Amortized Threshold Symmetric-key Encryption

Mihai Christodorescu
Visa Research
mihai.christodorescu@visa.com

Sivanarayana Gaddam
C3 Inc.
venganesh@gmail.com

Pratyay Mukherjee
Visa Research
pratyay85@protonmail.com

Rohit Sinha
Swirlds Inc.
sinharo@gmail.com

ABSTRACT

Threshold cryptography enables cryptographic operations while keeping the secret keys distributed *at all times*. Agrawal et al. (CCS'18) propose a framework for Distributed Symmetric-key Encryption (DiSE). They introduce a new notion of Threshold Symmetric-key Encryption (TSE), in that encryption and decryption are performed by interacting with a threshold number of servers. However, the necessity for *interaction* on each invocation limits performance when encrypting large datasets, incurring heavy computation and communication on the servers.

This paper proposes a new approach to resolve this problem by introducing a new notion called *Amortized Threshold Symmetric-key Encryption* (ATSE), which allows a "privileged" client (with access to sensitive data) to encrypt a large group of messages using a *single interaction*. Importantly, our notion requires a client to interact for decrypting each ciphertext, thus providing the same security (privacy and authenticity) guarantee as DiSE with respect to a "not-so-privileged" client. We construct an ATSE scheme based on a new primitive that we formalize as *flexible threshold key-derivation* (FTKD), which allows parties to interactively derive pseudorandom keys in different modes in a threshold manner. Our FTKD construction, which uses bilinear pairings, is based on a distributed variant of left/right constrained PRF by Boneh and Waters (Asiacrypt'13).

Despite our use of bilinear maps, our scheme achieves significant speed-ups due to the *amortized interaction*. Our experiments show 40x lower latency and 30x more throughput in some settings.

CCS CONCEPTS

• **Security and privacy** → **Key management**.

KEYWORDS

threshold cryptography; authenticated encryption; secret management systems; distributed pseudo-random functions; constraint pseudorandom functions

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3485256

ACM Reference Format:
Mihai Christodorescu, Sivanarayana Gaddam, Pratyay Mukherjee, and Rohit Sinha. 2021. Amortized Threshold Symmetric-key Encryption. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 22 pages. https://doi.org/10.1145/3460120.3485256

1 INTRODUCTION

Enterprise key-management systems for massive data encryption. Large modern enterprises acquire enormous amount of data everyday, and protect them using encryption. They rely on dedicated key management systems to protect encryption keys on behalf of the applications. The typical deployment has three main entities: one or more *encrypting clients*, a set of *key-management servers* holding the key material, and a *storage service* that stores encrypted data. Moreover, there are other *non-encrypting clients* (e.g. application servers), which may frequently access the storage service to decrypt specific data as required. Among them the key management server is often backed by Hardware Security Modules (e.g. AWS KMS, Azure KMS, GCP KMS). However, in practice, they admit significant operational concerns surrounding scalability (especially during bursty traffic), availability, and cost.

Threshold Crypto for key-management. To address these concerns of HSMs, we look towards threshold cryptography, where the secret-key is split into shares and distributed across multiple key-management servers (namely, commodity server machines) and never reconstructed during use. A cryptographic operation requires the participation of at least a threshold number of servers, and security holds as long as the attacker compromises less than the threshold number of servers. The benefits of threshold cryptography have caught the attention of practitioners. For example, the U.S. National Institute of Standards and Technology (NIST) has initiated an effort to standardize threshold cryptography [7]. Furthermore, an increasing number of commercial products use the technology, such as the data protection services offered by Vault [9], Coinbase Custody [2], and the HSM replacements by Unbound Tech [8] etc.

Distributed Symmetric-key Encryption (DiSE). Recently, Agrawal et al. gave the first formal treatment of threshold symmetric-key encryption (TSE) in DiSE [13], and provided a construction based on distributed pseudo-random functions (DPRF). In a nutshell the construction works as follows: to encrypt a message m, the client locally produces a commitment of m, and sends the commitment to the servers, who upon authenticating the client return the PRF (partially) evaluated over the commitment (using their key-shares); the client combines a threshold number of responses to construct a

message-specific key that a client uses to locally encrypt m. Moreover, to defend against malicious servers, which can otherwise cause the client to derive an "invalid" key and permanently lose the plaintext, DiSE requires the servers to provide a zero-knowledge proof of their PRF evaluations. Not surprisingly, the round-trip interaction and server-side computation for each message becomes prohibitively expensive when encrypting *large datasets*. The latency and throughput worsens when the key-management servers are geo-distributed, as often recommended for availability and security.

Inherent limitation of DiSE. A closer look into DiSE exposes that the limitation is somewhat inherent to the requirement of their *authenticity* notion, in that *any* ciphertext must be authenticated via interacting with a threshold number of servers. While this strong guarantee is critical for some applications (e.g., multi-device authentication [13]), it is an *overkill* for our setting of large database encryption (elaborated in Sec. 2.1).

A naïve approach: Encrypting in group. Our exploration starts with the insight that if an encrypting client has access to a large group of messages (e.g. an entire dataset) to be encrypted, then interaction may not be necessary for authenticating each ciphertext individually. Instead, the client may send a *short* commitment to the group of messages to the servers; the servers return a common *group-specific key* to encrypt and *locally authenticate* all messages together. This lets us attain the same authenticity property as DiSE's, but without the need for interaction on each individual message. Viewed from a different angle, we observe that the encryptor client is already trusted or "privileged" as it handles large volumes of data in the clear. Therefore, we could allow that encryptor to encrypt an entire dataset using a single group-specific key (instead of message-specific key a la DiSE) derived via a *single interaction*, while also ensuring that this key can not be used to encrypt any other message that is not part of the data set (guaranteed by the commitment).

Issue with group-encryption. However, this approach turns out to be problematic during decryption of individual data records, which may be performed by any other "unprivileged" client, which is trusted to a lesser extent than the so-called "privileged" dedicated encryptor (such as analytics workloads that read a subset of the database). In particular, there is no obvious way to allow decryption of a single ciphertext without revealing the group-specific key and hence the entire dataset. Clearly, such revelation falls short of a desired privacy guarantee for an individual message. So, we need a more fine-grained solution that enables encryption of a group message with just one interaction, but decryption can be done for individual ciphertexts (each requiring one interaction).

Our approach. This paper provides a new scheme to that end. In a nutshell, our scheme amortizes the communication and computational effort for the encryptor without altering the security requirements, so we call it *Amortized Threshold Symmetric-key Encryption* (ATSE).

Balancing sensitivity and amortization. A consequence of amortization is making the derived encryption keys (that is group-specific keys) more sensitive. For example, if the encryptor encrypts 100 plaintexts using one such key derived via interaction, the same key can also be used to decrypt those 100 ciphertexts, and is therefore sensitive.[1] Nevertheless, since the privileged encryptor handles the entire dataset, this increased sensitivity of *short-lived* group-specific keys is affordable in favor of the efficiency gains from amortization.

Key challenge. In ATSE, the encryptor interacts with the servers to derive a group-specific key. Then, for each message, a unique message-specific key is derived locally, which is then used to encrypt the message deterministically. During decryption, the same message-specific key must be derived *directly* only from an individual ciphertext without any access to the group-specific key. The main challenge lies in deriving *identical keys* in these two different modes, while restricting access to the more sensitive group-specific key only to the designated encryptor, thereby achieving the same privacy and authenticity guarantees for an individual ciphertext.

Flexible Threshold Key-derivation. To resolve the challenge we introduce a new core primitive called *Flexible Threshold Key-Derivation* (FTKD), in that a client can derive pseudorandom keys in various modes. Each client can use an entire input to derive the *whole-key* (unique to the whole input), or just part of her input to derive *partial-keys*, which can be locally combined then with the remaining input to obtain the *same* whole-key. Pseudorandomness holds for any such whole-key even conditioned on knowledge of other whole-keys (derived from the same partial-key) or unrelated partial keys. Looking ahead, when constructing an ATSE scheme, we use a partial-key as a group-specific key whereas a whole-key as a message-specific key. This can also be thought of as a distributed version of a special type of constrained PRF, known as left/right constrained PRF. Indeed, our FTKD construction is based on the left/right constrained PRF proposed by Boneh and Waters [20]. Now, the main challenge boils down to deriving a whole-key, which is consistent across different modes and any threshold number of participants. Our construction uses bilinear pairings to that effect. We believe that our FTKD notion may be of independent interest and is useful in applications beyond ATSE (e.g. see Sec. 2).

Performance evaluation. We evaluate our scheme in a variety of scenarios, from LANs to geo-distributed servers and with varying number of servers. Since each server interaction incurs several group operations — from lagrange interpolation within DPRF [45], and zero-knowledge proofs to detect malicious servers (see Section 5.2) — we achieve significant speed-up at the encryptor from amortization, despite our use of a comparatively heavier operation like bilinear pairing. Our experiments indicate *latency reduction of upto 40x* and *throughput improvement of upto 30x* compared to a parallelized DiSE implementation,[2] when interacting with 24 servers and encrypting 10K messages in bulk. More importantly, when encrypting larger groups of messages in bulk (say 10K), the latency and throughput characteristics of our scheme is unaffected by both network latency and the threshold size, thus incentivizing

[1] This happens due to the symmetric nature of our scheme and can be avoided by using (threshold) public-key encryption: a public-key can not be used to decrypt, by definition. However, public-key encryption alone is unfit for our setting because it allows *anyone* to encrypt data thereby falling short of any conceivable authenticity requirement. One might think about adding signatures, but that leads to other issues as elaborated in Appendix D.

[2] In this version, to encrypt, say 100 messages in a group, 100 instances of DiSE scheme are executed simultaneously.

deployments that have more, geo-distributed servers, increasing availability and security.

Summary of contributions.

- We *formalize* the notion of amortized threshold symmetric-key encryption, based on which our scheme facilitates large data encryption in bulk.
- We *construct* an amortized threshold symmetric-key encryption scheme based on a new primitive called flexible threshold key derivation. We construct a flexible threshold key derivation scheme combining the ideas from a left/right constrained PRF by Boneh and Waters [20] and a distributed PRF by Naor, Pinkas and Reingold [45]. We prove the security based on bilinear decisional Diffie-Hellman assumption in the random oracle model. Additionally we require a Merkle-tree like commitment scheme, which we formalize as *group commitments*. We explore several interesting properties of group commitments as required for our construction.
- We implement and evaluate the system in several deployment scenarios, and report the performance measurements.

2 TECHNICAL OVERVIEW

We informally describe ATSE and its requirements in this section, with the help of a use case. Then, we provide overview of our constructions. Finally, we outline possible extensions of our techniques.

2.1 Use Case

Consider an enterprise such as Visa [10] that processes transactions containing sensitive data, such as a payments network that processes upwards of 5000 transactions per second [5]. In addition to processing these transactions, this enterprise performs offline analytics on this data. Therefore the data must be stored securely in a way such that it can be accessed later.

We find the following data pipeline to be a common design for enterprises that handle sensitive data. At the source of data ingress, a designated application is responsible for one task: encrypting each record as it arrives and forwarding it to a storage service. As it accepts all user data in the clear, we will refer to this application as a *privileged* client or encryptor, and it is often hardened with security defenses. The data is later accessed by different applications, each performing some analytics on some subset of the entire database. In practice, as different developers possess varying levels of security expertise, we do not trust these applications to access the entire data; so, we call them *unprivileged* clients or decryptors.

We want the encryptors to encrypt a large number of messages efficiently (without a round-trip interaction with the key management servers, for instance), but we also want authenticity: the encryptor must only produce legitimate ciphertexts, where the key given to the encryptor is bound to the set of messages. We also wish to force decryptors to interact with the server(s) for decrypting every message, for fine-grained access control and auditability.

Our design is flexible in that it can be used with single or multiple encryptors and decryptors. However, since our formalization (ATSE) only supports amortization of encryption,[3] this is especially useful in write-heavy applications where different clients continuously encrypt and store data streams, but later access individual records on-demand. A good example is a payment gateway (e.g. Cybersource [3]), which continuously encrypts and stores transactions immediately after processing, and the decrypted transaction data is seldom accessed to resolve issues like payment disputes.

[3]We remark that it is just as easy to make the decryption amortized, using our building block FTKD. However, we do not formalize it due to lack of a clear motivation. We briefly mention a potential application called threshold symmetric IBE in Sec. 2.4.

2.2 Requirements

2.2.1 *Functional Requirements.*

- **distributed/threshold**: Similar to DiSE, we seek a threshold symmetric encryption scheme in lieu of HSMs, where the long-term symmetric key is secret-shared in an t (threshold) out of n (total number of participants) manner onto a collection of commodity servers and never reconstructed. The servers must be administered by independent parties for security and geo-distributed for availability.
- **amortized**: We are looking for an amortized framework, in that a privileged client can encrypt multiple data records in "bulk" with only one interaction, yet an unprivileged client is compelled to interact in order to decrypt each record; this allows a *fine-grained* access control, and minimizes the potential exposure of data in the event of a compromise of unprivileged clients.

2.2.2 *Security Requirements.*

- **privacy**: A decryptor, when decrypts a particular ciphertext, may not be able to learn any information about *any other* ciphertext, even those belonging to the same group. An encryptor may derive multiple group-specific encryptions keys, but those keys would not let her breach the privacy of any other ciphertext for which she does not own the key specific to that group. In particular, we must enforce that one encryptor may not be able to derive encryption *keys for another encryptor*.
- **(malicious) correctness**: The clients must be able to detect *malicious responses* from the servers lest the client derive incorrect keys, rendering the encrypted data useless. While optional for some applications of DiSE, this property is crucial for our setting as further discussed in Remark C.4.
- **authenticity**: We must ensure that the ciphertexts produced are "authentic". More formally, we can say that authenticity guarantees that as long as there are $< t$ corruptions each valid ciphertext *must* be produced via *properly* interacting with at least one honest server. (More intuitions are provided in Remark C.13.)

2.3 Our Construction

2.3.1 Flexible Threshold Key-derivation. Our ATSE scheme relies on a core component, formalized in this paper as a flexible threshold key-derivation (FTKD) scheme, which lets clients derive keys with different granularity levels: either a whole-key bound to the whole input, or a partial-key bound to only a part of the input — a partial-key can be combined with the other part of the message to derive the same whole-key locally.

This notion can also be thought of as a distributed variant of left/right constrained PRF [20], in that, for each input (x, y), it is possible to generate constrained keys $k_{x\|\star}$, $k_{\star\|y}$, such that one can combine $k_{x\|\star}$ (resp. $k_{\star\|y}$) with y (resp. with x) to obtain $w :=$

$\mathsf{PRF}_k(x, y)$. Furthermore, given any constrained key $k_{x\|\star}$ the value $\mathsf{PRF}(x', \cdot)$ remains pseudorandom as long as $x' \neq x$.

Our FTKD notion offers similar functionality and security guarantee in the distributed / threshold setting. In particular, the long-term key k is secret-shared across n parties such that any $t \leq n$ shares are sufficient to recover k. Any party $i \in [n]$ can interact with any other $t-1$ parties to derive either *partial-keys* $k_{x\|\star}$ (left-key), $k_{\star\|y}$ (right-key) or directly the output value w such that $k_{x\|\star}$ (resp. $k_{\star\|y}$) can be *locally* combined with y (resp. x) to obtain w (this is called *key-consistency*, which is formalized in Definition 5.1). Furthermore, the same w is derived as long as there are *any* t participants (this is *threshold-consistency*, which is formalized in Def. 5.1). The output w must be pseudorandom even conditioned on arbitrary many input/output pairs or constrained keys as long as there is *not enough* information to derive w trivially (pseudorandomness is formalized in Def. 5.4). Furthermore, we need a correctness property (Def. 5.6) in the presence of malicious parties — this ensures that no party can successfully cheat by sending an incorrect response.

Our FTKD construction uses ideas from the left/right constrained PRF proposed by Boneh and Waters [20], which is based on bilinear pairing over groups $G_1 \times G_2 \rightarrow G_T$ and uses independent hash functions $\mathcal{H}_0, \mathcal{H}_1$, modeled as random oracles in the proofs. For input (x, y) and key k, the PRF is computed as $\mathsf{PRF}_k(x, y) := e(\mathcal{H}_0(x), \mathcal{H}_1(y))^k$. The constrained keys $k_{x\|\star}$ and $k_{\star\|y}$ are constructed as $\mathcal{H}_0(x)^k$ and $\mathcal{H}_1(y)^k$ respectively. Given $k_{x\|\star}$, one can compute $\mathsf{PRF}_k(x, y)$ for any y using pairing $e(k_{x\|\star}, \mathcal{H}_1(y))$; similarly $k_{\star\|y}$ can be combined with x.

Note that, the structure of this constrained PRF is similar to the "key-homomorphic" structure of the (distributed)PRF by Naor, Pinkas and Reingold (NPR [45], used in DiSE) which defines a prf on x as $\mathsf{PRF}_k(\mathcal{H}(x))$ (where $\mathcal{H}$ is modeled as a random oracle). Our FTKD construction leverages this observation simply by secret-sharing the key k. Each party i holds a share k_i. On a left-key request on a left-input x, party-i returns $\mathcal{H}_0(x)^{k_i}$. The client can combine t such partial computations using Lagrange interpolation in the exponent to obtain the left-key $k_{x\|\star} := \mathcal{H}_0(x)^k$ — let us call this mode of key-derivation a *left-mode* (similarly a right-key $k_{\star\|y}$ can be constructed in a *right-mode*). Moreover, it is also possible for a client to directly obtain w just by making a query on the whole input (x, y) in another mode (say, *whole-mode*); in response each server sends back $e(\mathcal{H}_0(x), \mathcal{H}_1(y))^{k_i}$.

The pseudorandomness follows assuming BDDH over the bilinear group in the random oracle model, albeit extending to the distributed setting requires using arguments from DiSE [13] and NPR [45]. The malicious correctness is obtained using non-interactive zero-knowledge proof (and trapdoor commitments) a la DiSE.

2.3.2 Our ATSE scheme. Now, given a FTKD scheme, we are ready to describe our ATSE construction at a high level. However, before that, let us briefly describe how the DiSE TSE scheme works given a DPRF: an encryptor with identity id produces a commitment γ of the message m (with randomness/opening r); the encryptor then interactively evaluates a DPRF on input $(\mathsf{id}\|\gamma)$ to obtain a pseudorandom message-specific key $w = \mathsf{DPRF}(\mathsf{id}\|\gamma)$; which is then used to produce a deterministic encryption $e = \mathsf{PRG}(w) \oplus (m\|r)$. The ciphertext is of the form (id, γ, e). One may decrypt by first re-evaluating the DPRF on the first two values $(\mathsf{id}\|\gamma)$ of the ciphertext, then decrypting e, and finally checking the consistency of commitment γ with opening (m, r).

We take an analogous approach for ATSE. However, instead of standard commitment we use Merkle-tree commitment, because Merkle-tree commitments come with the exact fine-grained property we are looking for: an ordered group of messages $(m_1, \ldots m_N)$ can be committed together with a *short* commitment, which can be opened later for any m_i without revealing any other m_j (for hiding of m_j we use randomized hashing at the leaves). In our ATSE scheme, the encryptor produces a Merkle-tree commitment γ of a group of messages $(m_1, \ldots, m_N)$ with randomness $(r_1, \ldots, r_N)$. The root-to-leaf path, q_i is unique to each message m_i. The encryptor then computes a partial-key (say, left-key) $k_{\mathsf{id}\|\gamma\|\star}$ by computing FTKD on the partial input $(\mathsf{id}\|\gamma)$ of the whole input $(\mathsf{id}\|\gamma\|q_i)$. Then, each message m_i can be locally encrypted similar to DiSE: $e_i = \mathsf{PRG}(k_{\mathsf{id}\|\gamma\|q_i}) \oplus (m_i\|r_i)$ where the message-specific whole-key $k_{\mathsf{id}\|\gamma\|q_i}$ is locally derived from group-specific key $k_{\mathsf{id}\|\gamma\|\star}$ and q_i. The ciphertext for m_i is $(\mathsf{id}, \gamma, q_i, e_i)$. A decryptor directly derives the whole-key $k_{\mathsf{id}\|\gamma\|q_i}$ using the whole input $(\mathsf{id}\|\gamma\|q_i)$; using that she can decrypt e_i to get $(m_i\|r_i)$ and then finally verify the root commitment γ with opening r_i with respect to the path q_i.

The functionality requirements, such as threshold computation and amortization follows directly from the underlying FTKD. The privacy against a decryptor is guaranteed by the pseudorandomness of the whole-key $k_{\mathsf{id}\|\gamma\|q_i}$ (even given any other whole-key or unrelated partial-key) and the hiding of the commitment scheme. Privacy against an encryptor is enforced by including id into the input of the FTKD, as this makes the keys bound to the identity of an encryptor. However, to prevent a cheating encryptor from impersonation, each server verifies whether an encryption request including identity id is indeed coming from a client with identity id — this requires authenticated channels (also needed by DiSE).

Authenticity relies on more sophisticated arguments. In fact, in the above construction, since γ, as it is, independent of the number of messages, it becomes tricky to keep track of the exact number N of valid ciphertexts generated per invocation. To fix this, we use a technique to augment the Merkle-tree: at each node we produce the hash values with a number denoting the total number of leaves under that node — the leaves are labeled 1, whereas the root is labeled exactly N. While opening, the verifier checks whether this labels recursively sums up to N at the top via the given leaf to root path. This augments the Merkle-tree to satisfy a new property, that we call cardinality-binding. This enables our construction to achieve a one-more authenticity notion analogous to DiSE. We put forward an abstraction, called *group commitments* (Sec. 6), to capture all the properties that we need from the commitment scheme.

We illustrate a flow of our simplified [4] ATSE scheme in Figure 1. It is worth noting how the key management service is used. Though, for simplicity we present our ATSE framework as a symmetric distributed system, where each party is treated equally, it can be deployed differently with dedicated roles of privileged encryptors, unprivileged decryptors, key-holders. Furthermore, one could put policies like any encryption request must come from specific clients. We do not formalize these in the paper.

[4] We do not use the "labeling trick" here for simplicity; full details are in Figure 11.

Amortized Encryption of $(m_1, \ldots, m_N)$ by privileged client-1

- sample random values $(s_1, \ldots, s_N)$. Compute Merkle-tree MT on the set $((1 \| m_1 \| s_1), \ldots, (N \| m_N \| s_N))$ and compute the commitment γ to MT.
- send $(\gamma, \omega := \mathcal{H}(1, \gamma)$, 'Group-key') to servers $\{2, 3, 4\}$. Each server i verifies $\omega = \mathcal{H}(1, \gamma)$ and returns $\mathcal{H}_0(\omega)^{k_i}$ on success.
- compute group-key $\mathsf{gk} := \mathcal{H}_0(\omega)^k$ by Lagrange interpolation in the exponent from the server response and its own value $\mathcal{H}_0(\omega)^{k_1}$.
- for each m_i, compute message-key $\mathsf{mk_i} := e(\mathsf{gk}, \mathcal{H}_1(q_i))$, where q_i is the unique commitment derived from the root-to-leaf path in MT.
- store ciphertexts $c_1, \ldots, c_N$, where $c_i = (1, \gamma, q_i, \mathsf{PRG}(\mathsf{mk}_i) \oplus (m_i, s_i))$.

Decryption of c_j by unprivilged client-7

- parse c_j as $(\mathsf{id}, \gamma, q, x)$.
- send $(\mathsf{id}, \gamma, q,$ 'Message-key') to servers $\{4, 5, 6\}$. Each server i computes $\omega = \mathcal{H}(\mathsf{id} \| \gamma)$ and returns $e(\mathcal{H}_0(\omega), \mathcal{H}_1(q))^{k_i}$.
- compute message-key $\mathsf{mk} := e(\mathcal{H}_0(\mathsf{id}, \gamma), \mathcal{H}_1(h))^k$ by Lagrange interpolation in the exponent from the server response and its own value $e(\mathcal{H}_0(\omega), \mathcal{H}_1(q))^{k_7}$.
- extract $(m, s) \leftarrow \mathsf{PRG}(\mathsf{mk}) \oplus x$
- verify unique commitment q with respect to γ and opening s. On success, output m, otherwise output $\perp$.

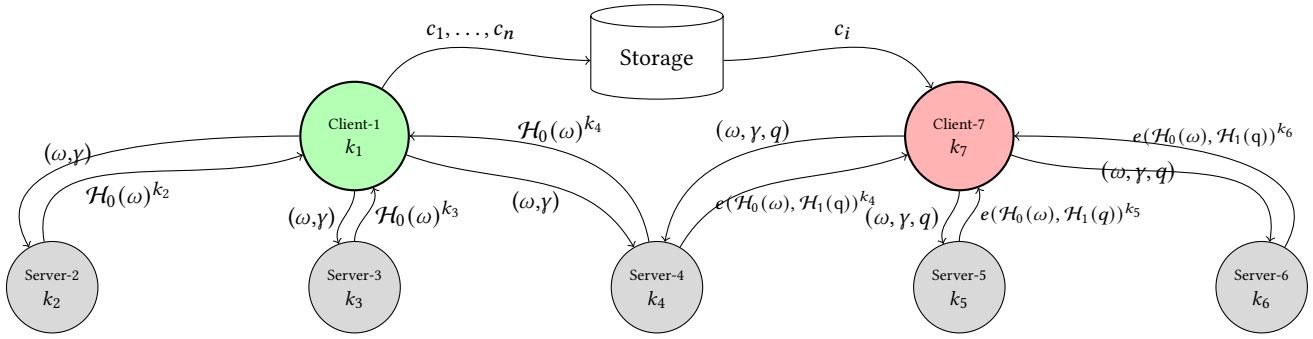


Figure 1: Flow of ATSE protocol for $n = 7$ and $t = 4$. Privileged client (id 1) encrypts messages $m_1, \ldots, m_n$ in bulk using $t - 1$ servers $2, 3, 4$, and unprivileged client only decrypts m_i using servers $4, 5, 6$. To encrypt, client computes Merkle tree with commitment γ sends (ω, γ) to each server i, who returns $\mathcal{H}_0(\omega))^{k_i}$ if $\omega = \mathcal{H}(1, \gamma)$. Client then locally derives per-message keys using pairings. A client decrypts c_i by sending (ω, q) to each server i, who returns $e(\mathcal{H}_0(\omega), \mathcal{H}_1(q))^{k_i}$. Client then combines responses and computes the per-message key locally.

2.4 Other Applications of FTKD

Using FTKD for Identity-based Threshold Symmetric Encryption. Our FTKD notion can be used to achieve another notion of threshold symmetric encryption scheme simply by reversing the roles of the privileged and the unprivileged clients from ATSE. In particular, we can construct a primitive, in that many unprivileged clients can encrypt messages $m_1, m_2, \ldots$ with respect to a particular privileged client's identity j just by directly deriving the message-specific whole-key $\mathsf{wk}_i := e(\mathcal{H}_0(j), \mathcal{H}_1(\mathsf{Com}(m_i, s_i)))^k$ based on the identity j and the message m_i. The privileged client with identity j may obtain its "identity-key" (which is a renaming of a partial-key in this context) $\mathsf{idk} := e(\mathcal{H}_0(j))^k$ later by interacting with servers when the servers do check whether the requester's identity is indeed j. Once the identity-key is obtained, each individual whole-key wk_i can be obtained by pairing $\mathcal{H}_0(\mathsf{idk})$ with $\mathcal{H}_1(\mathsf{Com}(m_i, s_i))$ (which is included in the ciphertext) locally. Therefore, any message encrypted under j's identity is now accessible to party j. This scheme can be termed as *identity-based threshold symmetric encryption.*

Encrypting Tabular data. Our FTKD scheme can also be used for encrypting tabular data with amortization. Each cell in a table can be uniquely identified by its row number i and column number j. A message-specific unique whole-key is given by $e(\mathcal{H}_0(i), \mathcal{H}_1(j))^k$. One can obtain a left-key $\mathcal{H}_0(i)^k$ for row-i and use that to locally encrypt/decrypt all elements in that row. Similarly, one can use a right-key $\mathcal{H}_1(j)^k$ to encrypt/decrypt elements across columns.

3 RELATED WORK

3.1 Competing approaches

3.1.1 DiSE. The most closely related work is **DiSE [13]** (recently extended to adaptive setting in [**?**]), which is the first work to propose a definition and construction for threshold, authenticated, symmetric-key encryption, based on distributed pseudo random functions. An alternative to our approach could be to run *DiSE in parallel* for many messages. We provide an experimental comparison with this approach with ours in Section 8.

3.1.2 MPC. Secure **multi-party computation (MPC)** enables cryptographic algorithms (in a circuit form) such as AES wherein keys remain split during operation. Prior works have shown how to evaluate ciphers such as AES [4, 25, 36, 47], but we find them to be too expensive for our large-data encryption setting, as they require *many rounds of communication*, *high bandwidth*, and a *prohibitively heavy preprocessing phase*. The preprocessing phase in these works depends on the number of operations to be performed in the online phase. This becomes a potential bottleneck in our setting, where input data is continuously arriving, as preprocessing must be performed *repeatedly* (by alternating or running concurrently with the online phase). In contrast, our scheme has a one-time setup phase which is independent of the number of operations to be performed.

To our knowledge, the state-of-art construction of the AES is by Keller et al. [39]. We compare with their work (specifically, their AES-LT scheme for 2PC) along the following lines:

Compatibility: Unlike our work, [39] implements a standardized AES cipher, and therefore provides backward compatibility.

Throughput: [39] gives 236K ops / sec in the online phase, but the (bottleneck) pre-processing step gives 17 ops / sec (LAN setting). In contrast, our scheme provides around 23K ops / sec.
Network sensitivity: In the WAN setting, [39]'s throughput lowers to 29K ops / sec and pre-processing lowers to 0.83 ops / sec. Our scheme is insensitive to network timings (due to the amortized interaction), and achieves 23K ops / sec in all network conditions.
Bandwidth complexity: MPC protocols use a non-blackbox approach, in that the computation is modeled as a boolean circuit and the communication is proportional to the circuit size. For example, [39] uses 8.4 MB of bandwidth for *each AES block*, whereas we require 132^*t bytes ($t < n$ is the corruption threshold; set $t = 2$ to compare with them) bytes for *an unbounded-size group of messages* — that is, our bandwidth is independent of number and size of messages. So, even a parallel execution of their construction to encrypt a large number (say 10000) of messages in a batch would lead to a huge blow up in bandwidth.
Round complexity: 10 rounds per block in [39] vs. our 2 rounds
Flexibility: Our scheme can easily be adapted to any n and any threshold t — our implementation considers n as large as 24 and various t values for each n. [39] considers only n out of n settings for a few small values of n (upto 5). While it is possible to implement a version of their scheme that supports arbitrary t out of n, that may lead to considerable overhead and require significant changes.

MPC-friendly ciphers, such as LowMC [15], provide a slight improvement in some metrics but still require at least 12 rounds, and they implement a non-standard cipher (so does not have the compatibility advantage as AES) — our encryption is also non-standard but provides significantly better performance.

3.2 Additional related works

Our notion of FTKD enables delegating generation of pseudorandom values in a restricted manner. This is reminiscent of (also borrows ideas from) **constrained PRF**, albeit in a threshold setting. Constrained PRFs, first proposed in [20, 40], found many applications (mostly on the theoretical side of) cryptography, such as broadcast encryption [20], attribute-based encryption [16], indistinguishability obfuscation [48], watermarking [41], keyword search over shared cloud data [50], etc. Our work can be thought of as a *new application* of constrained PRF (more precisely, a threshold version of constrained PRF), more on the applied side.

Threshold constructions have been designed for **pseudorandom functions** [19, 29–32, 43, 45, 46]. Naor et al. [45] propose a mechanism for encrypting messages using their DPRF construction, albeit without any authentication guarantee. Beyond symmetric-key primitives, **threshold public-key encryption** is also well studied [21, 26–28, 34, 45, 49]. Similar to the symmetric key setting that we study, the decryption key is secret-shared amongst a set of servers, requiring at least a threshold to decrypt the ciphertext. More recently, Jarecki et al. [38] provide a **threshold, updatable, and oblivious key management** system based on oblivious pseudo-random functions, based on public key encryption. However, as we discussed earlier and also in Appendix D, public-key schemes allow *any* party (potentially unprivileged entity) to encrypt a message and produce the ciphertext, without any interaction, thus falling short of our requirement of authenticity. Works like PASTA [12], BETA [11] and PESTO [17] provide frameworks for threshold authentication. While their designs can be extended to achieve a different notion of token/signature privacy (such as threshold blind signatures [42]), they do not consider the problem of message privacy (blind signatures do not offer decryption). Similar to DiSE, our framework is targeted towards achieving authenticated encryption/decryption of the message and hence is broadly incomparable with those works despite some technical similarities (namely, both give threshold access structure and use bilinear maps).

Our notion of group commitment (c.f. Sec. 6) has some similarity with the notion of **vector commitments [22]** — vector commitments too produce a short commitment of a sequence/vector of messages and offer so-called position-binding similar to group commitments. However, what distinct their notion from ours is the requirement of having both short opening *and* short commitment — we only require the group-commitment (root of the Merkle-tree) to be short. Moreover, their notion offers other features such as updatability, that is not present in ours. Nevertheless, their constructions are based on computationally-heavy public-key operations.

4 NOTATION AND PRELIMINARIES

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

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

We denote a sequence of values $(x_1, x_2, \ldots)$ by a standard vector notation $\mathbf{x}$, and it's i-th element is denoted by $\mathbf{x}[i]$. $|\mathbf{x}|$ denotes the number of elements in the vector x. A list can be thought of as an ordered set; the i-th element of a list L is denoted by $L[i]$. Lists and vectors can be used interchangeably. Concatenation of two strings a and b is denoted by $(a\|b)$, or (a, b). Let $x \in \{a, b\}$ is a variable that can have only two values a or b. We use $\overline{x}$ to denote the value within this set which is different from x. For example if $x = a$ (resp. $x = b$) then $\overline{x} = b$ (resp. $x = a$).

We write $[j : x]$ to denote that the value x is private to party j. 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 $[\![\mathbf{x}]\!]_S$ to denote that each party $i \in S$ has a private value x_i.

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

4.1 Security games and oracles.

While our formalization uses intuitive security games, to handle many cases, the descriptions often become cumbersome. Therefore, we use simple pseudo-code notation that we explain next.

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.

4.2 Building blocks used in our constructions

4.2.1 Bilinear Pairing. Our construction relies on bilinear pairing. Following the notation of [6] we consider three groups G_0, G_1, G_T all of prime order p and an efficiently computable map $e : G_0 \times G_1 \rightarrow G_T$ which is *bilinear* and *non-degenerate*. We rely on *bilinear decisional diffie-helman (BDDH)* assumption which states that, given a generators $g_0 \in G_0^*$, $g_1 \in G_1^*$ and values $g_0^a, g_1^a, g_0^b, g_1^c$ for random $a, b, c \in \mathbb{Z}_p$, it is computationally hard to distinguish between $e(g_0, g_1)^{abc}$ and a random value in G_T.

4.2.2 Secret Sharing. We use Shamir's secret sharing scheme.

Definition 4.1 (Shamir's Secret Sharing). Let p be a prime. An (n, t, p, s)-Shamir's secret sharing scheme is a randomized algorithm SSS that on input four integers n, t, p, s, where $0 < t \leq n < p$ and $s \in \mathbb{Z}_p$, outputs n shares $s_1, \ldots, s_n \in \mathbb{Z}_p$ such that the following two conditions hold for any set $\{i_1, \ldots, i_\ell\}$:

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

Concretely, Shamir's secret sharing works as follows. Pick $a_1, \ldots, a_{t-1} \leftarrow_{\$} \mathbb{Z}_p$. Let $f(x)$ be the polynomial $s + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_{t-1} \cdot x^{t-1}$. Then s_i is set to be $f(i)$ for all $i \in [n]$.

Additional building blocks can be found in Appendix B.

5 FLEXIBLE THRESHOLD KEY DERIVATION

In this section we introduce the concept of *flexible threshold key-derivation* (FTKD). First, in Sec 5.1 we provide formal definitions. Then, in Sec. 5.2 we provide a construction using bilinear pairing and argue that it satisfies our definitions.

5.1 Definition of FTKD

Now we provide our main definition for *flexible threshold key-derivation* (FTKD). We formalize it as a scheme consisting of both *non-interactive* algorithms and *interactive* protocols.

The notation within DKdf protocol ensures that only one party gets an output.[5] Our key-derivation protocol DKdf works in *three modes*; a party j (called the *client* for this execution) may send three different types of queries: either the whole input (x, y), only the left part x, or the right part y. The recipients (acting as the *servers* in this execution) may perform different computations based on the type of request. Among all (n) parties, if at least a threshold number of parties (t including the client) participate until the completion of the protocol, then the client is able to combine their responses to get either a *partial-key* or a *whole-key* depending on the mode.

Definition 5.1 (Flexible Threshold Key Derivation). A *flexible threshold key derivation* (FTKD) scheme consists of a tuple of algorithms / protocols with the following syntax:

- $\mathsf{Setup}(1^\kappa, n, t) \rightarrow (\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$. Setup is a randomized algorithm that takes the total number of participants n and a threshold t ($\leq n$) as input, and outputs n keys $\llbracket \mathbf{sk} \rrbracket_{[n]} := sk_1, \ldots, sk_n$ and public parameters pp. The i-th secret key sk_i is given to party i. We assume that each of the following algorithm takes pp and security parameter 1^κ as additional inputs implicitly.
- $\mathsf{DKdf}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : \rho, S]) \rightarrow [j : k/\perp]$. DKdf is a protocol that a party j engages with parties in the set S, using which the party j derives a key $k \in \{\mathsf{lk}, \mathsf{rk}, \mathsf{wk}\}$ (or $\perp$ on failure) based on the request $\rho \in \{(x, \texttt{'left'}), (y, \texttt{'right'}), ((x, y), \texttt{'whole'})\}$. We refer to lk as the left-key, rk as the right-key and wk as the whole-key. Only party j receives a private output from DKdf.
- $\mathsf{WKGen}(v, \sigma) =: \mathsf{wk}$. This algorithm deterministically produces a whole-key wk on input $(v, \sigma) \in \{(\mathsf{lk}, y), (\mathsf{rk}, x)\}$.

such that they satisfy the following *consistency* properties. For any $n, t, \kappa \in \mathbb{N}$ such that $t \leq n$, any $(\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$ output by $\mathsf{Setup}(1^\kappa, n, t)$:

[5] These types of protocols are also known as solitary output protocols in literature [37].

Key Consistency: for any set $S \subseteq [n]$ such that $|S| \geq t$, any party $j \in S$, any input (x, y), for any $\sigma \in \{(x, \texttt{'left'}), (y, \texttt{'right'})\}$, if all parties behave honestly, then the following probability is at most $\mathsf{negl}(\kappa)$:[6]

$$\Pr\Big[\mathsf{WKGen}(v, \overline{\sigma}) \neq \mathsf{wk} \;\mid\; [j : \mathsf{wk}] \leftarrow \mathsf{DKdf}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : ((x, y), \texttt{'whole'}), S]); \; [j : v] \leftarrow \mathsf{DKdf}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : (\sigma, S])\Big]$$

Threshold Consistency: for any two sets $S, S' \subseteq [n]$ such that $|S|, |S'| \geq t$, any two parties $j \in S, j' \in S'$, any input (x, y), if all parties behave honestly, then the following probabiliy is at least $1 - \mathsf{negl}(\kappa)$:

$$\Pr\Big[[j' : \mathsf{wk}] \leftarrow \mathsf{DKdf}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j' : ((x, y), \texttt{'whole'}), S']) \;\mid\; [j : \mathsf{wk}] \leftarrow \mathsf{DKdf}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : ((x, y), \texttt{'whole'}), S])\Big]$$

Remark 5.2. Key-consistency only considers queries from the same party; combining with threshold consistency this extends immediately to different parties. Similarly, combining with key consistency, the threshold consistency extends to partial-keys.

Next, we define security of a FTKD scheme. Security is captured by two separate properties: *pseudorandomness* and *correctness*. First note that pseudorandomness is a crucial property for any cryptographic key-derivation. Looking ahead, in our ATSE scheme (Fig. 11) we use the key derived via FTKD to encrypt messages. However, our pseudorandomness property (similar to DPRF pseudorandomness) will guarantee that a key is pseudorandom even when corrupt parties participate in the derivation protocol. We also note that the consistency properties described above are indeed a form of basic correctness when all participants behave honestly. However, when the computation involves malicious parties, the above guarantee may not be sufficient. Therefore we have a separate correctness definition (c.f. Def. 5.6), in that the corrupt parties can behave in arbitrarily malicious way. While correctness can be optional for some use cases, it is indeed a crucial requirement for our application ATSE, as discussed in Remark C.4.

Definition 5.3 (Security of FTKD). We say that an FTKD scheme (as in Def. 5.1) is *secure* against malicious adversaries if it satisfies the pseudorandomness requirement (Def. 5.4). Moreover, an FTKD scheme is said to be *strongly-secure* against malicious adversaries if it satisfies both pseudorandomness *and* correctness (Def. 5.6).

In the definitions that follow, the adversary is given access to various interactive oracles (see Sec. 4.1 for detailed notations). On a legitimate query (as conditioned by the corresponding **require** command) the oracle runs an adequate instance of DKdf protocol. When the client (denoted by j usually) is corrupt, the protocol instance is initiated by the adversary (denoted by $\mathcal{A}$) (and the output is received *only* by the adversary)[7]; the honest servers (usually denoted by $S \setminus C$) are controlled by the oracle. If the client is honest, then the protocol is initiated by the oracle, and in some cases the output is explicitly given to the attacker; in other cases the output is not revealed to the attacker, though the attacker obtains the transcripts of the protocol execution in either case.

Definition 5.4 (Pseudorandomness of FTKD). An FTKD is *pseudorandom* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that

$$\left|\Pr\left[\text{DP-PR}_{\mathcal{A}}(1^\kappa, 0) = 1\right] - \Pr\left[\text{DP-PR}_{\mathcal{A}}(1^\kappa, 1) = 1\right]\right| \leq \mathsf{negl}(\kappa),$$

where the game DP-PR is defined in Figure 2, the key-derivation oracle is defined in 3 and the challenge oracle is defined in 4.

Game DP-PR$_{\mathcal{A}}(1^\kappa, \mathrm{b})$:
- **run** $(\llbracket \mathbf{sk} \rrbracket_n, pp) \leftarrow \mathsf{Setup}(1^\kappa, n, t)$.
- **set** CHAL, ABORT := 0.
- **set** $L_{\mathsf{lk}}, L_{\mathsf{rk}}, L_{\mathsf{wk}} := \emptyset$ **and set** $x^\star, y^\star, z^\star := \perp$.
- **set** $\forall\ (x, y, z)$: $\mathsf{LCT}_x, \mathsf{RCT}_y, \mathsf{WCT}_z := 0$.
- **run** $C \leftarrow \mathcal{A}(pp)$; **require** $C \subset [n]$ **and** $|C| < t$.
- **run** $b' \leftarrow \mathcal{A}^{O^{\text{pr-kd}}, O^{\text{pr-chal}}}(\{sk_i\}_{i \in C})$.
- **if** ABORT $\neq 1$ **then return** b'; **else return uniform** b'.

Figure 2: Pseudorandomness game of an FTKD scheme.

Oracle $O^{\text{pr-kd}}(j, \sigma, S)$:
require $j \in S \ \wedge\ \sigma \in \{(x, \texttt{'left'}), (y, \texttt{'right'}), (z, \texttt{'whole'})\}$.
- **run** $[j : \mathrm{op}] \leftarrow \mathsf{DKdf}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : \sigma, S])$.
- **if** $j \notin C$ **then return** op.
- **if** $j \in C$ **then do** :
  - **if** $\sigma = (x, \texttt{'left'})$ **then do** :
    - **set** $\mathsf{LCT}_x := \mathsf{LCT}_x + |S \setminus C|$.
    - **if** $\mathsf{LCT}_x \geq t - |C|$ **then do** :
      - **if** CHAL $= 1 \ \wedge\ x = x^\star$ **then set** ABORT := 1; **else set** $L_{\mathsf{lk}} := L_{\mathsf{lk}} \cup \{x\}$.
  - **else if** $\sigma = (y, \texttt{'right'})$ **then do** :
    - **set** $\mathsf{RCT}_y := \mathsf{RCT}_y + |S \setminus C|$.
    - **if** $\mathsf{RCT}_y \geq t - |C|$ **then do** :
      - **if** CHAL $= 1 \ \wedge\ y = y^\star$ **then set** ABORT := 1; **else set** $L_{\mathsf{rk}} := L_{\mathsf{rk}} \cup \{y\}$.
  - **else if** $\sigma = (z, \texttt{'whole'})$ **then do** :
    - **set** $\mathsf{WCT}_z := \mathsf{WCT}_z + |S \setminus C|$.
    - **if** $\mathsf{WCT}_z \geq t - |C|$ **then do** :
      - **if** CHAL $= 1 \ \wedge\ z = z^\star$ **set** ABORT := 1; **else set** $L_{\mathsf{wk}} := L_{\mathsf{wk}} \cup \{z\}$.

Figure 3: Key-derivation oracle for DP-PR

Remark 5.5. The flag CHAL is used to distinguish between the pre-challenge and post-challenge queries; CHAL = 1 implies that the challenge oracle was already accessed (also it ensures that the challenge oracle is accessed only once). The counters $\mathsf{LCT}_x, \mathsf{RCT}_y, \mathsf{WCT}_z$ keep track of the total number of honest parties contacted by the attacker on a particular input $x/y/z$ (resp.)–this enables keeping track of whether enough information is acquired by the adversary (that happens when $t - |C|$ parties are contacted on an input) to derive the corresponding key. These counters are initialized to 0 implicitly as there are exponentially many (x, y, z) values. The lists $L_{\mathsf{lk}}, L_{\mathsf{rk}}, L_{\mathsf{wk}}$ contains the left/right/whole inputs respectively for which the attacker already has enough information

[6] Recall from Section 4 that, $\overline{\sigma} \in \{x, y\}$ such that $\overline{\sigma} \neq \sigma$.

[7] This is automatically captured by the notation such as $[j : \mathrm{op}] \leftarrow \mathsf{DKdf}(\llbracket \mathbf{sk} \rrbracket_n, [j : \sigma, S])$, so no explicit statement such as **return** op is needed when j is corrupt.

Oracle $\mathcal{O}^{\text{pr-chal}}(j^\star, z^\star, S^\star)$:
require $j^\star \in S^\star \setminus C$ **and** $|S^\star| \geq t$ **and** CHAL = 0.
- **set** CHAL := 1.
- **set** $(x^\star, y^\star) := z^\star$.
- **if** $(z^\star \in L_{\text{wk}})$ **or** $(y^\star \in L_{\text{rk}})$ **or** $(x^\star \in L_{\text{lk}})$ **then** ABORT := 1;
  else run $[j^\star : \text{wk}^\star] \leftarrow \text{DKdf}([\![\mathbf{sk}]\!]_{[n]}, [j : \sigma, S^\star])$ **and do**:
  - **if** $(\text{wk}^\star = \perp)$ **then return** $\perp$;
    else do:
    - **if** $b = 0$ **then return** $\text{wk}^\star$.
    - **if** $b = 1$ **then return uniform** $\text{wk}^\star$.

Figure 4: Challenge oracle for DP-PR.

Oracle $\mathcal{O}^{\text{cr-kd}}(j, \sigma, S)$:
require $j \in S \ \wedge \ \sigma \in \{(x, \texttt{'left'}), (y, \texttt{'right'}), (z, \texttt{'whole'})\}$.
- **run** $[j : \text{op}] \leftarrow \text{DKdf}([\![\mathbf{sk}]\!]_{[n]}, [j : \sigma, S])$.
- **if** $j \notin C$ **then return** op.

Figure 6: Key-derivation oracle for DP-Correct

Oracle $\mathcal{O}^{\text{cr-chal}}(j^\star, \text{Type}, z^\star, S^\star)$:
require OUT = 0 **and** CHAL = 0 **and** Type $\in \{$'left', 'right', 'whole'$\}$ **and** $j^\star \in S^\star \setminus C$ **and** $|S^\star| \geq t$.
- **set** CHAL := 1, and **set** $(x^\star, y^\star) := z^\star$.
- **if** Type = 'left' **then do**:
  run $[j^\star : \text{lk}] \leftarrow \text{DKdf}([\![\mathbf{sk}]\!]_{[n]}, [j^\star : (x^\star, \texttt{'left'}), S^\star])$.
  if $\text{lk} \neq \perp$ **then set** $\text{wk}^\star := \text{WKGen}(\text{lk}, y^\star)$.
- **else if** Type = 'right' **then do**:
  run $[j^\star : \text{rk}] \leftarrow \text{DKdf}([\![\mathbf{sk}]\!]_{[n]}, [j^\star : (y^\star, \texttt{'right'}), S^\star])$.
  if $\text{rk} \neq \perp$ **then set** $\text{wk}^\star := \text{WKGen}(x^\star, \text{rk})$.
- **else if** Type = 'whole' **then do** :
  run $[j^\star : \text{wk}^\star] \leftarrow \text{DKdf}([\![\mathbf{sk}]\!]_{[n]}, [j^\star : (z^\star, \texttt{'whole'}), S^\star])$.
- **run** $\text{wk}^\dagger \leftarrow \text{DKdf}^\dagger([\![\mathbf{sk}]\!]_{[n]}, [j : (z^\star, \texttt{'whole'}), S^\star])$.
- **if** $\text{wk}^\star = \perp$ **then set** OUT := 0; **else set** OUT := $(\text{wk}^\star \neq \text{wk}^\dagger)$

Figure 7: Challenge oracle for DP-Correct

to compute the output and hence if a challenge query is made on any such input (present in one of the lists) the experiment is aborted (within the challenge oracle, by setting the flag ABORT = 1.)

Definition 5.6 (Correctness of FTKD). An FTKD is *correct* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that the DP-Correct$_{\mathcal{A}}$ game outputs 1 with probability at most $\text{negl}(\kappa)$. The game DP-Correct is given in Figure 5, the key-derivation oracle is defined in Figure 6 and the challenge oracle is defined in Figure 7.

Game DP-Correct$_{\mathcal{A}}(1^\kappa)$:
- **run** $([\![\mathbf{sk}]\!]_n, pp) \leftarrow \text{Setup}(1^\kappa, n, t)$.
- **set** CHAL, OUT := 0.
- **run** $C \leftarrow \mathcal{A}(pp)$ **and require** $C \subset [n]$ **and** $|C| < t$.
- **run** $\mathcal{A}^{\mathcal{O}^{\text{cr-kd}}, \mathcal{O}^{\text{cr-comp}}}(\{sk_i\}_{i \in C})$.
- **return** OUT.

Figure 5: Correctness game of an FTKD scheme.

Remark 5.7. The key-derivation oracle above is much simpler than the one in DP-PR as no book-keeping is required to prevent a trivial win. However, the challenge oracle gets more complex, because we allow the attacker to use *any* of the three "modes of interaction" available. As guaranteed by the consistency property in Def. 5.1, when no corruption is present then all three modes generate the same whole-key for the same input. However, a malicious server may decide to respond differently for different modes (even when the same input is used) potentially leading to different keys. Our correctness definition guarantees that such scenario can never happen. In particular, if correctness holds, then the attacker can not enforce an honest client to derive different keys even when different modes of key-derivation are used. Any such attempt would invoke a failure (that is $\perp$) with overwhelming probability. This is formalized by the (global) flag OUT, which is set to 1 inside the challenge oracle only when derived whole-key $\text{wk}^\star$ is equal to neither $\perp$, nor $\text{wk}^\dagger$ which is the value obtained from an honest execution (denoted by $\text{DKdf}^\dagger$ as explained in Sec. 4.1).

5.2 Our FTKD construction

Figure 8 specifies our FTKD construction. We require a bilinear pairing:[8] $e : G_0 \times G_1 \to G_T$ and two independent hash functions $\mathcal{H}_0, \mathcal{H}_1$ such that $\mathcal{H}_b : \{0,1\}^* \to G_b$ for any $b \in \{0,1\}$. In the setup phase, a master secret-key $sk \leftarrow_\$ \mathbb{Z}_p$ is sampled which is then secret-shared using t-out-of-n Shamir's secret sharing denoted $[\![\mathbf{sk}]\!]_{[n]} := \{sk_1, \ldots, sk_n\}$. Each party-$i$ obtains a key-share sk_i. For an input (x, y) the left, right and the whole-keys are given by $\mathcal{H}_0(x)^{sk}, \mathcal{H}_1(y)^{sk}$ and $e(\mathcal{H}_0(x), \mathcal{H}_1(y))^{sk}$ respectively. Clearly, given a partial-key $\mathcal{H}_0(x)^{sk}$ (resp. $\mathcal{H}_1(y)^{sk}$) and the other part of the input, y (resp. x) one can locally compute the whole-key using pairing. For any key-derivation query, depending on its type, a server performs an exponentiation on an appropriate value $w \in \{\mathcal{H}_0(x), \mathcal{H}_1(y), e(\mathcal{H}_0(x)), \mathcal{H}_1(y)\}$ with its own share sk_i, and returns that to the client. On receiving sufficiently many responses, the client combines any t (including the value computed from its own share) responses to compute the partial or whole-key using Lagrange interpolation in the exponent.

Remark 5.8 (Special structure of our DKdf protocol: Simple FTKD). We observe our FTKD scheme has a special property: in the first round of $\text{DKdf}([\![\mathbf{sk}]\!]_{[n]}, [j : \rho, S])$ the client j sends over part of its input, ρ to each server in S. Any FTKD scheme with a DKdf protocol with this specific simple structure is referred to as a *simple* FTKD scheme. Our ATSE construction requires this.

Theorem 5.9. *The construction, presented in Figure 8 is a secure FTKD construction under the BDDH assumption over the map $e(G_0, G_1) \to G_T$ in the programmable random oracle model.*

Proof. The threshold and key consistency properties follow immediately from the correctness of Shamir's Secret Sharing and bilinear pairing respectively.

Pseudorandomness. To see the pseudorandomness, first note that our FTKD construction is a distributed variant of the left/right constrained PRF construction of Boneh and Waters (BW [20]) in the random oracle model. Due to the structural similarity (in particular the "key-homomorphic" property) with the distributed PRF of Naor, Pinkas and Reingold [45], which is the DDH-based DPRF construction used in DiSE [13], it is possible to use it in a distributed manner

[8] We assume the asymmetric variant which subsumes the other variants, but our construction is compatible with other variants as well.

Ingredients

- Let G_0, G_1, and G_T be multiplicative cyclic groups of prime order p such that there exists a bilinear pairing e: $G_0 \times G_1 \to G_T$ that is efficiently computable and non-degenerate; we let $g_0 \in G_0$ and $g_1 \in G_1$ be generators of G_0 and G_1 respectively.
- Let $\mathcal{H}_0 : \{0,1\}^* \to G_0$ and $\mathcal{H}_1 : \{0,1\}^* \to G_1$ be hash functions modeled as random oracles.
- Let SSS be Shamir's secret sharing scheme.

Our FTKD Construction

- $\mathsf{Setup}(1^\kappa, n, t) \to (\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$. Sample $sk \leftarrow_\$ \mathbb{Z}_p$; run $\llbracket \mathbf{sk} \rrbracket_n \leftarrow \mathsf{SSS}(n, t, p, s)$. Set $pp := (p, g_0, G_0, g_1, G_1, G_T, e, \mathcal{H}_0, \mathcal{H}_1)$.
- $\mathsf{DKdf}(\llbracket \mathbf{sk} \rrbracket_S, [j : \rho, S]) \to [j : k/\perp]$. This is a two round protocol as described below:

Round-1. Party j sends ρ to each party $i \in S$.

Round-2. Each party $i \in S$ sends back h_i to j such that $h_i := w^{sk_i}$ where $w := \begin{cases} \mathcal{H}_0(x) \text{ if } \rho = (x, \texttt{'left'}) \\ \mathcal{H}_1(y) \text{ if } \rho = (y, \texttt{'right'}) \\ e(\mathcal{H}_0(x), \mathcal{H}_1(y)) \text{ if } \rho = ((x,y), \texttt{'whole'}) \end{cases}$

Finalize. Party j, if receives at least $t-1$ responses from parties in S then outputs $k := \prod_{i \in S'} h_i^{\lambda_{0,i,S'}}$ where $S' \subseteq S \cup \{j\}$ is of size t; otherwise outputs $\perp$.

- $\mathsf{WKGen}(v, \sigma) =: \mathsf{wk}$. Compute $\mathsf{wk} := e(v, \mathcal{H}_b(\sigma))$ where $b := 0$ if $\sigma = x$, else 1 if $\sigma = y$

Figure 8: Our Flexible Threshold Key Derivation Scheme

as done here. The pseudorandomness of our construction is loosely based on the arguments provided in BW [20] (Theorem 3.2) and DiSE [13] (Theorem 8.1). We defer the details to Appendix F.1 □

The construction of strongly secure FTKD (see Fig. 15) is achieved by techniques analogous to strongly-secure DiSE DPRF, namely using efficient ZK proofs, and is deferred to Appendix A.

6 GROUP COMMITMENTS

We introduce the notion of *group commitments* in this section. We construct this primitive based on Merkle-tree.

Definition 6.1 (Group Commitments). A *group commitment scheme* consists of a tuple of PPT algorithms, (GSetup, GCommit, CardVer, GVer) with the following syntax:

- $\mathsf{GSetup}(1^\kappa) \to pp$: The setup algorithm, on input the security parameter, outputs the public parameters.
- $\mathsf{GCommit}(pp, \mathbf{m}) \to (\gamma, \mathbf{q}, \mathbf{p})$: The commit algorithm takes a message vector as input and returns a group commitment string γ, a unique commitment vector $\mathbf{q}$, an opening vector $\mathbf{p}$.
- $\mathsf{CardVer}(pp, (\gamma, N)) =: 0/1$: The cardinality verification algorithm takes a commitment and an integer and outputs 1 if and only if the group-commitment's cardinality (the number of messages in the group used to produce γ) correctly verifies against N.
- $\mathsf{GVer}(pp, (\gamma, q), (m, p)) =: 0/1$: The verification algorithm takes a pair of group and unique commitments, a message-opening pair, and outputs a bit signifying the validity (1 if valid and 0 otherwise) of the message with respect to the commitment and the opening.

such that the following properties hold for any $\kappa \in \mathbb{N}$ and any $pp \leftarrow \mathsf{GSetup}(1^\kappa)$:

Game $\mathsf{Hide}_{\mathcal{A}}(1^\kappa, b)$:

- **set** $I := \emptyset$.
- **run** $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}(pp)$; **require** $|\mathbf{m}_0| = |\mathbf{m}_1|$ **and** $\exists\ i$ **such that** $\mathbf{m}_0[i] \neq \mathbf{m}_1[i]$.
- **for** $i = 1 \to |\mathbf{m}_0|$: **if** $m_0[i] = m_1[i]$ **then set** $I := I \cup \{i\}$.
- **run** $(\gamma, \mathbf{q}, \mathbf{p}) \leftarrow \mathsf{GCommit}(pp, \mathbf{m}_b)$.
- **run** $b' \leftarrow \mathcal{A}^{O^{\mathsf{open}}}(\gamma, \mathbf{q})$.
- **if** $b = b'$ **then return** 1; **else return** 0

Figure 9: Hiding game for group-commitments

Oracle $O^{\mathsf{open}}(i^\star)$: **if** $i^\star \in I$ **then return** $p_{i^\star}$

Figure 10: The opening oracle for game Hide.

Correctness: for any $\mathbf{m} = (m_1, \ldots, m_N)$ and any $i \in [N]$ we have:

$$\Pr[\mathsf{GVer}(pp, (\gamma, q_i), (m_i, p_i))) = 1 \mid (\gamma, \mathbf{q}, \mathbf{p}) \leftarrow \mathsf{GCommit}(pp, \mathbf{m})] = 1$$

where the randomness is over GCommit.

Compactness: for any $\mathbf{m} = (m_1, \ldots, m_N)$ and any $i \in [N]$, $|\gamma|$ is independent of N and $|q_i| \propto \log(N)$ where $(\gamma, \mathbf{q}, \mathbf{p}) \leftarrow \mathsf{GCommit}(pp, \mathbf{m})$.

Binding: for any PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that:

$$\Pr\Big[((m,p) \neq (m',p')); \mathsf{GVer}(pp, ((\gamma, q), (m,p))) = 1; \mathsf{GVer}(pp, ((\gamma, q), (m',p'))) = 1 \mid (\gamma, q, (m,p), (m',p')) \leftarrow \mathcal{A}(pp)\Big] \leq \mathsf{negl}(\kappa)$$

Cardinality Binding: for any PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that:

$$\Pr\Big[\{\mathsf{GVer}(pp, (\gamma, q_i), (m_i, p_i)) = 1\}_{i \in |\mathbf{m}|}; |\mathbf{m}| > N; \mathsf{CardVer}(pp, (\gamma, N)) = 1 \mid (\gamma, N, (\mathbf{m}, \mathbf{q}, \mathbf{p})) \leftarrow \mathcal{A}(pp)\Big] \leq \mathsf{negl}(\kappa)$$

Hiding: for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:

$$|\Pr[\mathsf{Hide}_{\mathcal{A}}(1^\kappa, b) = 1] - 1/2| \leq \mathsf{negl}(\kappa)$$

where the security game Hide is defined in Figure 9, and the opening oracle O^{open} is defined in Figure 10.

Remark 6.2. A group commitment scheme has a standard binding property, and a *fine-grained* hiding property, in that the commitment pair $(\gamma, \mathbf{q})$ of a set of messages $\mathbf{m}$ semantically hides an $m \in \mathbf{m}$ even when the openings p' of other messages ($m' \neq m$, but $m' \in \mathbf{m}$) in the same group are given. This is captured by the opening oracle O^{open}, which requires indexes $i^\star$ for which the message $m_{i^\star}$ is exactly the same in the two groups $\mathbf{m}_0$ and $\mathbf{m}_1$, to prevent trivial wins. Furthermore, it has a cardinality binding property, that prevents an attacker from lying about the cardinality of the vector[9]. This property only makes sense when compactness holds and the verification algorithm does not get a unique-commitment as input.

We defer our Merkle-tree based instantiation to Appendix C.

[9]Without this property one may carry out an attack by *honestly* using 100 messages to generate γ, but then *maliciously* claiming that she used only 10 messages. Looking ahead, this would make our one-more style authenticity definition to fail.

7 AMORTIZED THRESHOLD SYMMETRIC ENCRYPTION (ATSE)

We now present *amortized threshold symmetric encryption* (ATSE). We first provide the formal definition, and then present our construction based on FTKD, and prove that it satisfies our definition.

7.1 Definition of ATSE

In a t out of n ATSE scheme all parties obtain their corresponding key-shares in the setup phase. Any party, who wants to encrypt a tuple of messages $\mathbf{m}$, contacts at least $t-1$ other parties to interactively generate a tuple of ciphertexts $\mathbf{c}$. In this particular execution, the encrypting party is often called the *client*, whereas the helping parties are called the *servers*. Later, *any* party (possibly different from the previous client) may want to decrypt a single ciphertext c from the tuple $\mathbf{c}$. It can do so by contacting *any* $t-1$ (possibly different from the previous servers) other parties.

We remark that our formalization follows the definitional framework of *threshold symmetric encryption* (TSE) definition (Def. 6.1 in DiSE ePrint version [14]) of DiSE. The only difference is that we allow encryption of a group messages together, whereas DiSE supports encryption of one message at a time.[10]

Definition 7.1 (Amortised Threshold Symmetric Encryption (ATSE)). An *amortised threshold encryption* scheme (ATSE) is given by a triple of algorithms and protocols (Setup, DGEnc, DKdf) satisfying the consistency property described below.

- $\mathsf{Setup}(1^\kappa, n, t) \rightarrow (\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$: Setup is a randomized algorithm that outputs n secret keys $sk_1, \ldots, sk_n$ and public parameters pp. The i-th secret key sk_i is given to party i. All algorithms / protocols below take pp as an implicit input.
- $\mathsf{DGEnc}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : \mathbf{m}, S]) \rightarrow [j : \mathbf{c}/\perp]$: DGEnc denotes a distributed *group-encryption* protocol through which a party j encrypts any vector of messages $\mathbf{m} = (m_1, m_2 \ldots)$ to produce a vector of ciphertexts $\mathbf{c} = (c_1, c_2, \ldots)$ or $\perp$, when it fails.
- $\mathsf{DistDec}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : c, S]) \rightarrow [j : m/\perp]$: DistDec is a distributed protocol through which a party j decrypts a single ciphertext c with the help of parties in a set S. At the end of the protocol, j outputs a message (or $\perp$ to denote failure).

Consistency. For any $\kappa, n, t, N \in \mathbb{N}$ such that $t \leq n$, all $(\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$ output by $\mathsf{Setup}(1^\kappa, n, t)$, for any sequence of messages $\mathbf{m} = m_1 \ldots, m_N$, any $i \in [N]$, two sets $S, S' \subset [n]$ such that $|S|, |S'| \geq t$, and any two parties $j \in S, j' \in S'$, if all the parties behave honestly, then there exists a negligible function negl for which the following probability is at least $1 - \mathsf{negl}(\kappa)$.

$$\Pr\Big[[j' : m_i] \leftarrow \mathsf{DistDec}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j' : c_i, S']) \;\; | \;\; \mathbf{c} \leftarrow \mathsf{DGEnc}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : \mathbf{m}, S]) \Big]$$

where the probability is over the random coin tosses of the parties involved in DGEnc and DistDec.

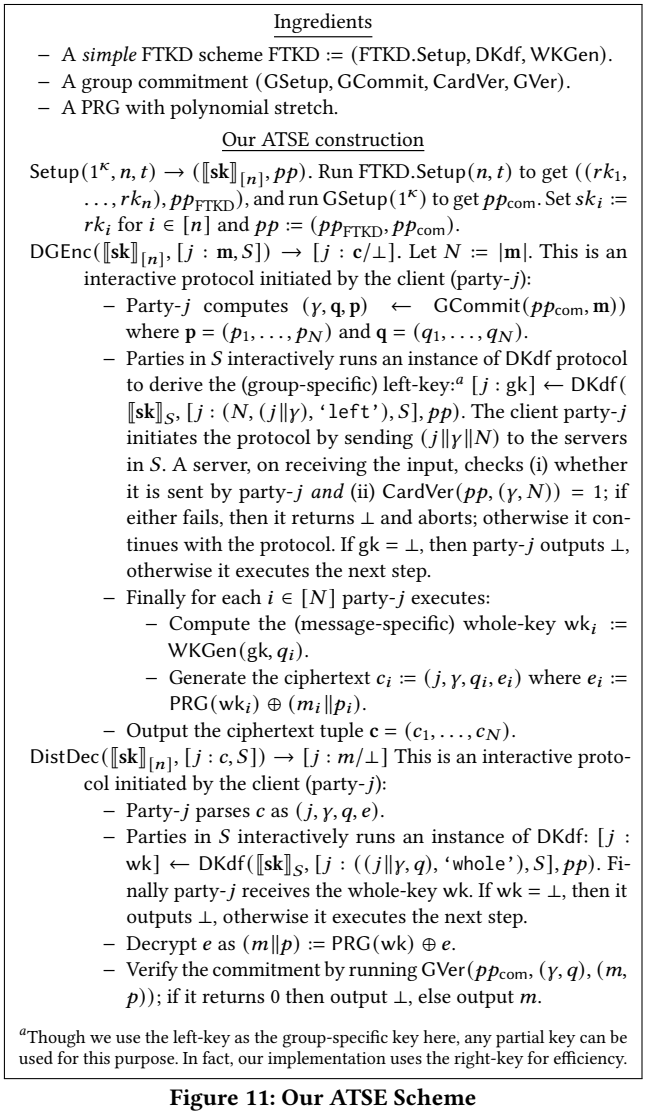
We now define the security of an ATSE scheme. Inspired by DiSE TSE [14], the security of an ATSE scheme is captured by three different properties, *correctness*, *message-privacy* and *authenticity*, of which correctness and authenticity have stronger versions.

Definition 7.2 (Security of ATSE). We say that a ATSE scheme is *(strongly)-secure* against malicious adversaries if it satisfies the (strong)-correctness (Def. C.2), message privacy (Def. C.5) and (strong)-authenticity (Def. C.8) requirements.

For space limitation we defer the formal definitions of the specific security properties to Appendix **??**.

7.2 Our ATSE Construction

In this section, we put forward our ATSE construction in Figure 11 based on the main two ingredients: (i) a *simple* FTKD scheme (as per Remark 5.8) and (ii) a group commitment scheme. Additionally we require a pseudorandom generator of polynomial stretch.



Ingredients

- A *simple* FTKD scheme FTKD := (FTKD.Setup, DKdf, WKGen).
- A group commitment (GSetup, GCommit, CardVer, GVer).
- A PRG with polynomial stretch.

Our ATSE construction

$\mathsf{Setup}(1^\kappa, n, t) \rightarrow (\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$. Run FTKD.Setup$(n, t)$ to get $((rk_1, \ldots, rk_n), pp_{\text{FTKD}})$, and run GSetup$(1^\kappa)$ to get pp_{com}. Set $sk_i := rk_i$ for $i \in [n]$ and $pp := (pp_{\text{FTKD}}, pp_{\text{com}})$.

$\mathsf{DGEnc}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : \mathbf{m}, S]) \rightarrow [j : \mathbf{c}/\perp]$. Let $N := |\mathbf{m}|$. This is an interactive protocol initiated by the client (party-j):

- Party-j computes $(\gamma, \mathbf{q}, \mathbf{p}) \leftarrow \mathsf{GCommit}(pp_{\text{com}}, \mathbf{m}))$ where $\mathbf{p} = (p_1, \ldots, p_N)$ and $\mathbf{q} = (q_1, \ldots, q_N)$.
- Parties in S interactively runs an instance of DKdf protocol to derive the (group-specific) left-key:[a] $[j : \mathsf{gk}] \leftarrow \mathsf{DKdf}(\llbracket \mathbf{sk} \rrbracket_S, [j : (N, (j \| \gamma), \texttt{'left'}), S], pp)$. The client party-$j$ initiates the protocol by sending $(j \| \gamma \| N)$ to the servers in S. A server, on receiving the input, checks (i) whether it is sent by party-j *and* (ii) $\mathsf{CardVer}(pp, (\gamma, N)) = 1$; if either fails, then it returns $\perp$ and aborts; otherwise it continues with the protocol. If $\mathsf{gk} = \perp$, then party-j outputs $\perp$, otherwise it executes the next step.
- Finally for each $i \in [N]$ party-j executes:
  - Compute the (message-specific) whole-key $\mathsf{wk}_i := \mathsf{WKGen}(\mathsf{gk}, q_i)$.
  - Generate the ciphertext $c_i := (j, \gamma, q_i, e_i)$ where $e_i := \mathsf{PRG}(\mathsf{wk}_i) \oplus (m_i \| p_i)$.
- Output the ciphertext tuple $\mathbf{c} = (c_1, \ldots, c_N)$.

$\mathsf{DistDec}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : c, S]) \rightarrow [j : m/\perp]$ This is an interactive protocol initiated by the client (party-j):

- Party-j parses c as (j, γ, q, e).
- Parties in S interactively runs an instance of DKdf: $[j : \mathsf{wk}] \leftarrow \mathsf{DKdf}(\llbracket \mathbf{sk} \rrbracket_S, [j : ((j \| \gamma, q), \texttt{'whole'}), S], pp)$. Finally party-$j$ receives the whole-key wk. If $\mathsf{wk} = \perp$, then it outputs $\perp$, otherwise it executes the next step.
- Decrypt e as $(m \| p) := \mathsf{PRG}(\mathsf{wk}) \oplus e$.
- Verify the commitment by running $\mathsf{GVer}(pp_{\text{com}}, (\gamma, q), (m, p))$; if it returns 0 then output $\perp$, else output m.

[a] Though we use the left-key as the group-specific key here, any partial key can be used for this purpose. In fact, our implementation uses the right-key for efficiency.

Figure 11: Our ATSE Scheme

Theorem 7.3. *Our ATSE scheme described in Figure 11 is (strongly)-secure if the underlying simple FTKD is (strongly-)secure.*

[10] This is, nevertheless, a syntactic difference as one may just run DiSE encryption protocol on a group of messages, assuming that to be a single message. The security property that follows provide a semantic distinction.

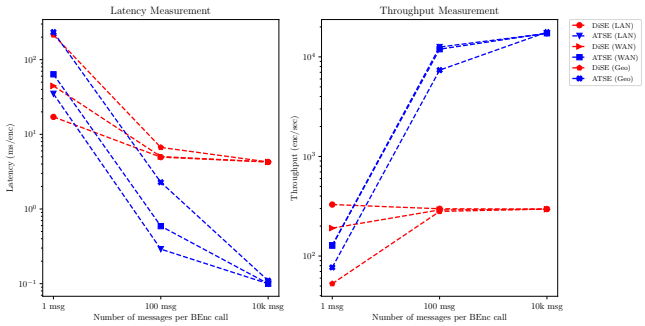


Figure 12: Effect of amortized interaction: $n = 24$, $t = 22$

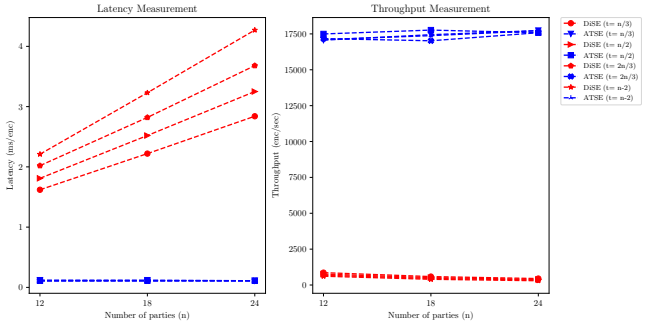


Figure 13: Effect of t and n: 10K messages, Geo deployment

Proof. The consistency property is straightforward to see given consistency of the underlying simple FTKD scheme. The other properties, namely message-privacy, (strong-)correctness and (strong-)authenticity follow arguments similar to that of DiSE and are formally provided in the full version [24]. □

8 EXPERIMENTAL EVALUATION

We implement our strongly secure ATSE scheme in Go. We use the 256-bit Barretto-Naehrig curves that support the Optimal Ate pairings as described in [44] (implementation from [1]). Observe that the construction for ATSE.DGEnc is embarrassingly parallel as we can concurrently compute the pairings for each input message and concurrently verify the zero-knowledge proofs from each server, and we make full use of such parallelism opportunities.

We compare throughput and latency metrics of ATSE with the strongly secure version of DiSE from [13] using the DDH-based PRF [45] (as the AES version requires a more expensive symmetric key NIZK [23, 35]). We evaluate both schemes with varying parameters for t and n, and in three network configurations for the servers: 1) LAN: 2 ms round-trip latency, 2) WAN: 30 ms latency, and 3) GEO: geo-distributed servers with 200 ms latency. We also vary the number of messages (1/100/10000) that are provided to bulk encryption, with the hypothesis that increasing messages further highlights the performance benefits of the amortized interaction. We only discuss bulk encryption here because decryption operates similarly in ATSE and DiSE.

For fair comparison, and to focus on the key benefits of ATSE, we implement the following optimizations in the DiSE scheme. Since DiSE only supports encryption of 1 message at a time, we implement a batched version which coalesces all messages into one server-bound request and have each server return the PRF evaluation of all messages within a single response (to avoid repeated round-trips when encrypting many messages). We even implement server-side parallelism when computing PRF over multiple messages in a batch.

Benchmarks were run using two server-grade machines (one for the client application, and one for the n server processes holding the key shares), each equipped with a 16-core Intel Xeon E5-2640 CPU @ 2.6 Ghz and 64 GB DDR4 RAM. We use Figure 36 and Figure 37 in Appendix E to report the latency and throughput for LAN and WAN settings, respectively, and Figure 14 here to report the geo-distributed setting — for each setting, we vary the threshold t, the number of parties n, and the number of messages. Figure 12 and Figure 13 illustrate the relevant trends, and we discuss them below.

Latency. We record latency as the time period between invoking encryption or decryption until the result is ready — both encryption and decryption take similar time in our scheme. When multiple messages are provided as input to DGEnc, we divide the total time by the number of messages to report the latency per message. The absolute numbers for the latency depends on the number of CPU cores available to the encryptor client, as the operation is compute-bound, so we focus our discussion to the relative trends.

When encrypting a single message, ATSE has higher latency by a few millisecs because of the additional bilinear pairing compared to DiSE — the round-trip network latency followed by the computation of DPRF combination and NIZK proof verification is equivalent in both schemes when handling 1 message. However, we observe that ATSE has significantly lower latency (0.09 msecs / encryption) when encrypting 10K messages together, providing between 12x-42x improvement over DiSE depending on the parameters n and t. This is because DiSE requires DPRF combination and NIZK proof verification for each message, whereas this computation is amortized over a group of messages in ATSE — even though ATSE requires a bilinear pairing for each message, that time is dwarfed by the added computation in DiSE, for even small values of n and t.

We find that t and n has negligible impact on ATSE's latency (for 100 or 10K messages) as the increased computation — a client must verify at least t NIZK proofs (each $O(1)$ group operations) and perform $O(t)$ group operations for DPRF combination (Lagrange interpolation in the exponent) — is only performed once, whereas DiSE latency suffers significantly with higher t and n. For that reason, the network latency also has negligible impact when encrypting multiple messages. In practice, this feature incentivizes deployments with larger t and n, and across many data centers, thus raising the bar for an attack while also increasing availability.

Throughput. We report throughput to be the average number of operations per second, which we measure by launching multiple clients in parallel, who send requests to a server pool. ATSE attains throughputs over 23K encryptions / sec, an order of magnitude improvement over DiSE, for the same reasons as latency. We see between 10x-30x improvement, as DiSE has lower throughput with higher values of t and n; they have negligible impact on ATSE's throughput when encrypting multiple messages, as the NIZK verification and Lagrange interpolation is performed once. Moreover, network latency has negligible impact on ATSE's throughput because the interaction only occurs once per group of messages.

t	n	Latency (ms/enc)						Throughput (enc/s)					
		DiSE 1 msg	ATSE 1 msg	DiSE 100 msg	ATSE 100 msg	DiSE 10000 msg	ATSE 10000 msg	DiSE 1 msg	ATSE 1 msg	DiSE 100 msg	ATSE 100 msg	DiSE 10000 msg	ATSE 10000 msg
$n/3$	6	205.92	212.97	3.17	2.21	1.04	0.09	57.39	54.76	1398.27	4880.45	1740.27	22543.26
	12	207.45	217.36	3.78	2.22	1.62	0.09	55.79	52.30	798.46	4673.12	861.67	22492.51
	18	209.00	218.75	4.41	2.21	2.22	0.10	53.79	50.22	548.47	4705.85	587.75	22460.39
	24	210.08	220.43	4.99	2.21	2.84	0.10	55.25	47.89	422.56	4645.98	449.92	22974.32
$n/2$	6	206.29	213.60	3.33	2.22	1.12	0.09	57.39	54.57	1274.24	4672.32	1627.96	22496.51
	12	208.26	216.11	4.08	2.22	1.81	0.09	56.26	52.08	691.33	4743.46	762.13	22963.56
	18	210.02	219.36	4.70	2.22	2.52	0.10	54.63	50.19	486.55	4796.76	517.15	22999.04
	24	210.62	225.35	5.47	2.23	3.25	0.09	54.85	48.39	372.41	4624.42	393.34	22800.52
$2n/3$	6	207.70	214.44	3.47	2.23	1.24	0.09	56.83	53.98	1129.56	4714.47	1415.88	22687.60
	12	209.04	219.38	4.23	2.23	2.02	0.10	56.13	51.17	620.78	4600.97	682.50	22537.69
	18	210.89	224.22	5.12	2.23	2.82	0.10	54.49	48.84	431.65	4537.45	455.21	22665.10
	24	212.34	227.41	6.06	2.25	3.68	0.09	53.36	46.84	325.83	4557.52	347.59	22784.53
$n-2$	6	207.70	216.27	3.47	2.22	1.24	0.09	56.83	53.52	1129.56	4754.78	1415.88	22939.19
	12	209.85	221.50	4.57	2.24	2.21	0.10	55.48	51.26	568.55	4644.43	611.72	22787.98
	18	212.12	227.34	5.65	2.24	3.23	0.10	53.32	47.67	382.09	4610.05	397.75	22763.13
	24	214.44	231.95	6.68	2.27	4.27	0.10	52.76	45.40	280.52	4486.86	296.88	22425.52
2	6	205.92	213.12	3.17	2.22	1.04	0.09	57.39	54.87	1398.27	4696.05	1740.27	22721.16
	12	206.66	216.00	3.53	2.21	1.42	0.10	56.20	52.84	897.56	4830.98	1024.26	22614.18
	18	207.25	215.56	3.87	2.20	1.83	0.09	55.10	50.83	665.82	4719.54	720.64	23048.52
	24	207.17	216.12	4.24	2.20	2.26	0.10	56.41	49.17	529.65	4745.87	578.95	23170.50

Figure 14: Encryption performance metrics averaging 10 repeated trials of 32 bytes messages in the geo-distributed setting.

Communication. Note that ATSE incurs constant communication overhead in the number of messages. For each interaction (i.e., for each invocation of DGEnc), ignoring the underlying TLS channel's overheads, the client receives 132^*t bytes, comprising one 256-bit group element and three 256-bit scalar values. In contrast, DiSE receives 132^*t^*k bytes, where k is the number of messages. For $k = 10^4$, that constitutes 4 orders of magnitude reduction in bandwidth.

Key Size. Each server is given a key share output by the underlying DPRF scheme, which is 64 bytes (2 32-byte field elements).

Ciphertext Expansion. Since each ciphertext includes a Merkle proof (root-to-leaf path), its size is a function of N: the number of messages being encrypted as a group. For a message of size $|m|$ bytes, the corresponding ciphertext occupies $|m|+40+32\lceil log_2(N)\rceil$ bytes, consisting of 8-byte client id, 32-byte commitment, and log-size Merkle proof (containing a sequence of SHA-256 values). In addition, we can compress the stored representation by storing the entire Merkle tree separately, rather than storing path-by-path, giving us an aggregated ciphertext size of $N * |m| + 40 + 32 * 2^{\lceil log_2(N)\rceil+1}$, for a group of N messages.

REFERENCES

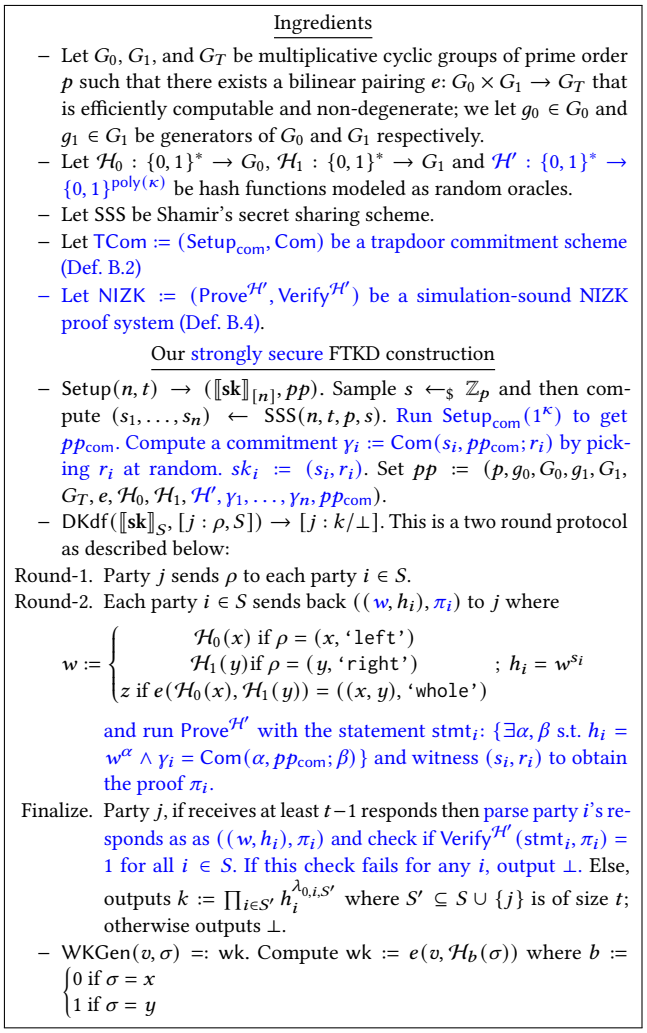
[1] Advanced crypto library for the Go language. https://github.com/dedis/kyber.
[2] Coinbase custody. custody.coinbase.com/. Use of secret sharing described in [?].
[3] Cybersource Payment Platform and Fraud management. https://www.cybersource.com/en-us.html.
[4] Dyadic Security. https://www.dyadicsec.com.
[5] Fact Sheet - Visa. [Online; posted June-2018].
[6] Introduction to Pairing-Based Cryptography. http://cseweb.ucsd.edu/~mihir/cse208-06/main.pdf.
[7] NIST tcg. csrc.nist.gov/Projects/threshold-cryptography.
[8] Unbound Tech. www.unboundtech.com/. Use of MPC mentioned in [?].
[9] Vault Seal. www.vaultproject.io/docs/concepts/seal.html.
[10] Visa. https://usa.visa.com/.
[11] S. Agrawal, S. Badrinarayanan, P. Mohassel, P. Mukherjee, and S. Patranabis. BETA: biometric-enabled threshold authentication. In J. A. Garay, editor, *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II*, volume 12711 of *Lecture Notes in Computer Science*, pages 290–318. Springer, 2021.
[12] S. Agrawal, P. Miao, P. Mohassel, and P. Mukherjee. PASTA: PASsword-based threshold authentication. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 2042–2059. ACM Press, Oct. 2018.
[13] S. Agrawal, P. Mohassel, P. Mukherjee, and P. Rindal. DiSE: Distributed symmetric-key encryption. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 1993–2010. ACM Press, Oct. 2018.
[14] S. Agrawal, P. Mohassel, P. Mukherjee, and P. Rindal. DiSE: Distributed symmetric-key encryption. Cryptology ePrint Archive, Report 2018/727, 2018. https://eprint.iacr.org/2018/727.
[15] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, Apr. 2015.
[16] N. Attrapadung, T. Matsuda, R. Nishimaki, S. Yamada, and T. Yamakawa. Constrained PRFs for NC^1 in traditional groups. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 543–574. Springer, Heidelberg, Aug. 2018.
[17] C. Baum, T. K. Frederiksen, J. Hesse, A. Lehmann, and A. Yanai. PESTO: proactively secure distributed single sign-on, or how to trust a hacked server. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 587–606. IEEE, 2020.
[18] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, Heidelberg, Dec. 2000.
[19] D. Boneh, K. Lewi, H. W. Montgomery, and A. Raghunathan. Key homomorphic PRFs and their applications. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Heidelberg, Aug. 2013.
[20] D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, Dec. 2013.
[21] R. Canetti and S. Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 90–106. Springer, Heidelberg, May 1999.
[22] D. Catalano and D. Fiore. Vector commitments and their applications. In K. Kurosawa and G. Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Heidelberg, Feb. / Mar. 2013.
[23] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1825–1842, 2017.
[24] M. Christodorescu, S. Gaddam, P. Mukherjee, and R. Sinha. Amortized threshold symmetric-key encryption. Cryptology ePrint Archive, Report 2021/1176, 2021. https://eprint.iacr.org/2021/1176.
[25] I. Damgård and M. Keller. Secure multiparty AES. In R. Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 367–374. Springer, Heidelberg, Jan. 2010.
[26] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *26th ACM STOC*, pages 522–533. ACM Press, May 1994.
[27] C. Delerablée and D. Pointcheval. Dynamic threshold public-key encryption. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 317–334. Springer, Heidelberg, Aug. 2008.
[28] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, Aug. 1990.
[29] Y. Dodis. Efficient construction of (distributed) verifiable random functions. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 1–17. Springer,

Heidelberg, Jan. 2003.
[30] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In S. Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, Jan. 2005.
[31] Y. Dodis, A. Yampolskiy, and M. Yung. Threshold and proactive pseudo-random permutations. In S. Halevi and T. Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 542–560. Springer, Heidelberg, Mar. 2006.
[32] A. Everspaugh, R. Chaterjee, S. Scott, A. Juels, and T. Ristenpart. The pythia PRF service. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 547–562, 2015.
[33] S. Faust, M. Kohlweiss, G. A. Marson, and D. Venturi. On the non-malleability of the Fiat-Shamir transform. In S. D. Galbraith and M. Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Heidelberg, Dec. 2012.
[34] Y. Frankel. A practical protocol for large group oriented networks. In J.-J. Quisquater and J. Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 56–61. Springer, Heidelberg, Apr. 1990.
[35] I. Giacomelli, J. Madsen, and C. Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security Symposium*, pages 1069–1083, 2016.
[36] L. Grassi, C. Rechberger, D. Rotaru, P. Scholl, and N. P. Smart. MPC-friendly symmetric key primitives. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 430–443. ACM Press, Oct. 2016.
[37] S. Halevi, Y. Ishai, E. Kushilevitz, N. Makriyannis, and T. Rabin. On fully secure MPC with solitary output. In D. Hofheinz and A. Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 312–340. Springer, Heidelberg, Dec. 2019.
[38] S. Jarecki, H. Krawczyk, and J. K. Resch. Updatable oblivious key management for storage systems. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *ACM CCS 2019*, pages 379–393. ACM Press, Nov. 2019.
[39] M. Keller, E. Orsini, D. Rotaru, P. Scholl, E. Soria-Vazquez, and S. Vivek. Faster secure multi-party computation of aes and des using lookup tables. In *International Conference on Applied Cryptography and Network Security*, pages 229–249. Springer, 2017.
[40] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, Nov. 2013.
[41] S. Kim and D. J. Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 503–536. Springer, Heidelberg, Aug. 2017.
[42] V. Kuchta and M. Manulis. Rerandomizable threshold blind signatures. In M. Yung, L. Zhu, and Y. Yang, editors, *Trusted Systems*, pages 70–89, Cham, 2015. Springer International Publishing.
[43] S. Micali and R. Sidney. A simple method for generating and sharing pseudo-random functions, with applications to clipper-like escrow systems. In D. Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 185–196. Springer, Heidelberg, Aug. 1995.
[44] M. Naehrig, R. Niederhagen, and P. Schwabe. New software speed records for cryptographic pairings. Cryptology ePrint Archive, Report 2010/186, 2010. https://eprint.iacr.org/2010/186.
[45] M. Naor, B. Pinkas, and O. Reingold. Distributed pseudo-random functions and KDCs. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 327–346. Springer, Heidelberg, May 1999.
[46] J. B. Nielsen. A threshold pseudorandom function construction and its applications. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 401–416. Springer, Heidelberg, Aug. 2002.
[47] D. Rotaru, N. P. Smart, and M. Stam. Modes of operation suitable for computing on encrypted data. Cryptology ePrint Archive, Report 2017/496, 2017. http://eprint.iacr.org/2017/496.
[48] A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In D. B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.
[49] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In K. Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 1–16. Springer, Heidelberg, May / June 1998.
[50] Y. Wu, J. Su, and B. Li. Keyword search over shared cloud data without secure channel or authority. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 580–587, 2015.

Appendix

A STRONG FTKD: CONSTRUCTION AND PROOF

In this section we provide our construction for strong FTKD (see Fig. 15. The construction is extended from the standard FTKD construction provided in Fig. 8 using a *trapdoor commitment* and a *non-interactive zero-knowledge proof*. The strengthening is analogous to the DPRF construction of DiSE.



Ingredients

- Let G_0, G_1, and G_T be multiplicative cyclic groups of prime order p such that there exists a bilinear pairing e: $G_0 \times G_1 \to G_T$ that is efficiently computable and non-degenerate; we let $g_0 \in G_0$ and $g_1 \in G_1$ be generators of G_0 and G_1 respectively.
- Let $\mathcal{H}_0 : \{0,1\}^* \to G_0$, $\mathcal{H}_1 : \{0,1\}^* \to G_1$ and $\mathcal{H}' : \{0,1\}^* \to \{0,1\}^{\text{poly}(\kappa)}$ be hash functions modeled as random oracles.
- Let SSS be Shamir's secret sharing scheme.
- Let TCom := $(\text{Setup}_{\text{com}}, \text{Com})$ be a trapdoor commitment scheme (Def. B.2)
- Let NIZK := $(\text{Prove}^{\mathcal{H}'}, \text{Verify}^{\mathcal{H}'})$ be a simulation-sound NIZK proof system (Def. B.4).

Our strongly secure FTKD construction

- $\text{Setup}(n,t) \to (\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$. Sample $s \leftarrow_{\$} \mathbb{Z}_p$ and then compute $(s_1, \ldots, s_n) \leftarrow \text{SSS}(n,t,p,s)$. Run $\text{Setup}_{\text{com}}(1^\kappa)$ to get pp_{com}. Compute a commitment $\gamma_i := \text{Com}(s_i, pp_{\text{com}}; r_i)$ by picking r_i at random. $sk_i := (s_i, r_i)$. Set $pp := (p, g_0, G_0, g_1, G_1, G_T, e, \mathcal{H}_0, \mathcal{H}_1, \mathcal{H}', \gamma_1, \ldots, \gamma_n, pp_{\text{com}})$.
- $\text{DKdf}(\llbracket \mathbf{sk} \rrbracket_S, [j : \rho, S]) \to [j : k/\perp]$. This is a two round protocol as described below:

Round-1. Party j sends ρ to each party $i \in S$.

Round-2. Each party $i \in S$ sends back $((w, h_i), \pi_i)$ to j where

$$w := \begin{cases} \mathcal{H}_0(x) \text{ if } \rho = (x, \text{`left'}) \\ \mathcal{H}_1(y) \text{if } \rho = (y, \text{`right'}) \\ z \text{ if } e(\mathcal{H}_0(x), \mathcal{H}_1(y)) = ((x,y), \text{`whole'}) \end{cases} ; \; h_i = w^{s_i}$$

and run $\text{Prove}^{\mathcal{H}'}$ with the statement stmt_i: $\{\exists \alpha, \beta \text{ s.t. } h_i = w^\alpha \wedge \gamma_i = \text{Com}(\alpha, pp_{\text{com}}; \beta)\}$ and witness (s_i, r_i) to obtain the proof π_i.

Finalize. Party j, if receives at least $t-1$ responds then parse party i's responds as as $((w, h_i), \pi_i)$ and check if $\text{Verify}^{\mathcal{H}'}(\text{stmt}_i, \pi_i) = 1$ for all $i \in S$. If this check fails for any i, output $\perp$. Else, outputs $k := \prod_{i \in S'} h_i^{\lambda_{0,i,S'}}$ where $S' \subseteq S \cup \{j\}$ is of size t; otherwise outputs $\perp$.

- $\text{WKGen}(v, \sigma) =: \text{wk}$. Compute $\text{wk} := e(v, \mathcal{H}_b(\sigma))$ where $b := \begin{cases} 0 \text{ if } \sigma = x \\ 1 \text{ if } \sigma = y \end{cases}$

Figure 15: Our Strongly Secure FTKD construction.

Theorem A.1. *The construction, presented in Figure 15 is a strongly-secure FTKD construction under the BDDH assumption over the map $e(G_0, G_1) \to G_T$ in the programmable random oracle model.*

Proof. *(sketch)* This strengthening from the previous constriction is analogous to the one presented in DiSE. In particular, the same tools, namely a trapdoor commitment and a simulation sound NIZK is used in a similar fashion. We omit the details. □

B ADDITIONAL BUILDING BLOCKS

In this section we provide additional building blocks and useful concepts, many of which are taken verbatim from DiSE [13]. We include them for completeness.

B.1 Commitment

Definition B.1. A (non-interactive) 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 B.2 (Trapdoor (Non-interactive) Commitments.). Let $\Sigma = (\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ be a (non-interactive) commitment scheme. A trapdoor commitment scheme has two more PPT algorithms SimSetup and SimOpen:

- $\mathsf{SimSetup}(1^\kappa) \to (pp_{\mathsf{com}}, \tau_{\mathsf{com}})$: 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 B.3. *Clearly, a trapdoor commitment can be binding against PPT adversaries only.*

B.1.1 Concrete instantiations. Practical commitment schemes can be instantiated under various settings:

Random oracle. In the random oracle model, a commitment to a message m is simply the hash of m together with a randomly chosen string of length r of an appropriate length.

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

B.2 Non-interactive Zero-knowledge

Let R be an efficiently computable binary relation. For pairs $(s, w) \in R$, we refer to s as the statement and w as the witness. Let L be the language of statements in R, i.e. $L = \{s : \exists w \text{ such that } R(s, w) = 1\}$. We define non-interactive zero-knowledge arguments of knowledge in the random oracle model based on the work of Faust et al. [33].

Definition B.4 (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 NIZK for a binary relation R consists of two PPT algorithms Prove and Verify with oracle access to $\mathcal{H}$ defined as follows:

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

These two algorithms must satisfy the following properties:

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

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

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

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

  for some negligible function negl, where
  - $\mathcal{S}_1$ simulates the random oracle $\mathcal{H}$;
  - $\mathcal{S}'_2$ returns a simulated proof $\pi \leftarrow \mathcal{S}_2(s)$ on input (s, w) if $(s, w) \in R$ and $\perp$ otherwise;
  - $\mathcal{S}_1$ and $\mathcal{S}_2$ share states.
- **Argument of knowledge**: There must exist a PPT simulator $\mathcal{S}_1$ such that for all PPT adversary $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}^{\mathcal{A}}$ such that

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

  for some negligible function negl, where
  - $\mathcal{S}_1$ is like above;

– Q is the list of (query, response) pairs obtained from $\mathcal{S}_1$.

Fiat-Shamir transform. Let (Prove, Verify) be a three-round public-coin honest-verifier zero-knowledge interactive proof system (a sigma protocol) with unique responses. Let $\mathcal{H}$ be a function with range equal to the space of the verifier's coins. In the random oracle model, the proof system $(\mathsf{Prove}^{\mathcal{H}}, \mathsf{Verify}^{\mathcal{H}})$ derived from (Prove, Verify) by applying the Fiat-Shamir transform satisfies the zero-knowledge and argument of knowledge properties defined above. See Definition 1, 2 and Theorem 1, 3 in Faust et al. [33] for more details. (They actually show that these properties hold even when adversary can ask for proofs of false statements.)

C GROUP COMMITMENTS FROM MERKLE-TREE

Figure 16 provides an instantiation of a group commitment scheme using Merkle-tree hashing. It is proven secure in the random oracle model. The claim of security is formalized in the following theorem.

Theorem C.1. *Let $\mathcal{H}$ be a random oracle. Then the construction described in Figure 16 is a group commitment scheme as per Definition 6.1.*

Proof. The correctness and compactness are obvious. The binding property follows from the collision resistance of the hash function (random oracles are collision resistant) easily.

Cardinality-binding. To observe the cardinality-binding property first note that the commitments are *position binding* [22], meaning that no cheating committer can open two messages in the same position i — this is straightforward to see from the binding property, because same position immediately implies that the unique commitments must be the same (due to collision resistance of the hash function). So, the only way an adversary can succeed in breaking cardinality-binding for a commitment $\gamma = (M, h)$ is by opening to more positions than M. There are two possibilities: first suppose that γ was honestly produced. In this case, attempting to open any message at a new position would lead to wrong label at a node and hence a wrong hash value at the next node (by collision resistance). In the other case, let's assume that the correct cardinality is $N > M$, and the adversary lied about it; then on opening to N positions all the hash values would match, but the final label at the root would be equal to N (instead of the M as claimed by the adversary) — leading GVer to fail.

Hiding. Finally, the hiding property can be derived from the fact that $\mathcal{H}(i\|m\|r)$ hides m semantically when r is uniformly random and $\mathcal{H}$ is a random oracle. It is not hard to see that this fact holds when $\mathcal{H}$ is a random oracle, because the adversary is not able predict r except with negligible probability–which implies that the output is uniformly random conditioned on the adversary's view.

Now, in the security-game $\mathsf{Hide}_{\mathcal{A}}$, the adversary's view consists of all unique commitments which are all the hash-values of the Merkle-tree plus the openings of the messages $\{m_i\}_{i\in I}$ that are indifferent across two message vectors $\mathbf{m}_0$ and $\mathbf{m}_1$. Additionally, the adversary may acquire random oracle outputs (polynomially many) by querying the random oracle. However, for all the messages $\{m_i\}_{i\notin I}$ that differ across two challenge vectors, the corresponding randomnesses are not known to the adversary (this is ensured by requiring that $i^\star \in I$ within the oracle O^{open}), therefore, the random oracle outputs are also unknown to the adversary with

– $\mathsf{GSetup}(1^\kappa)$: Sample a hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\kappa$ and output its description as the public parameters $pp := \mathcal{H}$.
– $\mathsf{GCommit}(pp, \mathbf{m})$: Let $N = |\mathbf{m}|$ and $\lambda = \lceil \log(N) \rceil$.
  – Sample N uniform random values $r_1, \ldots, r_N$.
  – Compute the hash values and labels for the leaves (or level-0) for $i := 1 \to 2^\lambda$:

$$h_i^0 := \begin{cases} \mathcal{H}(i\|m_i\|r_i) \text{ if } i \le N \\ \mathcal{H}(i\|0\ldots0) \text{ if } i > N \end{cases}$$

and

$$v_i^0 := \begin{cases} 1 \text{ if } i \le N \\ 0 \text{ if } i > N \end{cases}$$

  – Then recursively compute other nodes of the trees. In particular, the nodes at the j-th level for $j := 1 \to \lambda$ and $i := 1 \to 2^{\lambda - j}$ are computed as:[a].

$$h_i^j := \mathcal{H}(v_{2i-1}^{j-1}\|h_{2i-1}^{j-1}\|v_{2i}^j\|h_{2i}^j)$$

and

$$v_i^j = v_{2i-1}^{j-1} + v_{2i}^{j-1}$$

  – The commitment γ is defined as $\gamma := (N, h_1^\lambda)$.
  – For $i \in [N]$, the opening p_i is defined to be $(i\|r_i)$ and the unique commitment q_i is set to be a tuple consisting of all intermediate auxiliary hash values and corresponding labels required to compute the root: $(h^0_{\mathsf{sib}(i,0)}, h^1_{\mathsf{sib}(\lceil i/2\rceil,1)}, \ldots, h^\lambda_{\mathsf{sib}(\lceil i/2^{\lambda-1}\rceil)})$ $(v^0_{\mathsf{sib}(i,0)}, v^1_{\mathsf{sib}(\lceil i/2\rceil,1)}, \ldots, v^{\lambda-1}_{\mathsf{sib}(\lceil i/2^{\lambda-1}\rceil)})$ where $\mathsf{sib}(i,j)$ denotes the sibling of the i-th node at the j-th level.
  – Output the group commitment γ, the unique commitment vector $\mathbf{q}$ and the opening vector $\mathbf{p}$.
– $\mathsf{GVer}(pp, (\gamma, q), (m, p))$: Parse p as (i, r), q as $(g_0, g_1, \ldots, g_{\lambda-1}), (\mu_0, \ldots, \mu_{\lambda-1})$ and γ as (M, h). Then do as follows:
  – Compute $h_0 := \mathcal{H}(i\|m\|r)$ and $v_0 := 1$.
  – Let η be the bit-representation of the string i, let $\eta[j]$ be the j-th bit of η.
  – For $j := 1 \to \lambda$, compute:

$$h_j := \begin{cases} \mathcal{H}(v_{j-1}\|h_{j-1}\|\mu_{j-1}\|g_{j-1}) \text{ if } \eta[j] = 0 \\ \mathcal{H}(\mu_{j-1}\|g_{j-1}\|v_{j-1}\|h_{j-1}) \text{ if } \eta[j] = 1 \end{cases}$$

and

$$v_j := v_{j-1} + \mu_{j-1}$$

  – Finally verify whether $h = h_\lambda$ *and* $v_\lambda = M$ if it succeeds output 1, otherwise output 0.
– $\mathsf{CardVer}(pp, (\gamma, N)) = 0/1$: Parse γ as (M, h) and return 1 if $M = N$ and 0 otherwise.

[a]Note that, the labels v_{2i}^j denotes the total number of leaves below this node and for the root this will be equal to N

Figure 16: The Merkle-tree based group commitment

Game AT-Correct$_{\mathcal{A}}(1^\kappa)$:
- **run** $([\![\mathbf{sk}]\!]_n, pp) \leftarrow \mathsf{Setup}(1^\kappa, n, t)$.
- **set** CHAL, OUT := 0.
- **run** $C \leftarrow \mathcal{A}(pp)$;
  require $C \subset [n]$ **and** $|C| < t$.
- **run** $\mathcal{A}^{O^{\text{at-cor-enc}}, O^{\text{at-cor-dec}}, O^{\text{at-cor-chal}}}(\{sk_i\}_{i \in C})$.
- **return** OUT.

Figure 17: The correctness game of an ATSE scheme.

Oracle $O^{\text{at-cor-enc}}(j, \mathbf{m}, S)$:
require $j \in S$.
- **run** $[j : \text{op}] \leftarrow \mathsf{DGEnc}([\![\mathbf{sk}]\!]_{[n]}, [j : \mathbf{m}, S])$.
- **if** $j \notin C$ **then return** op.

Figure 18: The encryption oracle for the game AT-Correct**.**

Oracle $O^{\text{at-cor-dec}}(j, c, S)$:
require: $j \in S$.
- **run** $[j : \text{op}] \leftarrow \mathsf{DistDec}([\![\mathbf{sk}]\!]_{[n]}, [j : c, S])$.
- **if** $j \notin C$ **then return** op.

Figure 19: The decryption oracle for the game AT-Correct**.**

overwhelming probability. So, the only information adversary obtains on those differing messages $\{m_i\}_{i \in I}$ are $\mathcal{H}(i \| m_i \| r_i)$, which hides m_i as discussed above. □

An ATSE scheme is correct whenever a legitimately produced ciphertext c for an input message m, if decrypted (in presence of potential malicious corruption) in a distributed manner, outputs either m or $\perp$. In particular, an adversary should not be able to influence the decryption protocol to produce a message *different from m*. Furthermore, *strong-correctness* additionally requires that c should *only decrypt to m* (not even $\perp$) when decryption is performed honestly. We stress that strong-correctness is crucial for our use-case (see Remark C.4).

Definition C.2 ((Strong)-Correctness). An ATSE scheme is *correct* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that the game AT-Correct$_{\mathcal{A}}$ outputs 1 with probability at most $\text{negl}(\kappa)$. The security game AT-Correct is described in Figure 17, the corresponding oracles are described in Figure 18, Figure 19 and Figure 20. An ATSE scheme is called *strongly-correct* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that the game AT-Str-Correct$_{\mathcal{A}}$ outputs 1 with probability at most $\text{negl}(\kappa)$, where the game AT-Str-Correct is described in Figure 21, which uses a difference challenge oracle (but the same encryption and decryption oracle) as described in Figure 22.

Remark C.3. There are two flags in the game AT-Correct (Figure 17), namely CHAL which simply ensures that the challenge oracle is accessed only once; and OUT which indicates whether the adversary satisfies the winning condition. The default value of OUT is set to 0 and it is set to 1 only within the challenge oracles $O^{\text{at-cor-chal}}, O^{\text{at-str-cor-chal}}$, when the winning conditions are met.

Remark C.4 (Correctness for massive data encryption). For our application where a client encrypts large volumes of data, *strong correctness* is a crucial requirement. As also explained in DiSE (see

Oracle $O^{\text{at-cor-chal}}(j, S, j', S', \mathbf{m} = (m_1 \ldots, m_N), i)$:
require $j \in S \setminus C$ **and** $j' \in S' \setminus C$ **and** $i \in [N]$ **and** CHAL = 0 **and** OUT = 0.
- **set** CHAL := 1.
- **run** $[j : \text{op}] \leftarrow \mathsf{DGEnc}([\![\mathbf{sk}]\!]_n, [j : \mathbf{m}, S])$.
- **if** op = $\perp$ **then set** OUT := 0;
  else set $(c_1, \ldots, c_N) :=$ op **and do** :
  - **run** $[j' : \text{op}'] \leftarrow \mathsf{DistDec}([\![\mathbf{sk}]\!]_n, [j' : c_i, S'])$.
  - **if** $\text{op}' \in \{m_i, \perp\}$ **then set** OUT := 0; **else set** OUT := 1.

Figure 20: The challenge oracle for the game AT-Correct**.**

Game AT-Str-Correct$_{\mathcal{A}}(1^\kappa)$:
- **run** $([\![\mathbf{sk}]\!]_n, pp) \leftarrow \mathsf{Setup}(1^\kappa, n, t)$.
- **set** CHAL, OUT := 0.
- **run** $C \leftarrow \mathcal{A}(pp)$;
  require: $C \subset [n]$ **and** $|C| < t$.
- **run** $\mathcal{A}^{O^{\text{at-cor-enc}}, O^{\text{at-cor-dec}}, O^{\text{at-str-cor-chal}}}(\{sk_i\}_{i \in C})$.
- **return** OUT.

Figure 21: The strong-correctness game for ATSE scheme.

Oracle $O^{\text{at-str-cor-chal}}(j, S, j', S', \mathbf{m} = (m_1 \ldots, m_N), i)$:
require $j \in S \setminus C$ **and** $j' \in S' \setminus C$ **and** $i \in [N]$ **and** CHAL = 0 **and** OUT = 0.
- **set** CHAL := 1.
- **run** $[j : \text{op}] \leftarrow \mathsf{DGEnc}([\![\mathbf{sk}]\!]_n, [j : \mathbf{m}, S])$.
- **if** op = $\perp$ **then set** OUT := 0;
  else $(c_1, \ldots, c_N) :=$ op **and do** :
  - **run** $\text{op}' \leftarrow \mathsf{DistDec}^\dagger([\![\mathbf{sk}]\!]_n, [j' : c_i, S'])$.
  - **if** $\text{op}' = m_i$ **then set** OUT := 0; **else set** OUT := 1.

Figure 22: The challenge oracle for the game AT-Correct**.**

Remark 6.5 in [14]), the plaintext data is deleted from memory after storing the encrypted version. Without strong correctness, a corrupt server may return a wrong computation that remains undetected during encryption and leads the client to generate a malformed ciphertext. Later, even an honest decryption session would not be able to recover the plaintext, which is now lost permanently. With strong correctness, it is guaranteed that whatever a malicious party does during the encryption, any honest decryption at a later point would result in a correct message. Looking ahead, in our strongly secure FTKD scheme this is realized by adding a non-interactive zero-knowledge proof for the correct computation by each participating server during encryption (similar to DiSE). For this reason our performance evaluation only focuses on this version.

Definition C.5 (Message privacy). An ATSE scheme satisfies *message privacy* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that

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

where the security game AT-MsgPriv is described in Figure 23; the associated encryption, decryption and challenge oracles are described in Figure 24, Figure 25 and Figure 20 respectively.

Oracle $O^{\text{at-mp-enc}}(j, \mathbf{m}, S)$:
require $j \in S$.
– **run** $[j : \text{op}] \leftarrow \text{DGEnc}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : \mathbf{m}, S])$.
– **if** $j \notin C$ **then return** op.

Figure 24: The encryption oracle of game AT-MsgPriv**.**

Oracle $O^{\text{at-mp-dec}}(j, c, S)$:
require $j \in S \setminus C$.
– **run** $[j : \text{op}] \leftarrow \text{DistDec}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : c, S])$.

Figure 25: The decryption oracle of game AT-MsgPriv**.**

Oracle $O^{\text{at-mp-chal}}(j^\star, \mathbf{m}_0, \mathbf{m}_1, S^\star)$:
require $j^\star \in S \setminus C$ **and** $|\mathbf{m}_0| = |\mathbf{m}_1|$ **and** CHAL $= 0$.
– **set** CHAL $:= 1$.
– **for** i **such that** $\mathbf{m}_0[i] = \mathbf{m}_1[i]$ **set** $I := I \cup \{i\}$.
– **run** $[j^\star : \text{op}] \leftarrow \text{DGEnc}(\llbracket \mathbf{sk} \rrbracket_n, [j : \mathbf{m}_b, S^\star])$.
– **set** $\mathbf{c}^\star := \text{op}$.
– **return** $c^\star$.

Figure 26: The challenge oracle of game AT-MsgPriv**.**

Oracle $O^{\text{at-mp-pc-dec}}(j, c, S)$:
require $j \in S$ **and** CHAL $= 1$ **and** $c = \mathbf{c}^\star[i^\star]$ **and** $i^\star \in I$.
– **run** $[j : \text{op}] \leftarrow \text{DistDec}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : c, S])$.
– **if** $j \notin C$ **return** op.

Figure 27: Post-challenge decryption oracle for AT-MsgPriv**.**

Game AT-MsgPriv$_{\mathcal{A}}(1^\kappa, b)$:
– **set** CHAL $:= 0$.
– **set** $\mathbf{c}^\star := \emptyset$.
– **set** $I := \emptyset$.
– **run** $(\llbracket \mathbf{sk} \rrbracket_n, pp) \leftarrow \text{Setup}(1^\kappa, n, t)$.
– **run** $C \leftarrow \mathcal{A}(pp)$;
require $C \subset [n]$ **and** $|C| < t$.
– **run** $b' \leftarrow \mathcal{A}^{O^{\text{at-mp-enc}}, O^{\text{at-mp-dec}}, O^{\text{at-mp-chal}}, O^{\text{at-mp-pc-dec}}}(\{sk_i\}_{i \in C})$
– **return** b'.

Figure 23: Description of AT-MsgPriv

Remark C.6 (Comparison with DiSE). We note that the adversary gets access to two decryption oracles apart from the encryption and challenge oracles. The decryption oracle $O^{\text{at-mp-dec}}$ requires an honest client, and the output is *not* returned to the attacker. This captures that in the distributed setting an adversary may learn additional information from the transcript by participating in a decryption protocol (as a corrupt server) initiated by an honest client– this is similar to DiSE. The other oracle $O^{\text{at-mp-pc-dec}}$ does not have a counterpart in DiSE as it is specific to the group-encryption setting. It can only be accessed after the challenge query and it requires the ciphertext c (i) to come from the challenge vector $\mathbf{c}^\star$ and (ii) must be corresponding to a message m which *does not differ* across two challenge message vectors $\mathbf{m}_0$ and $\mathbf{m}_1$. These are ensured by requiring that $c \in c^\star[i^\star]$ and $i^\star \in I$. Note that the set I is updated within the challenge oracle– in particular I collects the indexes i for which $\mathbf{m}_0[i] = \mathbf{m}_1[i]$. Intuitively, this captures an important security goal which ensures that when a single ciphertext c (generated by encrypting a group of messages $\mathbf{m}$) is decrypted (to $m \in \mathbf{m}$), then the semantic security of any other message ($m' \in \mathbf{m}$ and $m \neq m'$) in the same group must hold.

Remark C.7 (Intuitions on privacy guarantee). Let us provide some intuitions here about on our privacy goals (as described in Section 2.2) are captured by the message privacy definitions. An unprivileged decryptor, who is potentially malicious, wins if it learns the message within a challenge ciphertext, even when it learns other messages from the same group. Formally we capture this by allowing the oracle $O^{\text{at-mp-pc-dec}}$: for two challenge messages $\mathbf{m}_0$ and $\mathbf{m}_1$, the adversary gets to query this oracle on the indexes for which the messages are the same. Furthermore, when an encryptor in malicious, then it wins the game if it learns the group-specific keys derived by $j^\star$ — the key would let her decrypt the challenge ciphertext without further interaction.

Finally we define (strong) authenticity. In particular, we use a one-more type definitions, in that the adversary wins the game if and only if it is able to produce *one more* legitimate ciphertexts than it is supposed to acquire by legitimate encryption/decryption queries. Additionally, the attacker is allowed to make targeted decryption queries on ciphertexts generated by honest clients.

Definition C.8 ((Strong)-Authenticity). An ATSE scheme satisfies *authenticity* if for all PPT adversary $\mathcal{A}$, there exists a negligible function negl such that, for any security parameter $\kappa \in \mathbb{N}$ the game AT-Auth$_{\mathcal{A}}(1^\kappa)$ outputs 1 with probability at most negl(κ), where the security game AT-Auth is described in Figure 28; the corresponding encryption, decryption, targeted-decryption and challenge(a.k.a. forgery) oracles are described in Figures 29, 30, 31, 32 respectively. The strong authenticity game (described in Figure 33) uses a different challenge oracle (described in Figure 34), but the same encryption/decryption oracles.

Game AT-Auth$_{\mathcal{A}}(1^\kappa)$:
– **set** ct $:= 0$ **and** $L_{\text{ctxt}} := \emptyset$.
– **set** SUCC $:= 0$ **and** CHAL $:= 0$.
– **run** $(\llbracket \mathbf{sk} \rrbracket_n, pp) \leftarrow \text{Setup}(1^\kappa, n, t)$.
– **run** $C \leftarrow \mathcal{A}(pp)$;
require $C \subset [n]$ **and** $|C| < t$.
– **run** $\mathcal{A}^{O^{\text{at-au-enc}}, O^{\text{at-au-dec}}, O^{\text{at-au-tar-dec}}, O^{\text{at-au-chal}}}(\{sk_i\}_{i \in C})$.
– **return** SUCC.

Figure 28: Description of the authenticity game

Oracle $O^{\text{at-au-enc}}(j, \mathbf{m}, N, S)$:
require $j \in S$.
– **run** $[j : \text{op}] \leftarrow \text{DGEnc}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : \mathbf{m}, S])$.
– **if** $j \notin C$ **then** $L_{\text{ctxt}} := L_{\text{ctxt}} \cup \{\text{op}\}$;
else set ct $:=$ ct $+ N|S \setminus C|$.

Figure 29: Encryption oracle for AT-Auth

Oracle $O^{\text{at-au-dec}}(j, c, S)$:
require: $j \in S$.
– **run** $[j : \text{op}] \leftarrow \text{DistDec}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : c, S])$.
– **if** $j \in C$ **then set** ct $:=$ ct $+ |S \setminus C|$.

Figure 30: Decryption oracle for AT-Auth**.**

Oracle $\mathcal{O}^{\text{at-au-tar-dec}}(j, i, S)$:
require: $j \in S \setminus C$ **and** $i \in [|L_{\text{ctxt}}|]$.
– **set** $c := L_{\text{ctxt}}[i]$.
– **run** $[j : \text{op}] \leftarrow \text{DistDec}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : c, S])$.

Figure 31: Targeted decryption oracle for AT-Auth

Oracle $\mathcal{O}^{\text{at-au-chal}}(L_{\text{forge}})$:
– **set** $k := \lfloor \text{ct}/g \rfloor$, where $g := t - |C|$.
– **set** $((j_1, S_1, c_1), \ldots, (j_{k+1}, S_{k+1}, c_{k+1})) := L_{\text{forge}}$;
require CHAL $= 0$ **and** SUCC $= 0$ **and** $j_1, j_2, \ldots, j_{k+1} \notin C$ **and** $\forall\, i \neq i' :\ c_i \neq c_{i'}$.
– **set** CHAL $:= 1$.
– **run** $\{\text{op}_i \leftarrow \text{DistDec}^{\dagger}(\llbracket \mathbf{sk} \rrbracket_n, [j_i : c_i, S_i])\}_{i \in [k+1]}$.
– **if** $\forall\, i \in [k+1]\ : \text{op}_i \neq \perp$ **then set** SUCC $:= 1$;
else set SUCC $:= 0$.

Figure 32: Challenge/forgery oracle for AT-Auth**.**

Remark C.9. In AT-Auth (and AT-Str-Auth) the flag SUCC is used for keeping track of whether the one-more condition is satisfied. Notice that, the flag is set to 1 only when the adversary is able to produce $k+1$ valid ciphertexts while acquiring enough information to produce at most k valid ciphertexts. In the stronger version a ciphertext is called valid when the decryption succeeds (that is, does not output $\perp$) even when it is done with corrupt servers, whereas for the plain version the decryption is done honestly.

Remark C.10. Both versions of authenticity are relevant for our massive-data encryption application. This is in contrast with the correctness (see Remark C.4), in which stronger guarantee is, in fact, crucial. Looking ahead, our ATSE construction requires the underlying FTKD to be strongly-secure to achieve strong correctness. Using strongly correct FTKD provides strong authenticity.

Game AT-Str-Auth$_{\mathcal{A}}(1^{\kappa}, b)$:
– **set** ct $:= 0$ **and** $L_{\text{ctxt}} := \emptyset$.
– **set** SUCC $:= 0$ **and** CHAL $:= 0$.
– **run** $(\llbracket \mathbf{sk} \rrbracket_n, pp) \leftarrow \text{Setup}(1^{\kappa}, n, t)$.
– **run** $C \leftarrow \mathcal{A}(pp)$;
require $C \subset [n]$ **and** $|C| < t$.
– **set** $g := t - |C|$.
– **run** $\mathcal{A}^{\mathcal{O}^{\text{at-au-enc}}, \mathcal{O}^{\text{at-au-dec}}, \mathcal{O}^{\text{at-au-tar-dec}}, \mathcal{O}^{\text{at-str-au-chal}}}(\{sk_i\}_{i \in C})$.
– **return** SUCC.

Figure 33: Description of the strong authenticity game

Oracle $\mathcal{O}^{\text{at-str-au-chal}}(L_{\text{forge}})$:
– **set** $k := \lfloor \text{ct}/g \rfloor$, where $g := t - |C|$.
– **set** $((j_1, S_1, c_1), \ldots, (j_{k+1}, S_{k+1}, c_{k+1})) := L_{\text{forge}}$;
require CHAL $= 0$ **and** SUCC $= 0$ **and** $j_1, j_2, \ldots, j_{k+1} \notin C$ **and** $\forall\, i \neq i' :\ c_i \neq c_{i'}$.
– **set** CHAL $:= 1$.
– **run** $\{\text{op}_i := \text{DistDec}(\llbracket \mathbf{sk} \rrbracket_n, [j_i : c_i, S_i])\}_{i \in [k+1]}$.
– **if** $\forall\, i \in [k+1]\ : \text{op}_i \neq \perp$ **then set** SUCC $:= 1$;
else set SUCC $:= 0$.

Figure 34: Challenge / forgery oracle for AT-Str-Auth**.**

Remark C.11 (Comparison with DiSE-TSE). The main difference with DiSE-TSE comes up in the encryption oracle $\mathcal{O}^{\text{at-au-enc}}$ (Fig. 29). The oracle, for each legitimate query, increments the counter by $N|S \setminus C|$ as opposed to $|S \setminus C|$ whenever j is corrupt. This exactly captures the fact that all N encryptions are authenticated together with one interaction — in other words, by one interaction the adversary may gain enough information to produce N legitimate ciphertexts.

Remark C.12 (Special structure of encryption query). Note that, we have a special structure of the encryption query to the oracle $\mathcal{O}^{\text{at-au-enc}}$. In particular, since N may not (in our construction it could not be unless we include it in the query explicitly) be correctly determined immediately from an execution when $j \in C$, it is required as an additional input. Now, observe that this makes the definition stronger, as it allows the adversary to cheat by providing a different number $N' < N$ enforcing the oracle to undercount ct (and thereby stay below the forgery budget). The construction needs to take care of such attack. Our construction achieves this by enforcing cardinality-verification at the server's end and the cardinality-binding property of the underlying group commitment (c.f. Sec. 6).

Remark C.13 (Intuition on authenticity guarantee). Recall from Sec. 2.2 that we require our authenticity definition to capture that *all ciphertexts* must be produced with legitimate interactions with at least one honest server even if $t-1$ servers are malicious; alternatively the honest servers (together) should be able to keep track of the total number of legitimate ciphertexts that are being produced in a given period. This prevents a corrupt encryptor to produce *any* legitimate ciphertext *locally*. Without this guarantee a potentially malicious encryptor would be able to *locally* produce arbitrary many ciphertexts and then dump all of them into the storage server. Furthermore, our notion ensures that a malicious encryptor can not even produce duplicate ciphertexts encrypting the same messages without further interaction— this is analogous to the strongest security guarantee of standard (non-interactive) authenticated encryption, known as *ciphertext integrity* [18].[11] To see this point consider the following attack: a malicious encryptor *interacts once* for encrypting a group of 100 messages and then produces 1000 distinct ciphertexts *locally* where each group of 10 ciphertexts encrypts the same message — this fails to satisfy our authenticity guarantee. Of course, one is able to duplicate by repeatedly interacting (10 times in this case) with honest servers, but our notion ensures that duplication has a significant cost (in terms of interaction). In the enterprise KMS application this particular feature helps in reducing the possibility of data duplication. (We leave it as a open question to fully protect against duplication in our setting — from the first glance it seems hard due to the CPA-security requirement which requires randomized commitments)

Remark C.14. The above guarantee is captured in our definition because, the adversary wins if it is able to produce *one more* ciphertexts than the challenger can keep track of. The challenger does

[11] A weaker guarantee is *plaintext integrity*, in which one can produce unlimited number of ciphertexts of the same message — the failed construction (c.f. Fig 35) seems to achieve a similar weaker guarantee in the interactive setting, but fails to achieve our definition.

the book-keeping by updating the count in various oracles, from which we can infer how legitimate ciphertexts can be generated. However, one slightly sub-optimal guarantee is that, we update the counter also in the decryption oracle — meaning that the adversary may generate one valid ciphertext by making a decryption query. This is the short-coming of our construction which we inherit from the DiSE TSE construction (see Remark 6.11 in [14]).

D FAILED APPROACHES USING PKE.

In this section we briefly discuss potential approaches involving *threshold public-key encryptions* such as threshold RSA [26]. Recall that, in a threshold public-key encryption the encryption works as a standard public encryption, whereas the decryption takes place with help of a threshold number of servers holding key-shares of the decryption key. In particular, a threshold PKE scheme (for t out of n threshold structure) has four algorithms: (i) $\mathsf{Setup}(1^\kappa, n, t) \to (pk, dk_1, \ldots dk_n)$ which outputs one single public encryption key and n secret decryption key shares $dk_1, \ldots, dk_n$; (ii) $\mathsf{Enc}(pk, m) \to c$ that encrypts a message to produce a ciphertext using the public key; (iii) $\mathsf{TDec}(dk_i, c) \to m_i$ which outputs a partial message when provided a key-share and a ciphertext as input; (iv) $\mathsf{Combine}(pk, m_1, \ldots, m_t) \to m/\perp$ which collects (at least) t message shares and output the whole message or $\perp$ (in case of error). Importantly, the encryption procedure is *purely non-interactive* and thus does not offer any authenticity by itself. (The OKMS framework [38] indeed follows a similar approach) However, one may think about combining it with a signature in the following two ways.

Signing by the client. Suppose, that the encryptor signs the ciphertext c locally after encryption to add a signature σ. The full ciphertext becomes (c, σ) and verification of the signature is performed during decryption. This is not useful as the goal of authenticity is to primarily *enforce the client* to generate legitimate ciphertexts. So, if the client holds the signing key, then no one can prevent her to sign a ciphertext with her own key.

Threshold signature by the servers. So, it is evident that we need to have some interaction with the servers during the encryption phase. This brings us to our next approach using threshold signature, Recall that, a threshold signature scheme also consists of four algorithms: (i) $\mathsf{Setup}(1^\kappa, n, t) \to (vk, sgk_1, \ldots, sgk_n)$, which outputs one verification key and n signing key shares $(sgk_1, \ldots, sgk_n)$; (ii) the signing algorithm $\mathsf{TSign}(sgk_i, m) \to \sigma_i$ that outputs a partial signature; (iii) $\mathsf{Combine}(vk, \sigma_1, \ldots, \sigma_t) \to \sigma/\perp$ which combines the partial signatures to a single one; (iv) the standard verification algorithm $\mathsf{SigVer}(vk, m, \sigma) \to 0/1$. Now, in this approach the servers hold shares of the signing-key of a threshold signature scheme; the client interacts with them to get a short commitment (like ours) of the group of message signed. Each individual message is then locally encrypted using the public encryption key. During the decryption, the client first locally verifies the commitment and then decrypts each ciphertext using threshold decryption. The detailed scheme is described in Figure 35.

The main problem of this approach is that public-key encryptions are inherently randomized. Therefore, no one can prevent the adversary from producing arbitrary many legitimate ciphertexts of the *same group of messages*locally, after obtaining the signature

Ingredients

- A threshold public-key encryption scheme (TPKE.Setup, Enc, TDec, TPKE.Combine).
- A threshold signature scheme (TSIG.Setup, TSign, SigVer, TSIG.Combine).
- A group commitment scheme (GSetup, GCommit, CardVer, GVer).

The Scheme

$\mathsf{Setup}(1^\kappa, n, t) \to (\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$. Execute the following steps:
- Run $\mathsf{TPKE.Setup}(1^\kappa, n, t)$ to get $(pk, dk_1, \ldots, dk_n)$
- Run $\mathsf{TSIG.Setup}(1^\kappa, n, t)$ to get $(vk, sgk_1, , \ldots, , sgk_n)$.
- Run $\mathsf{GSetup}(1^\kappa)$ to get pp_{com}.
- Set $sk_i := (dk_i, sgk_i)$ for $i \in [n]$ and $pp := (pp_{\mathsf{com}}, pk, vk)$.

$\mathsf{DGEnc}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : \mathbf{m}, S]) \to [j : \mathbf{c}/\perp]$. Let $N := |\mathbf{m}|$. This is an interactive protocol initiated by the client (party-j):
- Party-j computes $(\gamma, \mathbf{q}, \mathbf{p}) \leftarrow \mathsf{GCommit}(pp_{\mathsf{com}}, \mathbf{m})$ where $\mathbf{p} = (p_1, \ldots, p_N)$ and $\mathbf{q} = (q_1, \ldots, q_N)$.
- Party-i for $i \in S$ returns partial signatures $\sigma_i \leftarrow \mathsf{TSign}(sgk_i, (j, \gamma, N))$.
- Party-j, on receiving at least t such partial signatures combines them $\sigma \leftarrow \mathsf{TSIG.Combine}(vk, \sigma_1, \ldots, \sigma_t)$ and then compute ciphertext $e_i := (j, N, \gamma, m_i)$ for all $i \in [N]$.
- The final ciphertext consists of $(c_1, \ldots, c_N)$ where $c_i = (j, N, \gamma, \sigma, e_i)$

$\mathsf{DistDec}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : c, S]) \to [j : m/\perp]$ This is an interactive protocol initiated by the client (party-j):
- Party-j parses c as $(j, N, \gamma, \sigma, e)$ and locally verifies $\mathsf{SigVer}(vk, (j, N, \gamma), \sigma)$, if the verification fails then output $\perp$. Otherwise go to the next step.
- Party-j sends e to the servers in S.
- Each server i in S returns a partial decryption $m_i \leftarrow \mathsf{TDec}(dk_i, e)$.
- On receiving at least t such partial decryptions, party-j combines $m \leftarrow \mathsf{TPKE.Combine}(pk, m_1, \ldots, m_t)$ and outputs that.

Figure 35: A failed approach based on PKE and Threshold Signatures

(a similar failed approach is described in DiSE; see Appendix B.2 of [14]). Therefore, this approach falls short of our authenticity requirement. More intuitions on this can be found in Remark C.13.

E EXPERIMENTAL RESULTS

We report throughput and latency results in 3 settings: LAN (2 ms round-trip time) in Figure 36, WAN (30 ms round-trip time) in Figure 37, and geo-distributed (200 ms round-trip time) in Figure 14. We refer the reader to § 8 for a detailed explanation of these results and the comparison with DiSE.

t	n	Latency (ms/enc)						Throughput (enc/s)					
		DiSE 1 msg	ATSE 1 msg	DiSE 100 msg	ATSE 100 msg	DiSE 10000 msg	ATSE 10000 msg	DiSE 1 msg	ATSE 1 msg	DiSE 100 msg	ATSE 100 msg	DiSE 10000 msg	ATSE 10000 msg
$n/3$	6	7.67	15.17	1.21	0.24	1.02	0.08	1353.60	538.05	1935.06	17338.93	1740.39	23090.32
	12	8.77	18.36	1.80	0.23	1.60	0.08	777.31	356.17	970.34	16459.29	916.86	22939.21
	18	10.59	21.37	2.41	0.24	2.20	0.08	568.76	274.51	625.62	15572.01	610.03	23000.59
	24	11.10	23.54	3.01	0.25	2.83	0.08	490.91	217.81	477.74	15179.30	457.33	22732.65
$n/2$	6	8.02	16.19	1.34	0.24	1.10	0.08	1206.19	489.88	1643.08	17522.10	1708.22	22606.00
	12	10.26	21.27	2.09	0.25	1.79	0.08	740.08	317.91	848.86	16135.29	761.98	23062.49
	18	11.79	22.33	2.77	0.24	2.51	0.08	506.65	241.54	540.39	15484.78	512.59	23000.98
	24	13.07	27.37	3.52	0.26	3.24	0.08	390.81	185.69	409.22	14261.33	394.80	23227.21
$2n/3$	6	9.23	17.38	1.50	0.24	1.21	0.08	1137.89	475.06	1478.51	16916.38	1382.72	22737.52
	12	11.25	21.21	2.32	0.25	2.00	0.08	613.84	305.26	752.49	15702.13	693.90	23007.18
	18	12.85	25.26	3.20	0.25	2.81	0.08	447.55	214.51	483.95	14687.05	459.95	22861.80
	24	14.22	28.98	4.06	0.27	3.66	0.08	361.62	165.88	367.74	13648.82	348.42	22977.42
$n-2$	6	9.23	18.22	1.50	0.25	1.21	0.08	1137.89	493.38	1478.51	17218.14	1382.72	23057.09
	12	11.20	22.71	2.57	0.25	2.20	0.08	599.51	274.37	662.85	15434.01	608.87	22619.24
	18	14.34	29.46	3.67	0.27	3.21	0.08	403.58	185.74	429.69	14084.06	401.69	22785.89
	24	17.11	34.32	4.95	0.29	4.26	0.08	328.86	141.39	298.36	13085.54	296.27	22814.45
2	6	7.67	15.31	1.21	0.24	1.02	0.08	1353.60	552.28	1935.06	17672.40	1740.39	22808.23
	12	8.19	16.49	1.52	0.23	1.41	0.08	901.21	413.32	1128.34	17171.94	1022.70	22716.23
	18	8.76	17.88	1.88	0.23	1.81	0.08	706.69	297.89	831.78	16319.49	740.89	22929.70
	24	9.18	18.41	2.27	0.22	2.24	0.08	642.83	242.51	678.35	16124.29	580.40	22896.45

Figure 36: Encryption performance metrics averaging 10 repeated trials of 32 bytes messages in the LAN setting.

t	n	Latency (ms/enc)						Throughput (enc/s)					
		DiSE 1 msg	ATSE 1 msg	DiSE 100 msg	ATSE 100 msg	DiSE 10000 msg	ATSE 10000 msg	DiSE 1 msg	ATSE 1 msg	DiSE 100 msg	ATSE 100 msg	DiSE 10000 msg	ATSE 10000 msg
$n/3$	6	36.28	43.05	1.49	0.52	1.01	0.08	314.49	249.35	1657.44	13896.08	1844.77	23255.97
	12	37.27	46.22	2.09	0.51	1.60	0.08	276.10	208.23	889.04	13526.79	864.09	22701.11
	18	39.14	49.43	2.65	0.51	2.21	0.08	233.34	172.61	607.03	12752.33	588.50	23600.82
	24	39.20	51.37	3.28	0.51	2.82	0.08	247.29	148.51	443.23	12881.85	451.54	22819.92
$n/2$	6	36.10	44.25	1.60	0.53	1.11	0.08	305.11	235.45	1540.22	13746.63	1558.98	22587.91
	12	37.04	49.27	2.36	0.52	1.80	0.08	261.07	197.77	792.06	13107.11	762.56	23184.55
	18	39.54	50.83	3.04	0.54	2.50	0.08	262.93	168.53	526.64	12777.58	526.33	22882.63
	24	40.97	55.03	3.80	0.54	3.21	0.08	232.00	142.21	387.46	12195.49	392.91	23073.57
$2n/3$	6	36.48	44.15	1.73	0.52	1.21	0.08	298.07	225.83	1357.15	13574.06	1341.21	22901.73
	12	38.65	49.33	2.58	0.53	2.00	0.08	265.64	185.81	691.62	13328.30	679.10	22883.03
	18	41.10	54.32	3.39	0.54	2.81	0.08	242.94	153.92	456.75	12489.62	460.92	23066.88
	24	42.22	58.26	4.44	0.55	3.65	0.08	218.17	129.39	340.22	11684.43	350.85	22797.72
$n-2$	6	36.48	46.01	1.73	0.53	1.21	0.08	298.07	226.42	1357.15	13889.17	1341.21	22985.34
	12	39.45	52.26	2.84	0.53	2.20	0.08	267.34	184.07	616.21	12805.00	614.13	22738.41
	18	41.93	57.09	3.93	0.56	3.20	0.08	218.91	145.53	401.83	12141.07	395.32	22676.14
	24	44.51	62.56	5.04	0.56	4.27	0.08	190.73	114.34	290.57	11171.21	294.88	22560.14
2	6	36.28	44.82	1.49	0.53	1.01	0.08	314.49	253.26	1657.44	13403.01	1844.77	22888.23
	12	36.09	44.86	1.80	0.51	1.40	0.08	283.35	219.49	1011.29	13423.24	1007.36	22694.18
	18	37.41	46.03	2.16	0.50	1.82	0.08	265.28	188.43	724.91	13683.31	732.99	22932.86
	24	37.40	46.75	2.53	0.50	2.24	0.08	236.96	160.31	572.38	13437.24	574.20	22602.86

Figure 37: Encryption performance metrics averaging 10 repeated trials of 32 bytes messages in the WAN setting.

F MISSING PROOFS

In this section we provide the proofs missing from the main body.

F.1 Proof of Theorem 5.9

For simplicity we consider a simpler case where there are exactly $t-1$ corruptions (implies that $|S \setminus C| = 1$), and in fact they are exactly the parties with identities $\{1, \ldots, t-1\}$. It can be generalized to the general case with techniques analogous to DiSE [13].

The reduction from DP-PR(b) to BDDH works as follows:

- Receive a challenge tuple $(g_1^a, g_2^a, g_1^b, g_2^c, u)$. Compute $g_T^a := e(g_1^a, g_2)$.
- Sample $a_1, \ldots, a_{t-1}$ uniformly at random and set $sk_i := a_i$ for $i \in [t-1]$. Implicitly assume $sk := a$. All a_i for $i \in \{t, \ldots, n\}$ are well defined and can be computed in the exponent (but not in the clear due to hardness of discrete logarithm in all groups G_1, G_2, G_T) of either g_1 or g_2 by Lagrange interpolation.
- **Simulating the random oracle.** Guess the order of the random oracle queries on challenge $(x^\star, y^\star)$ ahead of time – this incurs a loss of $O(q^2)$ in the security assuming each random oracle receives q queries (because if the guess is wrong, then the simulation becomes incorrect). For a query $\mathcal{H}_0(x^\star)$, set $\mathcal{H}_0(x^\star) := g_1^b$ and for a query $\mathcal{H}_1(y^\star)$, set $\mathcal{H}_1(y^\star) := g_2^c$. For all other queries x to $\mathcal{H}_0$ (resp. y to $\mathcal{H}_1$) choose uniform random $r \leftarrow_\$ \mathbb{Z}_p$ and set $\mathcal{H}_0(x) := g_1^r$ (resp. $\mathcal{H}_1(x) := g_2^r$) and store the pair (x, r) (resp. (y, r)).
- **Simulating key-derivation oracle $O^{\text{pr-kd}}$.** Notice that in this case, the adversary is not allowed to ask a single key-derivation query to oracle $O^{\text{pr-kd}}$ (otherwise it would be listed in L_{wk} invoking ABORT = 1) on the challenge $(x^\star, y^\star)$. The partial-key queries are simulated using g_1^a and g_2^a for the `'left'` and `'right'` queries. In particular, for a `'left'` query on a value x return $(g_1^{a_i})^r$ on behalf of an honest party i, where $g_1^{a_i}$ is computed by Lagrange interpolation in the exponent and r is such that $\mathcal{H}_0(x) = g_1^r$. Note that, r is known to the

reduction due to programmability of random oracle, which also guarantees that even if the query $\mathcal{H}_0(x)$ is made later it will be returned g_1^r. The 'right' partial-key queries are simulated analogously. A 'whole' query on (x, y) is simulated by returning $(g_T^{a_i})^{r_1 r_2}$ on behalf of an honest party i, where r_1, r_2 are such that $\mathcal{H}_0(x) = g_1^{r_1}$ and $\mathcal{H}_1(y) = g_2^{r_2}$. As we already observed that, no query on $(x^\star, y^\star)$ is made, all such exponents (r_1, r_2) of the random oracle outputs are known.

– **Simulating challenge oracle $O^{\text{pr-chal}}$.** Finally, for a challenge query ('Challenge', $j^\star, S^\star, (x^\star, y^\star)$) return u' computed as follows:
  – Assuming $u = g_T^d$ (where d is either equal to abc or a uniform random value in $\mathbb{Z}_p$), the reduction computes $u_i := g_T^{d_i}$ where d_i is the i-th t-out-of-n Shamir secret sharing of d for every honest $i \in S^\star \setminus C$ by Lagrange interpolation.
  – Then use these u_i along with h_i received from the corrupt parties $i \in S^\star \cap C$ in the round-2 of protocol $\mathsf{DKdf}([\![\mathbf{sk}]\!]_{S^\star}, [\![j^\star]\!](x^\star, y^\star), S^\star)$ to reconstruct u' again by Lagrange interpolation.

Note that, the adversary does not gain any additional advantage from sending incorrect h_i's on behalf of the corrupt parties, in which case u' becomes different from u–because, information theoretically, even an unbounded adversary obtains exactly the same information irrespective of the exact values of h_i's in both the cases. So, without loss of generality we can assume $u' = u$. Now clearly when $u = g_T^{abc}$ then the reduction perfectly simulates DP-PR(0), as in that case each u_i is equal to $g_T^{a_i bc}$ as desired. On the other hand, when u is a random element in G_T, then the reconstruction would yield a uniform random value in G_T perfectly simulating DP-PR(1). This concludes the proof for pseudorandomness.

G A BRIEF OVERVIEW OF DISE

For reader's convenience we provide a brief overview of DiSE construction here. Some of the texts below are taken almost verbatim from [14].

DPRF. DiSE provides a generic construction of a threshold symmetric encryption (TSE). The construction is based on distributed pseudorandom functions (DPRF) as the main building block in a manner similar to our ATSE construction is based on FTKD. A DPRF is a distributed (and interactive) analog of a standard PRF. It involves a setup where each party obtains their secret-key and the public parameters. In other words, it can be thought of as a more restricted version of FTKD (Definition 5.1), that allows key-derivation only in one mode (namely "whole"). Evaluation on an input is performed collectively by any t parties where t ($\leq n$) is a threshold. Importantly, at the end of the protocol, only one special party (evaluator/client) learns the output. Similar to FTKD, a DPRF should meet two main requirements: (i) *consistency*: the evaluation should be independent of the participating set, and (ii) *pseudorandomness*: the evaluation's output should be pseudorandom to everyone else but the evaluator even if the adversary corrupts all other $t - 1$ parties and behaves maliciously.

In the malicious case, one can think of a slightly stronger property, called (iii) *correctness*, where after an evaluation involving up to $t - 1$ malicious corruptions, an honest evaluator either receives the correct output or can detect the malicious behavior.

Naor et al. [45] propose two very efficient (two-round) instantiations of DPRF, one based only on symmetric-key cryptography and another based on the DDH assumption. As mentioned earlier, the DDH based construction has similarities with our FTKD construction (Figure 8).

DiSE Construction. At a high level, DiSE uses a DPRF scheme to generate a pseudorandom key w that is used to encrypt the message m. But one needs to ensure that an adversary cannot use the same w to generate more than one valid ciphertext. To do so, Agrawal et al. bind w to the message m (and the identity of party-j), in particular by using $(j\|m)$ as an input to the DPRF. First, note that it is necessary to put j inside the DPRF, otherwise a malicious attacker can easily obtain w by replaying the input of the DPRF in a new encryption query and thereby recovering any message encrypted by an honest encryptor. In the protocol they made sure each party *checks* if a message of the form $(j, *)$ (just like our ATSE construction in Figure 11) is indeed coming from party-j. Second, this does not suffice as it reveals m to all other parties during the encryption protocols originated by honest parties and as a result fails to achieve even message privacy. To overcome this, they instead input a simple commitment to m to the DPRF.[12] The hiding property of the commitment ensures that m remains secret, and the binding property of the commitment binds w to this particular message. To enable the verification of the decommitment during the decryption, they need to also encrypt the commitment randomness along with m.

This almost works except that the attacker can still generate valid new ciphertexts by keeping m, j and w the same and using new randomness to encrypt m. This is prevented by making the ciphertext deterministic given m and w: they input w to a pseudorandom number generator to produce a pseudorandom string serving as a "one-time pad" that is used to encrypt m just by XOR'ing. (Again, this is similar to our ATSE construction.)

To summarize, the final DiSE construction can be informally described as follows: (i) the encryptor with identity j chooses a random ρ to compute $\alpha := \mathsf{Com}(m; \rho)$ where Com is a commitment and sends (j, α) to the participating parties, (ii) the participating parties then first check if the message (j, α) is indeed sent by j (otherwise they abort) and then evaluate the DPRF on $(j\|\alpha)$ for the encryptor to obtain the output w, (iii) finally, the encryptor computes $e = \mathsf{PRG}(w) \oplus (m\|\rho)$ and outputs the ciphertext (j, α, e).

It is not too hard to see that the DiSE construction satisfies ATSE correctness and security definitions for groups consisting of a single message (that is when $N = 1$).

[12] Since they use a single message each time, they can afford to use a plain commitment, whereas we need a special type of commitment such as group commitments.