption as follows:

1. $\mathcal{B}$ receives a TXDH challenge instance $(g_2^{\boldsymbol{A}}, g_2^{\boldsymbol{b}})$, where $\boldsymbol{A} \in \mathbb{Z}_q^{(\tau+1)\times Q}$ (with $\tau = t - |C| - 1$), and $\boldsymbol{b}$ is either $A \cdot \boldsymbol{u}$ for a random $\boldsymbol{u}$ or a random vector.
2. $\mathcal{B}$ simulates the Hyb_2 for $\mathcal{A}$ as follows:
   - $\mathcal{B}$ generates the master secret β and its shares $\{\beta_i\}_{i\in[n]}$ honestly. It provides $\{\beta_i\}_{i\in C}$ to $\mathcal{A}$. The master secret α is kept implicit in the exponent of $g_2^{\boldsymbol{A}}$.
   - $\mathcal{B}$ guesses that the adversary's forgery will be for the k-th unique tuple (j, N, X_ε) queried to the random oracle $\mathcal{H}_2$.
   - $\mathcal{B}$ programs the random oracle $\mathcal{H}_2$. For the guessed forgery target $(j^*, N^*, X_\varepsilon^*)$, it does not define an output yet. For all other $Q-1$ queries $(j_i, N_i, X_{\varepsilon,i})$, it maps them to the columns of the matrix A from its challenge. The rows of A are mapped to the secret shares of α for the $\tau+1$ honest parties that $\mathcal{B}$ must simulate.
   - When $\mathcal{A}$ queries an honest server $i \notin C$ for a tuple $(j_q, N_q, X_{\varepsilon,q})$ that is not the forgery target: $\mathcal{B}$ can compute the required share $\mathcal{H}_2(j_q, N_q, X_{\varepsilon,q})^{\alpha_i}$. In the exponent, this corresponds to an entry in the matrix $\boldsymbol{A}$, which $\mathcal{B}$ has access to via $g_2^{\boldsymbol{A}}$.
   - When $\mathcal{A}$ submits its forgery L_{forge}, $\mathcal{B}$ checks if it contains a ciphertree for the target tuple $(j^*, N^*, X_\varepsilon^*)$. If not, $\mathcal{B}$ aborts.
   - If the guess was correct, $\mathcal{A}$ must have computed $z^* = \mathcal{H}_2(j^*, N^*, X_\varepsilon^*)^\alpha$ to create a valid ciphertree $\widehat{\boldsymbol{C}}^*$. To verify this ciphertree, one needs the key z^*. $\mathcal{B}$ uses its challenge vector $\boldsymbol{b}$ to construct the key. It sets $z^* := g_2^{b_0}$ (the first element of $\boldsymbol{b}$). It then attempts to decrypt a ciphertext from $\widehat{\boldsymbol{C}}^*$ using this key.
3. Finally, if the decryption succeeds, it implies that the key z^* used by $\mathcal{A}$ was indeed $g_2^{b_0}$. If $\boldsymbol{b}$ was from the "real" distribution $(A \cdot \boldsymbol{u})$, then this corresponds to a correctly computed output. If $\boldsymbol{b}$ was from the "random" distribution, then the probability of a random group element being the correct key is negligible. Therefore, if $\mathcal{A}$'s forgery succeeds, $\mathcal{B}$ can conclude with high probability that it was given a "real" TXDH instance. Otherwise, it was likely an "random" instance. This implies whenever $\mathcal{A}$ succeeds with non-negligible probability, $\mathcal{B}$ breaks TXDH with non-negligible probability as well.

□

This concludes the proof of authenticity of the base HiSE construction.

C Preliminaries for the Strong HiSE Construction

C.1 Commitment

Definition 7. A (non-interactive) secure commitment scheme Σ consists of two PPT algorithms $(\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ which satisfy hiding and binding properties:

- $\mathsf{Setup}_{\mathsf{com}}(1^\kappa) \to pp_{\mathsf{com}}$: 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 8 (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 $\mathsf{SimSetup}$ and $\mathsf{SimOpen}$:

- $\mathsf{SimSetup}(1^\kappa) \to (pp_{\mathsf{com}}, \tau_{\mathsf{com}})$: 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 1. Clearly, a trapdoor commitment can be binding against PPT adversaries only.

C.1.1 Concrete instantiations.

Practical commitment schemes can be instantiated under various settings. Here we specifically 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, $\mathsf{SimSetup}$ 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, $\mathsf{SimOpen}$ 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.2 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. [FKMV12].

Definition 9 (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.

- **Knowledge Soundness**: 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 $(\mathsf{Prove}, \mathsf{Verify})$ 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 $(\mathsf{Prove}, \mathsf{Verify})$ 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. [FKMV12] for more details. (They actually show that these properties hold even when adversary can ask for proofs of false instances.) Here we define and construct a stronger version of HiSE, which we call Strong HiSE.

D Strongly secure HiSE

Definition 10 (Strong Security of HiSE)**.** A HiSE scheme is said to be strongly-secure if it satisfies *strong correctness, message-privacy (Def 5) and strong authenticity.* Strong-correctness is the same as correctness (Def. 4), except that the challenge oracle is replaced by $\mathcal{O}^{\mathsf{hs\text{-}str\text{-}cor\text{-}ch}}$. Strong authenticity is the same as authenticity (Def. 6), except that the challenge oracle is replaced by $\mathcal{O}^{\mathsf{hs\text{-}st\text{-}au\text{-}ch}}$. We describe these two oracles below. The difference with the previous oracles are highlighted in blue.

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

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

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

- **set** $\mathsf{CH} := 1$.
- **run** $([j : \mathsf{op}], \ldots) \leftarrow \mathsf{DistGrEnc}(pp, [\![sk]\!]_S, [j : \boldsymbol{m}, S])$.
- **if** $\mathsf{op} = \perp$ **then set** $\mathsf{OUT} := 0$;

  else set $\widehat{\boldsymbol{C}} := \mathsf{op}$ **and do** :

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

For strong authenticity, instead of an honest $\mathsf{DistGrDec}^{\dagger}$, $\mathsf{DistGrDec}$ 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).

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

- **set** $\ell := \lfloor \mathsf{ct}/g \rfloor$, where $g := t - |C|$.
- **set** $\Big((j_1, S_1, \widehat{\boldsymbol{C}}_{\omega_1}), (j_2, S_2, \widehat{\boldsymbol{C}}_{\omega_2}) \ldots)\Big) := L_{\mathsf{forge}}$;

  require $\mathsf{CH} = 0$ **and** $\mathsf{SUCC} = 0$ **and** $j_1, j_2, \ldots \notin C$ **and** $\forall\ i \neq i' :\ \widehat{\boldsymbol{C}}_i \not\preceq \widehat{\boldsymbol{C}}_{i'}$.
- **set** $\mathsf{CH} := 1$ **and** $L_{\mathsf{succ}} := \emptyset$ **and set** $\gamma := |L_{\mathsf{forge}}|$.
- **for** $\forall\ i\ \in [\gamma]$:

- **run op** $\leftarrow \mathsf{DistGrDec}(pp, [\![sk]\!]_n, [j_i : \widehat{C}_i, S_i])$.
- $\forall\ k \in [|\mathbf{op}|]$: **if** $\mathsf{op}_k \neq \perp$ **then set** $L_{\mathsf{succ}} := L_{\mathsf{succ}} \cup \{\mathsf{op}_k\}$;

– **if** $|L_{\mathsf{succ}}| > \ell$ **then set** $\mathsf{SUCC} := 1$.

D.1 Our Strongly Secure HiSE Construction

Our strongly secure HiSE construction is achieved by extending the basic HiSE construction by adding a trapdoor commitment and a non-interactive zero-knowledge (NIZK) 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 protocol. In this section we present the strongly secure HiSE construction assuming the NIZKs for described relations. In the next section we provide the concrete NIZK instantiations. In our construction below we highlight the changes from our base HiSE construction (Section 7)) using blue.

Ingredients and Parameters

Public parameters:

- An efficiently computable Type-3 bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, 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 $\mathcal{H}_1 : \{0,1\}^* \rightarrow \{0,1\}^\kappa$; $\mathcal{H}_2 : \{0,1\}^* \rightarrow \mathbb{G}_2$; $\mathcal{H}_3 : \mathbb{G}_T \rightarrow \{0,1\}^{\mathsf{poly}(\kappa)}$; $\mathcal{H}_4 : \mathbb{G}_1 \rightarrow \{0,1\}^{\mathsf{poly}(\kappa)}$; $\mathcal{H}' : \{0,1\}^* \rightarrow \{0,1\}^{\mathsf{poly}(\kappa)}$.

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. 9). Consists of algorithms $(\mathsf{Prove}_{\mathfrak{R}}^{\mathcal{H}'}, \mathsf{Verify}_{\mathfrak{R}}^{\mathcal{H}'})$.

Our Strongly secure HiSE Construction

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

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 $pp := (g_1^\beta, \gamma_{\alpha,1}, \ldots, \gamma_{\alpha,n}, \gamma_{\beta,1}, \ldots, \gamma_{\beta,n}, pp_{\mathsf{com}})$.

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

- **Pre-computation (local).** Let $(m_1 \dots, m_N) := \boldsymbol{m} \in \{0,1\}^*$ 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 $\boldsymbol{X} := \mathsf{MTCom}(y_1, \dots, y_N)$ 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: $c_k := (g_1^{r_k}, \mathcal{H}_2(X_{\omega|_1})^{r_k}, \mathcal{H}_2(X_{\omega|_2})^{r_k}, \dots, \mathcal{H}_2(X_\omega)^{r_k}), (m_k, \rho_k, \mathcal{H}_4(g_1^{r_k})) \oplus \mathcal{H}_3(e(g_1^{r_k}, z)))$
  - Set $\omega_k := \omega$.

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

- $\mathsf{DistGrDec}(pp, [\![\boldsymbol{sk}]\!]_S, [j : \widehat{\boldsymbol{C}}, S]) \to ([j : \boldsymbol{m}/\perp], [S : N, X_\varepsilon, X_\omega])$: Parse $\widehat{\boldsymbol{C}}$ as $(j, \boldsymbol{X}, \omega, (\omega_1, c_1), \dots, (\omega_\ell, c_\ell))$ 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 $(R, S_{\omega'|_1}, S_{\omega'|_2}, \dots, E)$.
    - 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 $(m_1, \dots, m_\ell)$.

Theorem 2. *Our construction is a strongly secure HiSE scheme assuming XDH and ℓ-masked BDDH over the underlying bilinear pairing in the programmable random oracle model plus the security of trapdoor commitment and non-interactive argument of knowledge.*

To prove the theorem we show strong correctness, message-privacy and strong authenticity below.

D.2 Strong Correctness

The *strong-correctness* property says that given a ciphertext $\widehat{\boldsymbol{C}}$ of a message $\boldsymbol{m}$ generated using the distributed encryption protocol $\mathsf{DistGrEnc}(\cdot,\cdot,\cdot)$ by an honest encryptor, under honest run of the distributed decryption protocol $\mathsf{DistGrDec}(\cdot,\cdot,\cdot)$, must output $\boldsymbol{m}$. This is to capture the requirement that encryption in the presence of malicious servers should not produce a malformed ciphertext. That is, any maliciously computed key material can be detected by the honest parties during encryption. Our strong-HiSE construction

satisfies strong correctness. This is enforced via use of NIZK proofs in the protocol. Each server along with their computation share attaches a proof of correct computation. The honest encryptor verifies each of these proofs and only proceeds if at least t many proofs verify. In case less than t proofs verify, the encryptor aborts. As a result, a complete run of the distributed encryption protocol, initiated by an honest encryptor, only produces valid ciphertexts.

D.3 Message Privacy for Strong HiSE

We define a sequence of hybrids. The first hybrid Hyb_0, corresponds to the real HiSE-MsgPriv(0) security game, whereas the final hybrid, Hyb_7, corresponds HiSE-MsgPriv(1).

Hyb_1: This game is identical to Hyb_0 with one modification: all NIZK proofs $(\pi_i, \tilde{\pi}_i)$ returned by honest servers (i.e., those not in the corrupted set C) in response to oracle queries are now generated by a simulator.

- The challenger runs $\mathsf{SimSetup}(1^\kappa)$ to generate $(pp_{\mathsf{com}}, \tau_{\mathsf{com}})$ and uses this pp_{com} in the global public parameters whereas τ_{com} is kept secret.
- When an honest server $i \notin C$ is required to generate a proof for an instance the challenger uses the NIZK simulator $\mathcal{S}_2$ and the trapdoor τ_{com} to generate a simulated proof π – the challenger can produce these proofs without knowledge of the actual secret shares α_i, β_i.

It is not hard to see that Hyb_0 and Hyb_1 are computationally indistinguishable – this follows directly from the computational zero-knowledge property of the NIZK.

Hyb_2: This game modifies the challenge oracle, $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch}}$. When the adversary submits its challenge query (j^*, m_0, m_1, S^*), the challenger proceeds as follows:

- It computes the Merkle root X^*_ε for the message vector m_0.
- Instead of computing the key $z^* = \mathcal{H}_2(j^*, N^*, X^*_\varepsilon)^\alpha$, it samples a uniform random $z^* \leftarrow_{\$} \mathbb{G}_2$.
- It then constructs the challenge cipher-tree $\widehat{\boldsymbol{C}}^*$ using this random z^*.
- All other oracle queries are handled exactly as in Hyb_1.

The indistinguishability of Hyb_1 and Hyb_2 is based on the TXDH assumption. The value z^* is effectively an evaluation of a NPR-DPRF on the input $(j^*, N^*, X^*_\varepsilon)$. The TXDH assumption guarantees that this output is pseudorandom and computationally indistinguishable from a random group element, even given the adversary's view of other DPRF evaluations from its queries to $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}en}}$. A detailed reduction is provided in Section D.3.1.

Hyb_3: This hybrid modifies the challenge decryption oracle, $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}$. When queried on a valid sub-tree $\widehat{\boldsymbol{C}}_\omega \leq \widehat{\boldsymbol{C}}^*$ (where all messages in the sub-tree are in $\mathcal{M}$), the challenger does as follows:

- Instead of computing the decryption key $\tilde{z} = z^* \cdot \mathcal{H}_2(X_\omega)^\beta$ (where z^* is random from $\mathbb{G}_2$), it samples a fresh, uniformly random element $\tilde{z} \leftarrow_s \mathbb{G}_2$.
- It uses this new random $\tilde{z}$ to perform the final decryption steps.
- All other oracle queries are handled as in Hyb_2.

The indistinguishability of Hyb_2 and Hyb_3 is based on the ℓ-masked BDDH assumption. In Hyb_2, the value z^* is already uniformly random. The adversary is given $pp = g_1^{\beta}$ and, through a query to $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}$, observes the interaction that leads to the reconstruction of $\tilde{z} = z^* \cdot \mathcal{H}_2(X_\omega)^\beta$. The ℓ-masked BDDH assumption implies that the resulting masking key for any ciphertext c_k outside the sub-tree $\widehat{\boldsymbol{C}}_\omega$, which is of the form $e(R_k, z^*)$, remains pseudorandom. Essentially, the term $\mathcal{H}_2(X_\omega)^\beta$ acts as a random mask on z^*, and the adversary cannot leverage the decryption of $\widehat{\boldsymbol{C}}_\omega$ to "unmask" z^* and gain any "additional information". A formal reduction is provided in Section D.3.2.

Hyb_4: This hybrid is identical to Hyb_3, except that in the challenge oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch}}$, the challenger now encrypts the message vector m_1 instead of m_0.

In Hyb_3, the challenge ciphertree $\widehat{\boldsymbol{C}}^*$ is generated using a key z^* that is uniformly random and independent of all other values in the game. The encryption of each message plus the associated values is performed via an XOR with a hash of a value derived from this random key, i.e., $E_k = \text{payload}_k \oplus \mathcal{H}_3(e(g_1^{r_k}, z^*))$. Since z^* is uniform and the pairing is non-degenerate, $e(g_1^{r_k}, z^*)$ is uniform in $\mathbb{G}_T$. As $\mathcal{H}_3$ is modeled as a random oracle, the output is a pseudorandom string. This means the encryption functions as a one-time pad. The adversary's view is therefore information-theoretically independent of the choice amongst m_0 and m_1.

Finally, we can reverse the sequence of modifications to go from hybrid Hyb_4 to Hyb_7, which is essentially the same as HiSE-MsgPriv(1) game.

- **Hyb_5:** Same as Hyb_4, but we revert the change made in Hyb_3. The challenge decryption oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}$ now again uses the correct computation $\tilde{z} = z^* \cdot \mathcal{H}_2(X_\omega)^\beta$, where z^* is still random. By the same argument as for $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_3$, this is indistinguishable from Hyb_4.
- **Hyb_6:** Same as Hyb_5, but we revert the change made in Hyb_2. The challenge oracle $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch}}$ now uses the correctly computed key $z^* = \mathcal{H}_2(j^*, N^*, X_\varepsilon^*)^\alpha$. By the same argument as for $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$, this is indistinguishable from Hyb_5.
- **Hyb_7:** Same as Hyb_6, but we revert the change made in Hyb_1. All NIZK proofs are now generated by the real Prove algorithm instead of the simulator. By the same argument as for $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$, this hybrid is indistinguishable from Hyb_6.

This concludes the proof. For completeness, in the next two sections we present formal reductions to show indistinguishability between $\mathsf{Hyb}_1, \mathsf{Hyb}_2$, and then $\mathsf{Hyb}_2, \mathsf{Hyb}_3$ respectively. Similarly the proofs for hybrid pairs $(\mathsf{Hyb}_5, \mathsf{Hyb}_6)$ and $(\mathsf{Hyb}_4, \mathsf{Hyb}_5)$ are done.

D.3.1 $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$

We show that: if there exists a PPT adversary $\mathcal{A}$ that can distinguish between Hyb_1 and Hyb_2 with non-negligible probability ε, we can construct a PPT algorithm (reduction) $\mathcal{B}$ that breaks the (Q, τ)-TXDH assumption with non-negligible probability ε/Q, where Q is an upper bound on the number of $\mathcal{A}$'s queries. $\mathcal{B}$ works as follows:

1. $\mathcal{B}$ receives a TXDH challenge instance: (g_2^A, g_2^b), where $\boldsymbol{A} \in \mathbb{Z}_q^{(\tau+1)\times Q}$ (with $\tau = t - |C| - 1$), and $\boldsymbol{b} \in \mathbb{Z}_q^{\tau+1}$. Here, $\boldsymbol{b}$ is either $\boldsymbol{A} \cdot \boldsymbol{u}$ for a random $\boldsymbol{u}$ (real case) or a random vector (ideal/random case).
2. $\mathcal{B}$ guesses which of $\mathcal{A}$'s encryption queries will be the challenge query. Let this be the k-th query, for $k \in [Q]$. $\mathcal{B}$ aborts if its guess is wrong.
3. $\mathcal{B}$ runs the Setup algorithm for strong HiSE, but it generates the trapdoor commitment parameters (pp_α, τ_α) itself. It provides pp and the corrupted keys $\{sk_i\}_{i \in C}$ to $\mathcal{A}$.

4. $\mathcal{B}$ maps the rows of the matrix $\boldsymbol{A}$ to the secret shares of the $\tau + 1$ honest parties it must simulate, and the columns to the Q potential encryption queries. The first row of $\boldsymbol{A}$ is associated with the master secret α, while the other τ rows are associated with the shares of τ honest parties.

5. The oracles are simulated by $\mathcal{B}$ as follows:

   – For the i-th encryption query (j, m, S) where $i \neq k$: $\mathcal{B}$ can compute the correct value $z = \mathcal{H}_2(j, N, X_\varepsilon)^\alpha$ because it corresponds to the i-th column of $\boldsymbol{A} \cdot \boldsymbol{u}$ where $u_i = 1$ and $u_j = 0$ for $j \neq i$. $\mathcal{B}$ also computes the shares for the honest parties from its view of $g_2^{\boldsymbol{A}}$. Moreover, it simulates the NIZK proofs using the trapdoors.

   – For the k-th encryption query, which is the challenge query $(j^*, \boldsymbol{m}_0, \boldsymbol{m}_1, S^*)$: $\mathcal{B}$ uses the challenge vector $\boldsymbol{b}$ from its TXDH instance to construct the response. Specifically, the reconstructed key will be $z^* = g_2^{b_0}$ (mapping the first element of $\boldsymbol{b}$ to the master secret evaluation), and the honest party shares will be derived from the other elements of $\boldsymbol{b}$.

   – Decryption Oracles can be simulated correctly since $\mathcal{B}$ knows the master secret key (α, β) implicitly in the exponents through the TXDH challenge matrix.

6. $\mathcal{A}$ eventually outputs its guess b'. If $\mathcal{A}$ was in Hyb_1, its environment was simulated using the "real" TXDH distribution ($\boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{u}$). If $\mathcal{A}$ was in Hyb_2, its environment was simulated using the random TXDH distribution ($\boldsymbol{b}$ is random). If $\mathcal{A}$ distinguishes the games, it effectively distinguishes the TXDH distributions. $\mathcal{B}$ outputs 1 if it thinks it received a real instance, and 0 otherwise, based on $\mathcal{A}$'s behavior.

The advantage of $\mathcal{B}$ is ε/Q due to the guessing of the challenge query. Since ε is non-negligible, so is $\mathcal{B}$'s advantage, which contradicts the TXDH assumption. This concludes this proof.

D.3.2 $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_3$

Again we construct a reduction $\mathcal{B}$ which breaks breaks the ℓ-masked BDDH assumption with non-negligible probability using a PPT adversary $\mathcal{A}$ that can distinguish between Hyb_2 and Hyb_3 with non-negligible probability. $\mathcal{B}$ works as follows:

1. It receives an ℓ-masked BDDH challenge: $(g_1^a, g_1^b, \{g_2^{d_i}, g_2^{c+bd_i}\}_{i\in[l]}, h \in \mathbb{G}_T)$. $\mathcal{B}$'s goal is to determine if $h = e(g_1^a, g_2^c) = g_T^{ac}$ or if h is a random element from $\mathbb{G}_T$.

2. $\mathcal{B}$ simulates the queries from $\mathcal{A}$ as follows:

   – $\mathcal{B}$ sets the HiSE public parameters by defining the master secret keys implicitly: $\alpha := a$ and $\beta := b$. It sets pp to include $g_1^\beta := g_1^b$.

   – $\mathcal{B}$ generates (pp_α, τ_α) and simulates the rest of the setup for $\mathcal{A}$.

3. It programs the random oracle $\mathcal{H}_2$ as:

   – When $\mathcal{A}$ makes its challenge query $(j^*, \boldsymbol{m}_0, \boldsymbol{m}_1, S^*)$, $\mathcal{B}$ computes the Merkle tree X^*. For the root, it programs $\mathcal{H}_2(j^*, N^*, X^*_\varepsilon) := g_2^c$.

   – For the nodes ω_i that could be queried in $\mathcal{O}^{\mathsf{hs\text{-}mp\text{-}ch\text{-}dc}}$ (i.e., nodes covering messages in $\mathcal{M}$), $\mathcal{B}$ programs $\mathcal{H}_2(X_{\omega_i}) := g_2^{d_i}$.

   – For all other inputs, $\mathcal{B}$ programs $\mathcal{H}_2$ to output random group elements whose discrete logs it knows.

4. It simulates the oracles as:

- $\mathcal{B}$ constructs the challenge ciphertree $\widehat{\boldsymbol{C}}^*$ as follows: For any message m_k where $m_0[k] \neq m_1[k]$, the masking key is $e(g_1^{r_k}, \mathcal{H}_2(j^*, N^*, X_\varepsilon^*)^a) = e(g_1^{r_k}, (g_2^c)^a) = e(g_1^a, g_2^c)^{r_k} = h^{r_k}$ (if h is the real challenge) or a random value (if h is random). $\mathcal{B}$ sets $R_k := (g_1^a)^{\delta_k}$ for a random δ_k, so it does not know $r_k = a\delta_k$, but can compute the masking key as h^{δ_k}. For messages where $m_0[k] = m_1[k]$, $\mathcal{B}$ can compute the correct masking key as it knows c.
- When queried on a node ω_i, $\mathcal{B}$ needs to provide the shares for $\tilde{z} = \mathcal{H}_2(j^*, N^*, X_\varepsilon^*)^\alpha \mathcal{H}_2(X_{\omega_i})^\beta = (g_2^c)^a (g_2^{d_i})^b = g_2^{ac+bd_i}$. The shares for this value can be computed using the challenge term $g_2^{c+bd_i}$.

5. If $h = g_T^{ac}$, $\mathcal{B}$ has perfectly simulated hybrid Hyb_2. If h is random, $\mathcal{B}$ has perfectly simulated hybrid Hyb_3. This concludes the proof.

D.4 Strong Authenticity

We start by describing the hybrids:

Hyb_0**:** This is the original HiSE-Auth game for the strong construction.

Hyb_1**:** This hybrid is same as Hyb_0, with only one modification: all NIZK proofs $(\pi_i, \tilde{\pi}_i)$ returned by honest servers (i.e., those not in the corrupted set C) in response to oracle queries are now generated by a simulator. To facilitate this, the challenger runs $\mathsf{SimSetup}(1^\kappa)$ to generate (pp_α, τ_α) and uses this pp_α in the global public parameters. The trapdoor τ_α is kept secret by the challenger. When an honest server $i \notin C$ is required to generate a proof, the challenger uses the NIZK simulator $\mathcal{S}_2$ and the trapdoor τ_α to generate a simulated proof.

It is easy to see that Hyb_1 and Hyb_0 are computaionally indistinguishable due to computational zero-knowledge property of the NIZK argument system.

Hyb_2**:** This hybrid is identical to Hyb_1, but with two additional checks performed inside the $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}ch}}$ oracle, identical to those in the proof for the base construction. First, the challenger verifies that the number of leaves N in a forged ciphertree's Merkle tree is consistent with the value N' used during any key derivation interactions for the root X_ε. Second, the challenger verifies that for any ciphertext $c_k = (R, S_{\ldots}, E)$, the value R is consistent with the value committed to in the corresponding leaf of the Merkle tree X. If either check fails for any forgery, the game aborts.

The indistinguishability of Hyb_1 and Hyb_2 follows from the same arguments as in the base authenticity proof. A mismatch in N would lead to an incorrect key derivation except with negligible probability due to the random oracle $\mathcal{H}_2$. A mismatch in R would require finding a collision in the random oracle $\mathcal{H}_4$, which also occurs with only negligible probability.

In Hyb_2, a successful forgery requires an adversary to create a valid ciphertree. In the strong construction, this means that any key shares provided by corrupt servers must be accompanied by NIZK proofs that are accepted by an honest verifier (in this case, the honest decryptor within the $\mathcal{O}^{\mathsf{hs\text{-}au\text{-}ch}}$ oracle). The knowledge soundness of the NIZK system ensures that any such valid proof implies that the associated key share was computed correctly according to the protocol.

Lemma 4. *The probability of an adversary winning in* Hyb_2 *is negligible, assuming the knowledge soundness of the NIZK system and the TXDH assumption hold.*

Proof. The knowledge soundness of the NIZK system guarantees that if any party (including a corrupt one controlled by the adversary $\mathcal{A}$) produces a valid proof π_i for an instance s_i,

then that party must "know" the corresponding witness w_i. Formally, there exists a PPT extractor $\mathcal{E}$ that can recover the witness.

Consider a forgery attempt by $\mathcal{A}$ on a fresh tuple $(j^*, N^*, X^*_\varepsilon)$. To construct a valid ciphertree, $\mathcal{A}$ needs the key $z^* = \mathcal{H}_2(j^*, N^*, X^*_\varepsilon)^\alpha$. This key is reconstructed from at least t shares. The adversary controls $|C| < t$ parties. For each corrupt party $i \in C$, to make its share z^*_i useful to an honest party (either the encryptor or the decryptor in the challenge oracle), it must also produce a valid NIZK proof π^*_i.

If $\mathcal{A}$ produces a valid proof π^*_i for the statement $s^*_i = (\gamma_{\alpha,i}, z^*_i, (j^*, N^*, X^*_\varepsilon))$, the knowledge soundness property implies that an extractor $\mathcal{E}$ can obtain the witness $w^*_i = (\alpha_i, \nu_{\alpha,i})$. This witness confirms that z^*_i was computed correctly as $z^*_i = \mathcal{H}_2(j^*, N^*, X^*_\varepsilon)^{\alpha_i}$. The same argument applies to decryption shares, where the witness includes both α_i and β_i.

Therefore, any share that an adversary produces from a corrupt party, if it is to be accepted by an honest party, must be a correctly computed share. An incorrectly computed share would be accompanied by an invalid proof, which would be rejected by the honest party's verification check, causing the protocol to abort.

This effectively reduces the strong authenticity game against a malicious adversary to the base authenticity game against an honest-but-curious adversary. The adversary cannot gain any advantage by deviating from the protocol in computing its shares. It is forced to compute its $|C|$ shares correctly. To forge a key z^* for a fresh root, it must therefore obtain at least $t - |C|$ additional correct shares from honest parties. The game's counter, ct, tracks exactly these interactions. Any attempt to produce more valid ciphertexts than accounted for by ct would require breaking the unpredictability of the underlying NPR-DPRF, which is reducible to the TXDH assumption, as shown in the proof of authenticity for the base construction. Thus, the probability of $\mathcal{A}$ winning in Hyb_2 is negligible. □

This concludes the proof of strong authenticity of the strong HiSE construction.

E Specific NIZK Proof Systems for Strong HiSE

We need NIZK proofs generated from sigma protocols via Fiat-Shamir for two different relations $\mathfrak{R}_a$ and $\mathfrak{R}_{a,b}$ as described in the construction. We describe the specific constructions in this section.

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 random 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 checks 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 9 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:

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

F Proof of ℓ-masked BDDH in the Generic Group Model

Boneh, Boyen and Goh extend the Generic Group Model defined by Shoup [Sho97] to include the class of assumptions in pairs of groups having an efficiently computable bilinear pairing. They define the Generalized Diffie-Hellman problem in [BBG05] 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 [BBG05], as defined by Boyen in [Boy08] to account for groups equipped with an asymmetric bilinear pairing.

Definition 11 ((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

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

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

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

We now state the notion of independence given in [BBG05] for a set of polynomials (P, Q, R, f).

Definition 12. 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

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

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 [Sho97, BBG05, Boy08].

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 [BBG05, Boy08]:

Theorem 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:*

$$Adv_{\mathcal{A}}^{(P,Q,R,f)\text{-}DDH} = \left| \Pr \left[\begin{array}{c|c} \mathcal{A}(\sigma_1(P(x_1,\ldots,x_n)), & (x_1,\ldots,x_n) \leftarrow_{\$} \mathbf{F}_p, \\ \sigma_2(Q(x_1,\ldots,x_n)), & b \leftarrow_{\$} \{0,1\}, \\ \sigma_T(R(x_1,\ldots,x_n)), & \nu_b \leftarrow f(x_1,\ldots,x_n), \\ \sigma_T(\nu_0), \sigma_T(\nu_1)) = b & \nu_{1-b} \leftarrow_{\$} \mathbf{F}_p \end{array} \right] - \frac{1}{2} \right|$$
$$\leq \frac{(q+2n+2)^2 d}{2p} \quad \text{(F.4)}$$

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

Lemma 5. *$f = ac$ 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 12. 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 11 given by BBG and thus can be proven secure in GGM by directly invoking the above theorem.
From theorem 3 we have,

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

G Hardness of TXDH from XDH assumption

This reduction closely follows the reduction of TDDH from DDH [ADL+22].

Lemma 6. *Let $\mathbb{G}_1, \mathbb{G}_2$ be two groups of prime order q, and let $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be generators. Then for any $\alpha, \beta \in \mathbb{N}$ and any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that*

$$Adv^{(\alpha,\beta)-\mathsf{TXDH}}(\mathcal{A}) \leq 2 \cdot Adv^{\mathsf{XDH}}(\mathcal{B})$$

Proof. The proof proceeds via a hybrid argument. An adversary $\mathcal{A}$ for the (α, β)-TXDH problem is given an instance $(g_2^{\boldsymbol{A}}, g_2^{\boldsymbol{b}})$, where $\boldsymbol{A} = (\boldsymbol{s}, 1) \otimes \boldsymbol{r} \in \mathbb{Z}_q^{(\alpha+1)\times\beta}$ for uniform random $\boldsymbol{s} \in \mathbb{Z}_q^{\alpha}, \boldsymbol{r} \in \mathbb{Z}_q^{\beta}$. The vector $\boldsymbol{b} \in \mathbb{Z}_q^{\alpha+1}$ is either $\boldsymbol{A} \cdot \boldsymbol{u}$ for a uniform random $\boldsymbol{u} \in \mathbb{Z}_q^{\beta}$ or is itself uniform random. We define three experiments:

- **REAL**: The adversary $\mathcal{A}$ is given $(g_2^{\boldsymbol{A}}, g_2^{\boldsymbol{b}})$ where $\boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{u}$.
- **IDEAL**: The adversary $\mathcal{A}$ is given $(g_2^{\boldsymbol{A}}, g_2^{\boldsymbol{b}})$ where $\boldsymbol{b} \in \mathbb{Z}_q^{\alpha+1}$ is uniform random.

– **HYB**: The adversary $\mathcal{A}$ is given $(g_2^{\boldsymbol{A}}, g_2^{\boldsymbol{b}})$ where $b_{\alpha+1} = \langle \boldsymbol{r}, \boldsymbol{u} \rangle$ and for all $i \in [\alpha]$, b_i is uniform random in $\mathbb{Z}_q$.

We will show that REAL is computationally indistinguishable from HYB, and HYB is computationally indistinguishable from IDEAL, by reducing each distinction to the XDH assumption.

Distinguishing REAL from HYB: We construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to solve the XDH problem. $\mathcal{B}$ receives an XDH challenge $(g_1, g_2, g_2^x, g_2^y, g_2^z)$, where either $z = xy$ or z is uniform random in $\mathbb{Z}_q$. $\mathcal{B}$ simulates the view for $\mathcal{A}$ as follows:

– It implicitly sets $\langle \boldsymbol{r}, \boldsymbol{u} \rangle = x$ and for each $i \in [\alpha]$, it implicitly sets $s_i = w_i^{(1)} + w_i^{(2)} y$ by choosing $w_i^{(1)}, w_i^{(2)} \in \mathbb{Z}_q$ uniformly at random. Since y is unknown to $\mathcal{A}$ and $w_i^{(1)}, w_i^{(2)}$ are random, each s_i is distributed uniformly at random from $\mathcal{A}$'s perspective.

– It chooses $\boldsymbol{r} = (r_1, \ldots, r_\beta) \in \mathbb{Z}_q^\beta$ uniformly at random.

– It computes the components of $g_2^{\boldsymbol{A}}$:

  – For $i \in [\alpha], j \in [\beta]$, it computes $g_2^{A_{ij}} = g_2^{s_i r_j} = (g_2^{w_i^{(1)}} (g_2^y)^{w_i^{(2)}})^{r_j}$.
  – For $j \in [\beta]$, it computes $g_2^{A_{\alpha+1,j}} = g_2^{r_j}$.

– It computes the components of $g_2^{\boldsymbol{b}}$:

  – For $i \in [\alpha]$, it computes $g_2^{b_i} = (g_2^x)^{w_i^{(1)}} (g_2^z)^{w_i^{(2)}}$.
  – It sets $g_2^{b_{\alpha+1}} = g_2^x$.

If $z = xy$, then for each $i \in [\alpha]$, $b_i = w_i^{(1)} x + w_i^{(2)} xy = (w_i^{(1)} + w_i^{(2)} y)x = s_i x = s_i \langle \boldsymbol{r}, \boldsymbol{u} \rangle$. Also, $b_{\alpha+1} = x = \langle \boldsymbol{r}, \boldsymbol{u} \rangle$. Thus, $\boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{u}$, and $\mathcal{B}$ has perfectly simulated the REAL experiment.

If z is uniform random, then for each $i \in [\alpha]$, $b_i = w_i^{(1)} x + w_i^{(2)} z$. Since $w_i^{(2)}$ is a non-zero random value and z is uniform random and independent of all other variables, b_i is uniform random in $\mathbb{Z}_q$. The component $b_{\alpha+1}$ remains correctly computed as $\langle \boldsymbol{r}, \boldsymbol{u} \rangle$. Thus, $\mathcal{B}$ has perfectly simulated the HYB experiment.

Therefore, any non-negligible advantage of $\mathcal{A}$ in distinguishing REAL from HYB translates directly into a non-negligible advantage for $\mathcal{B}$ in solving the XDH problem.

Distinguishing HYB from IDEAL: We construct another adversary $\mathcal{B}'$ that uses $\mathcal{A}$ to solve the XDH problem. $\mathcal{B}'$ receives an XDH challenge $(g_1, g_2, g_2^x, g_2^y, g_2^z)$, where either $z = xy$ or z is uniform random. $\mathcal{B}'$ simulates the view for $\mathcal{A}$ as follows:

– It chooses $\boldsymbol{s} = (s_1, \ldots, s_\alpha) \in \mathbb{Z}_q^\alpha$ uniformly at random.

– It chooses $r_2, \ldots, r_\beta \in \mathbb{Z}_q$ and $u_2, \ldots, u_\beta \in \mathbb{Z}_q$ uniformly at random.

– It implicitly sets $r_1 = x$ and $u_1 = y$ by using its challenge values g_2^x and g_2^y in place of $g_2^{r_1}$ and $g_2^{u_1}$ in its computations.

– It computes the components of $g_2^{\boldsymbol{A}}$:

  – For $i \in [\alpha], j \in [\beta]$, it computes $g_2^{A_{ij}} = (g_2^{r_j})^{s_i}$.
  – For $j \in [\beta]$, it computes $g_2^{A_{\alpha+1,j}} = g_2^{r_j}$.

– It computes the components of $g_2^{\boldsymbol{b}}$:

– For $i \in [\alpha]$, it chooses $b_i \in \mathbb{Z}_q$ uniformly at random.
– It computes $g_2^{b_{\alpha+1}} = g_2^z \cdot \prod_{j=2}^{\beta} (g_2^{r_j})^{u_j}$.

If $z = xy$, then $b_{\alpha+1} = xy + \sum_{j=2}^{\beta} r_j u_j = r_1 u_1 + \sum_{j=2}^{\beta} r_j u_j = \langle \boldsymbol{r}, \boldsymbol{u} \rangle$. The other components b_i for $i \in [\alpha]$ are chosen randomly. Thus, $\mathcal{B}'$ has perfectly simulated the HYB experiment.

If z is uniform random, then $b_{\alpha+1} = z + \sum_{j=2}^{\beta} r_j u_j$. Since z is uniform random, $b_{\alpha+1}$ is also uniform random in $\mathbb{Z}_q$. The other components b_i are also random. Thus, $\mathcal{B}'$ has perfectly simulated the IDEAL experiment.

Therefore, any non-negligible advantage of $\mathcal{A}$ in distinguishing HYB from IDEAL translates directly into a non-negligible advantage for $\mathcal{B}'$ in solving the XDH problem.

By the triangle inequality, the advantage of $\mathcal{A}$ in distinguishing REAL from IDEAL is bounded by the sum of its advantages in distinguishing REAL from HYB and HYB from IDEAL. Thus, $Adv^{(\alpha,\beta)-\mathsf{TXDH}}(\mathcal{A}) \leq 2 \cdot Adv^{\mathsf{XDH}}(\mathcal{B})$. □