Masters Thesis

Threshold Symmetric-Key Encryption for Tabular Data

Subhendu Pramanick

This page intentionally left blank.

Threshold Symmetric-Key Encryption for Tabular Data

Thesis submitted to the
Indian Statistical Institute, Kolkata
for partial fulfillment of requirements for the degree

of

Master of Technology in Cryptology & Security

by

Subhendu Pramanick
Roll No. CrS2320

Under the guidance of

Dr. Pratyay Mukherjee
Senior Director of Research
Supra Research

Dr. Debrup Chakraborty
Associate Professor
Cryptology & Security Research Unit (CSRU)
R. C. Bose Centre for Cryptology and Security
Indian Statistical Institute, Kolkata

Indian Statistical Institute, Kolkata

October 2025

This page intentionally left blank.

CERTIFICATE

This certifies that **Subhendu Pramanick (CrS2320)** submitted the thesis "**Threshold Symmetric-Key Encryption for Tabular Data** to **Indian Statistical Institute, Kolkata**. It is a record of legitimate research conducted under my supervision and guidance, and I believe it is deserving of consideration for the award of the Institute's *Masters of Technology in Cryptology and Security* degree.

Date:
Place:

Dr. Pratyay Mukherjee
Senior Director of Research
Supra Research

Date:
Place:

Dr. Debrup Chakraborty
Associate Professor
Cryptology and Security Research Unit
R. C. Bose Centre for Cryptology and Security
Indian Statistical Institute, Kolkata

DECLARATION

I hereby declare that the project entitled "**Threshold Symmetric-Key Encryption for Tabular Data**" submitted in partial fulfillment for the award of the degree of *Master of Technology in Cryptology and Security* completed under the supervision of Dr. Pratyay Mukherjee and Prof. Debrup Chakraborty, at Indian Statistical Institute is an authentic work. Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Date:

Place:

Subhendu Pramanick

Roll No.: CrS2320

Acknowledgements

I would like to express my deepest gratitude to my primary supervisor, Dr. Pratyay Mukherjee, for his expert guidance, constant encouragement, and invaluable support throughout this research. His insightful feedback and dedicated mentorship greatly shaped this thesis and inspired me to achieve my best.

I am also deeply grateful to my co-supervisors, Dr. Debrup Chakraborty for his valuable suggestions and for providing the resources and supportive environment necessary for my research. Their profound knowledge and constructive feedback significantly enhanced the quality of this thesis.

I would like to sincerely thank Subha Kar and Pabitra Mandal for their collaboration, insightful discussions, and continuous support throughout this research. Working alongside you has been invaluable, and your dedication and expertise greatly contributed to the progress and quality of this thesis.

Thank you all for your invaluable contributions to this work.

Date: Tuesday 14th October, 2025
Place:

Subhendu Pramanick
Roll No.: CrS2320

Abstract

Through the distribution of secret key information among several parties, threshold cryptography improves the security of cryptographic systems by preventing any one entity from possessing the entire secret key and requiring a threshold number of participants to carry out cryptographic operations. This paradigm not only mitigates single points of failure but also ensures fault tolerance in the presence of compromised or unavailable parties.

The Distributed Symmetric-key Encryption (DiSE) framework, introduced by Agrawal et al., realizes Threshold Symmetric-key Encryption (TSE) by requiring interactive participation from a threshold subset of servers for each encryption or decryption operation. While DiSE and similar TSE schemes provide strong security guarantees, they become inefficient for large datasets, as each encryption or decryption requires a separate round of server interaction, leading to significant computational and communication overhead.

To address these scalability challenges, Christodorescu et al. proposed Amortized Threshold Symmetric-key Encryption (ATSE), which allows a privileged client to encrypt a large collection of messages with a single interaction with the key servers. Importantly, decryption still requires threshold server participation for each ciphertext, ensuring that all clients receive the same level of privacy and authenticity as in DiSE. The ATSE framework is built upon the novel primitive of Flexible Threshold Key-derivation (FTKD), enabling the interactive derivation of pseudorandom keys in a threshold-secure manner.

This dissertation extends the ATSE paradigm to tabular data, presenting a new scheme that enables efficient and secure encryption of entire tables. Unlike traditional TSE schemes, which incur high overhead by requiring server interaction for each data cell, our construction allows a privileged client to encrypt an entire table with a single round of interaction, while maintaining strong security guarantees for individual decryptors. This advancement significantly improves the practicality and scalability of threshold encryption for modern, data-intensive applications.

Table of Contents

Abbreviations

PPT Probabilistic Polynomial Time. 13, 15, 17, 21, 23, 25, 26, 35

ATSE Amortized Threshold Symmetric-key Encryption. vii, 3, 4, 6, 11, 17, 19–21, 23, 25, 26, 29, 31, 32, 34, 35

BDDH Bilinear Decisional Diffe-Helman. 8, 17, 34

CPA Chosen-Plaintext Attack. 20

DiSE Distributed Symmetric-key Encryption. vii, 3, 5, 6, 31

DPRF Distributed PRF. 5, 11

FTKD Flexible Threshold Key-derivation. vii, 3, 4, 6, 11–17, 19, 28, 29, 31, 33–35

HSM Hardware Security Modules. 4, 5

KMS Key Management System. 5, 11, 12

NIST National Institute of Standards and Technology. 5

PRF Pseudo-Random Function. 3, 6, 11, 17

PRG Pseudo-Random Generator. 28, 31, 34, 35

TSE Threshold Symmetric-key Encryption. vii, 3, 5, 23, 31

1

Introduction

The proliferation of sensitive tabular data in enterprise and cloud environments has underscored the need for robust, scalable cryptographic protection. Threshold Symmetric-key Encryption (TSE) [AMMR18, CGMS21, DMSS24] schemes, such as DiSE [AMMR18, WH20], distribute key material across multiple servers and require a threshold of server participation for each encryption or decryption operation, thereby mitigating single-point compromise and supporting strong access control. However, these schemes incur significant communication and computational overhead when encrypting large datasets, as each cell or row must be processed through a separate round of server interaction. This limitation is especially acute in contemporary data-intensive applications, where high throughput and low latency are critical.

The Amortized Threshold Symmetric-key Encryption (ATSE) [CCMS21, dS22, VSB21], introduced by Agrawal et al. [CGMS21], addresses this bottleneck by allowing a *privileged* client to encrypt an entire batch of messages with a single round of server interaction, while ensuring that decryption for each ciphertext still requires threshold participation. This approach preserves the privacy and authenticity guarantees of traditional TSE, but achieves up to lower latency and higher throughput compared to parallelized DiSE, especially when encrypting large groups of messages [CGMS21]. The core innovation is a new primitive called Flexible Threshold Key-derivation (FTKD) [CCMS21], which enables interactive, threshold-based derivation of pseudorandom keys for each message in a batch, leveraging bilinear pairings and distributed constrained PRFs [BW13].

The original ATSE scheme by Agrawal et al. [CGMS21] is designed for *message vectors*: a privileged client interacts once with the key servers to derive a set of pseudorandom keys, each used to encrypt a single message in the vector. The security of the scheme relies on the FTKD protocol, which ensures that the derived keys are indistinguishable from random and cannot be reconstructed without threshold participation.

In this dissertation, we extend the ATSE framework to efficiently encrypt *tabular data*, i.e., matrices, which are the predominant data structure in modern databases and analytics pipelines [CGMS21]. Our scheme allows a privileged client to encrypt an entire table (matrix) in a single protocol round, while each cell is protected with a unique, locally derived key. Specifically, the construction employs:

- *Dual group commitments:* Each row and column of the matrix receives a unique group commitment, binding the data in two dimensions and strengthening authenticity guarantees.
- *Row and column key derivation:* Using FTKD, servers collaboratively derive row-specific group keys, which are then combined with column commitments to compute a unique whole key for each cell.
- *Efficient encryption and auditability:* Each cell is encrypted with a pseudorandom key derived from its row and column, and the ciphertext includes both commitments and opening information, enabling fine-grained access control and verifiability.

This design generalizes and strengthens the vector-based ATSE scheme: while the original construction only amortizes over a single dimension (message vector), our tabular ATSE scheme achieves amortization over both rows and columns, further reducing interaction overhead and enhancing scalability for large datasets [CGMS21].

Outline. Chapter 2 introduces the foundational notations, standard cryptographic tools such as bilinear pairings and Shamir secret sharing, and formalizes the algorithmic conventions used in security game abbreviations used throughout the dissertation. Chapter 3 presents the main Amortized Threshold Symmetric-key Encryption (ATSE) scheme for message vectors, following Christodorescu et al. (2021), beginning with its core building blocks—Flexible Threshold Key-derivation (FTKD) and Merkle tree group commitment and culminating in the formal construction of the ATSE scheme. Chapter 4 details our extension of ATSE to the tabular data setting, describing the construction, security definitions, and proofs tailored for efficient and secure threshold encryption of large-scale tabular datasets.

1.1 Related Work

Enterprises today deal with large amounts of (tabular) data on a regular basis, which must be safely protected from attackers. These data must be encrypted, which requires a lot of encryption keys. To safeguard encryption keys on behalf of the apps, they rely on specialized key management systems. Three primary components make up a typical deployment: a storage service that houses encrypted data, a group of [HAP18, BSA+25] that house the key material, and one or more encrypting clients. Additionally, additional non-encrypting clients (like application servers) might regularly use the storage service to decrypt particular data when needed. Among these, Hardware Security Modules (HSM) [SHS, NB23] frequently support the key management server. Replicated HSM deployment for high availability (three nodes are advised) raises expenses and complicates matters, which puts enterprises with limited resources at a disadvantage. Strict rotation, hierarchy, and deletion rules are necessary to prevent performance bottlenecks caused by storage constraints in HSMs, such as maximum tokens or session keys. While hybrid infrastructures have challenges with uneven key management across several cloud platforms, shared HSMs in cloud environments run the danger of illegal cross-tenant access in the event that provider software is compromised and are vulnerable to hardware side-channels [**?**].

Utilizing Threshold Cryptography to Address HSM Limitations. Threshold cryptography has

become a strong substitute for Hardware Security Modules (HSM) in order to solve security and scalability issues. By dividing secret keys into shares that are dispersed over several commodity servers, this method makes sure the key is never entirely rebuilt while being used. As long as attackers breach fewer servers than this threshold cryptographic [SY23, VM16] procedures require cooperation from a certain number of servers.

Both standardization initiatives and commercial adoption of threshold cryptosystems have been fueled by their advantages. For example, companies like Hashicorp Vault, Coinbase Custody, and Unbound Tech use the technology for data security and HSMs replacements, and the U.S. National Institute of Standards and Technology (NIST) is actively striving to standardize it. By utilizing commodity technology, these systems may scale to enterprise workloads at a reasonable cost[Nat22, KS22, Has24].

Meanwhile, by managing keys and encrypting data while it's at rest, programs like Keywhiz, Knox, and Hashicorp Vault [Has24, Ham23] automate secret management in cloud environments [Urm17]. A serious flaw still exists, even though some employ secret sharing during initialization: after keys are rebuilt in memory, they frequently remain unencrypted until the system is restarted. This weakness is immediately addressed by threshold cryptography, which makes sure that keys are never completely put back together, even during encryption and decryption. This characteristic highlights its expanding relevance in contemporary key management systems, together with scalability and compliance support.

Threshold Symmetric-key Encryption. We concentrate on threshold symmetric-key encryption (TSE) with an eye toward the use case of protecting enterprise data. To encrypt or decrypt any data in a symmetric-key setup, the application must communicate with the KMS. This allows us to verify the integrity of the encrypted data and implement fine-grained access control rules. Public-key encryption systems, on the other hand, let anybody encrypt data, leaving the application vulnerable to assaults like data poisoning and corruption. Current TSE systems [AMMR18, CGMS21, Ebr24, Muk20, MGB+24, DWZ+23, LLWL23, ZLA+23] adhere to a fundamental schemata:

Setup A threshold secret-sharing of the private key is established during the setup phase. This configuration can be used as a distributed key generation mechanism or as a ceremony in which a trustworthy administrator or group of administrators allot a share to every KMS server.

Encryption The client application communicates with a threshold number of KMS servers in order to encrypt a record. The KMS servers use the aforementioned shares to evaluate a (variant of) DPRF and determine a unique key that is associated with that record.

Decryption Using a portion of the ciphertext as input to the DPRF function to derive the same key material (as in the encryption phase above), the client application must once more communicate with a threshold number of KMS servers in order to decode any record.

DiSE. The first formal treatment of TSE in DiSE was recently published by Agrawal et al. [AMMR18], who also offered a construction based on DPRF. The construction process basically goes like this: in order to encrypt a message m, the client locally generates a commitment

of m and transmits it to the servers, who, after authenticating the client, return the PRF (partially) evaluated over the commitment (using their key-shares); the client then combines a predetermined number of responses to create a message-specific key that the client uses to locally encrypt m.

Amortized Symmetric Key Encryption. ATSE is a cryptographic scheme designed to efficiently secure large-scale enterprise data by leveraging threshold cryptography and amortized operations, allowing a privileged client to encrypt a large group of messages with a single interaction with key management servers, rather than requiring one interaction per message as in traditional schemes like DiSE[AMMR18]. ATSE introduces the FTKD primitive, enabling the derivation of both group-specific and message-specific keys in a secure, threshold-based way, so that encryption is efficiently amortized while each decryption still requires server interaction, preserving strong privacy and authenticity for individual records. This approach is especially beneficial for enterprise scenarios where a trusted application ingests and encrypts large volumes of data, and later less trusted applications decrypt individual records as needed, resulting in significant improvements in latency and throughput and supporting scalable, cost-effective, secure data management in distributed deployments [CGMS21].

2

Preliminaries

2.1 Notation

Let $\mathbb{N}$ denote the set of positive integers, and for any $n \in \mathbb{N}$, let $[n]$ represent the set $\{1, 2, \ldots, n\}$. We use κ to denote the security parameter. Unless explicitly stated, all algorithms are assumed to take κ as an implicit input, and all definitions hold for sufficiently large $\kappa \in \mathbb{N}$. Throughout, we use the symbol $\perp$ to indicate invalidity; specifically, if an algorithm outputs $\perp$, it signifies that the execution failed or encountered an error.

The symbol negl denotes a negligible function; that is, a function $f : \mathbb{N} \to \mathbb{N}$ is negligible if, for every positive polynomial p, it holds that $f(n) < 1/p(n)$ for all sufficiently large n. We write $\mathsf{D}(x) := y$ or $y := \mathsf{D}(x)$ to denote the output y produced by a deterministic algorithm D when evaluated on input x. Similarly, the notation $x := \mathsf{var}$ signifies the assignment of the value of var to the variable x. For randomized algorithms, we write $\mathsf{R}(x) \to y$ or $y \leftarrow \mathsf{R}(x)$ to denote the evaluation of R on input x, producing output y. The randomness used by a randomized algorithm R can be made explicit as $\mathsf{R}(x; r) := y$, where r is the random string utilized by R.

A sequence of values $(x_1, x_2, \ldots)$ is denoted in vector form as $\mathbf{x}$, with its i-th element represented by $\mathbf{x}[i]$. The length of the vector $\mathbf{x}$ is denoted by $|\mathbf{x}|$. A list is treated as an ordered collection, and its i-th element is similarly denoted by $L[i]$. In this work, lists and vectors are used interchangeably. The concatenation of two strings a and b is denoted either by $(a \| b)$ or (a, b). If a variable x takes a value from the set $\{a, b\}$, then $\bar{x}$ denotes the complementary value in this set; specifically, if $x = a$ (respectively, $x = b$), then $\bar{x} = b$ (respectively, $\bar{x} = a$).

We denote the possession of a private value x by party j using the notation $[j : x]$. For a protocol π, the notation $[j : z'] \leftarrow \pi([i : (x, y)], [j : z], c)$ indicates that party i holds private inputs x and y, party j holds a private input z, no other parties hold private inputs, c is a public input, and upon completion, party j receives a private output z'. The notation $[i : x_i]_{\forall i \in S}$ or, equivalently, $[\![\mathbf{x}]\!]_S$ is used to denote that each party $i \in S$ holds a private value x_i.

On our communication model and protocol structure. All protocols in this work are executed over *secure* and *authenticated* channels. Each protocol proceeds in two rounds: an initiating

party (typically called the client for that execution) sends messages to a set of other parties (referred to as the servers); each server processes its received message and returns a response to the client in the second round; the client then aggregates these responses to compute the final output. It is important to note that servers do not engage in direct communication with each other during a protocol execution. Nonetheless, the definitions and protocol notation presented here are sufficiently general to accommodate alternative communication patterns and protocol structures.

2.2 Security games and oracles

In describing the security games, we adopt a concise and structured pseudocode notation based on a simple command set. We use standard `if--then--else` statements for branching, and `for` to denote loops, with nested branching indicated through consistent indentation for readability.

The command `set` is employed for initializing, updating, or assigning values to variables, while `run` is used to invoke or execute a specified algorithm or protocol. Random sampling is denoted using the `uniform` command; for instance, `uniform m` indicates that a uniform random element from the domain of m is sampled and assigned to m.

To impose conditions on previously defined variables, we use the `require` command. If the specified condition holds, the execution proceeds further; otherwise, the experiment aborts at this point. For brevity, such aborts are left implicit in our algorithm descriptions but can be made explicit by incorporating suitable flags or control mechanisms when needed.

Finally, all variables—including counters, flags, and lists—initialized within the security game are considered global. This allows them to be accessed and modified by any oracle or subroutine invoked during the execution of the experiment.

2.3 Well Known Tools

Here, we present two fundamental tools in cryptography, specifically within the context of threshold cryptography.

2.3.1 Bilinear Pairing

The scheme we discussed in this dissertation is built upon the notion of bilinear pairings. Adopting the conventions from [PBC], we consider three cyclic groups G_0, G_1, G_T, each of prime order p, along with an efficiently computable bilinear map $e : G_0 \times G_1 \to G_T$. This pairing is assumed to be bilinear and non-degenerate.

We base the security of our construction on the *Bilinear Decisional Diffe-Helman (BDDH)* assumption. Specifically, given generators $g_0 \in G_0^*$ and $g_1 \in G_1^*$, and elements $g_0^a, g_1^a, g_0^b, g_1^c$ for randomly chosen exponents $a, b, c \in \mathbb{Z}_p$, it is computationally infeasible to distinguish the value

$e(g_0, g_1)^{abc}$ from a uniformly random element in G_T.

2.3.2 Secret Sharing

We employ Shamir's threshold secret sharing mechanism in our construction.

Definition 2.3.1 (Shamir's Secret Sharing)**.** Let p be a prime number. An (n, t, p, s)-Shamir secret sharing scheme is a randomized procedure SSS that, given parameters n, t, p, s where $0 < t \leq n < p$ and $s \in \mathbb{Z}_p$, produces n shares $s_1, \ldots, s_n \in \mathbb{Z}_p$ satisfying the following properties for any subset of indices $\{i_1, \ldots, i_\ell\}$:

- If $\ell \geq t$, there exist fixed (i.e., input-independent) coefficients $\lambda_1, \ldots, \lambda_\ell \in \mathbb{Z}_p$, known as Lagrange interpolation coefficients, such that:

$$\sum_{j=1}^{\ell} \lambda_j s_{i_j} = s \pmod{p}.$$

- If $\ell < t$, the tuple $(s_{i_1}, \ldots, s_{i_\ell})$ is uniformly distributed over $\mathbb{Z}_p^\ell$, revealing no information about s.

In practice, the scheme operates by selecting random coefficients $a_1, \ldots, a_{t-1} \xleftarrow{\$} \mathbb{Z}_p$, and defining a polynomial $f(x) = s + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$ over $\mathbb{Z}_p$. Each share s_i is computed as $f(i)$ for $i \in [n]$.

3

Literature Survey

In this chapter, we provide a comprehensive review of the Amortized Threshold Symmetric-key Encryption (ATSE) scheme as introduced by Christodorescu et al. (2021). We begin by examining the fundamental building blocks underpinning ATSE, namely the Flexible Threshold Key-derivation (FTKD) primitive and the Merkle tree-based group commitment mechanism. Section 3.1 presents the formal definitions and construction of the FTKD primitive, establishing its role in enabling threshold-based key derivation within the scheme. Section 3.2 introduces the concept of group commitment, followed by a detailed exposition of the Merkle tree group commitment scheme, which serves as a critical component for efficient and secure aggregation of commitments. In Section 3.3, we first outline the overarching cryptographic framework that contextualizes the ATSE construction. Subsequently, we describe the detailed construction of the ATSE scheme itself and provide its security guarantees.

3.1 Flexible Threshold Key Derivation

Here we discuss the framework of FTKD. First, we give formal definitions in Section 3.1.1. Next, we present a bilinear pairing construction in Section 3.1.2 and contend that it meets our definitions. *Motivation* FTKD emerged from the need to securely manage cryptographic keys in distributed environments, particularly as organizations faced the challenge of encrypting massive datasets without relying on a single trusted party. Traditional threshold cryptography allowed secret keys to be split among multiple servers, requiring collaboration for cryptographic operations, but these approaches often involved heavy communication and computation overhead for each operation, limiting scalability and efficiency[CGMS21]. The motivation for FTKD was to enable more efficient, scalable, and flexible key management, allowing parties to derive pseudorandom keys in various modes and granularities, such as deriving group-specific or message-specific keys, while maintaining threshold security guarantees[CGMS21]. This flexibility is crucial for modern enterprise KMS, where different clients may need to derive keys with different access rights or levels of granularity, and where no single party should hold the entire key at any time. FTKD draws on concepts like Distributed PRFs and constrained PRFs[BW13], extending them to allow threshold-based, interactive key derivation that supports both privacy and authenticity.

3.1.1 Definition

The Flexible Threshold Key-derivation (FTKD) framework enables clients to derive cryptographic keys at different levels of granularity: either a complete key associated with the full input or a partial key linked to a subset of the input. In this construction, a partial key can be combined with the complementary portion of the message to locally reconstruct the same full key.

This concept can be interpreted as a distributed analogue of the left/right constrained pseudorandom function (PRF) model introduced in [BW13]. Specifically, for any input pair $x\|y$, it is possible to generate constrained keys k_{x*} and k_{*y} such that k_{x*} (respectively, k_{*y}) can be combined with y (respectively, with x) to compute the value $PRF_k(x\|y)$. Furthermore, the function $PRF_k(x, \cdot)$ remains pseudorandom as long as queries involving inputs $x' \neq x$ are made, even in the presence of a constrained key k_{x*}.

We now formalize the main definition of the FTKD scheme, presenting it as a system comprising both interactive protocols and non-interactive algorithms. In particular, the notation for the DKdf protocol is designed to ensure that only a designated party obtains an output. Within this framework, a party—acting as the client during a protocol execution—may issue three types of queries: a complete input $x\|y$, the left segment x, or the right segment y. The DKdf key-derivation protocol operates according to the nature of the client's query. Depending on the query type, the recipient parties, acting as servers in this context, perform appropriate computations. Upon receiving their responses, the client can aggregate them to derive either a partial key or the complete key, subject to the requirement that a minimum threshold of participating parties (including the client) contribute until the protocol concludes.

Definition 3.1.1. The tuple of algorithms or protocols that comprise up a FTKD scheme have the following form:

$\mathsf{Setup}(1^\kappa, n, t) \to (\llbracket sk \rrbracket_{[n]}, pp)$ is a randomized algorithm that, given a threshold t ($\leq n$) and the total number of participants n, produces n keys $\llbracket sk \rrbracket_{[n]} := \{sk_1, sk_2, ..., sk_n\}$ and public parameters pp. Party s receives the s-the secret key, sk_s. We assume that security parameter and pp are implicitly taken as additional inputs by each of the subsequent algorithms.

$\mathsf{DKdf}(\llbracket sk \rrbracket_{[n]}, \mathcal{S}, [j : \rho, \mathcal{S}]) \to [j : k/\perp]$. Through the DKdf protocol, a party j interfaces with parties in the set S. Based on the request $\rho \in (x, `left'), (y, `right'), ((x, y), `whole')$, the party j receives a key $k \in \{lk, rk, wk\}$ (or $\perp$ on failure). The left key is denoted by lk, the right by rk, and the complete key by wk. DKdf only provides a private output to party j.

$\mathsf{WKGen}(v, \rho) \to wk$. A whole-key wk is deterministically generated by this procedure on input $(v, \rho) \in \{ (\mathrm{lk}, y), (\mathrm{rk}, x) \}$.

FTKD is a core cryptographic primitive introduced to enable efficient and secure key management in distributed settings, particularly for large-scale data encryption in enterprise environments [CGMS21]. FTKD allows a set of servers, each holding a share of the secret key, to collaboratively derive pseudorandom keys in a threshold manner such that the key material is never reconstructed or exposed in the clear. This property is crucial to protect sensitive data against server compromise and to meet compliance requirements in modern KMS [AMMR18].

The security of FTKD is formalized through three main notions: *pseudorandomness*, *correctness*, and overall *security* (robustness against adversarial attacks). These notions are motivated by both practical deployment needs and foundational cryptographic principles.

Pseudorandomness ensures that even if an adversary corrupts up to $t-1$ out of n servers, any key derived via FTKD is computationally indistinguishable from a uniformly random value, unless the adversary interacts with a threshold set of honest servers. This property is essential for guaranteeing the confidentiality of the derived keys and, by extension, the privacy of encrypted data[CGMS21, BW13].

Definition 3.1.2 (Pseudorandomness of FTKD)**.** An FTKD scheme is *pseudorandom* if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible:

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

where the game $\mathsf{DP\text{-}PR}_{\mathcal{A}}(1^\kappa, b)$ is defined below that runs the oracles $O_{\text{pr-kd}}, O_{\text{pr-chal}}$.

$\mathrm{DP\text{-}PR}_{\mathcal{A}}(1^\kappa, b)$ Game

1. **set** $\mathsf{CHAL} \leftarrow 0$, $\mathsf{ABORT} \leftarrow 0$.
2. **set** $g_{\mathsf{lk}}, g_{\mathsf{rk}}, g_{\mathsf{wk}} \leftarrow \emptyset$ and $x^*, y^*, z^* \leftarrow \perp$.
3. **set** counters $LCT_x, RCT_y, WCT_z \leftarrow 0$ for all (x, y, z).
4. **run** $([\![sk]\!]_{[n]}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\kappa, n, t)$.
5. **run** $\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$; **require** $\mathcal{C} \subseteq [n]$ and $|\mathcal{C}| < t$.
6. **run** $i' \leftarrow \mathcal{A}^{O_{\text{pr-kd}}, O_{\text{pr-chal}}}(\{sk_p\}_{p \in \mathcal{C}})$.
7. **if** $\mathsf{ABORT} = 1$ **then return** i'; **else return uniform** i'.

Oracle $O_{\text{pr-kd}}(j, r, \mathcal{S})$:

1. **require** $j \in \mathcal{S}$ and $r \in \{(x, \text{left}), (y, \text{right}), (z, \text{whole})\}$.
2. **run** $[j : op] \leftarrow \mathsf{DKdf}([\![sk]\!]_{[n]}, [j : r, \mathcal{S}])$.
3. **if** $j \notin \mathcal{C}$ **then return** op.
4. **if** $j \in \mathcal{C}$ **then:**
   - **if** $r = (x, \text{left})$ **then do:**
     - **set** $LCT_x \leftarrow LCT_x + |\mathcal{S} \setminus \mathcal{C}|$.
     - **if** $LCT_x \geq t - |\mathcal{C}|$
       - **if** $\mathsf{CHAL} = 1$ **and** $x = x^*$ **then set** $\mathsf{ABORT} = 1$.
       - **else set** $g_{\mathsf{lk}} \leftarrow g_{\mathsf{lk}} \cup \{x\}$.

- Similar for (y, right) with RCT_y and g_{rk}.
- Similar for (z, whole) with WCT_z and g_{wk}.

ORACLE $O_{\text{pr-chal}}(j^*, z^*, \mathcal{S}^*)$:

1. **require** $j^* \in \mathcal{S}^* \setminus \mathcal{C}$, $|\mathcal{S}^*| \geq t$, and $\mathsf{CHAL} = 0$.
2. **set** $\mathsf{CHAL} = 1$ and $(x^*, y^*) := z^*$.
3. **if** $z^* \in g_{\mathsf{wk}}$ or $y^* \in g_{\mathsf{rk}}$ or $x^* \in g_{\mathsf{lk}}$ **then set** $\mathsf{ABORT} = 1$
   else run $[j^* : wk^*] \leftarrow \mathsf{DKdf}([\![sk]\!]_{[n]}, [j^* : z^*, \mathcal{S}^*])$
   - **if** $wk^* = \perp$ **then return** $\perp$
   - **else do:**
     - **if** $b = 0$, **return** wk^*;
     - **if** $b = 1$, **return** uniform random value.

Correctness guarantees that any honest client, interacting with at least t honest servers, will always derive the same key as specified by the protocol, regardless of the adversary's behavior. This prevents denial-of-service attacks or data loss due to malicious servers returning inconsistent or incorrect results, which is a critical requirement for reliable enterprise encryption systems [AMMR18].

Definition 3.1.3 (Correctness of FTKD)**.** An FTKD scheme satisfies *correctness* if for all $([\![sk]\!]_{[n]}, \mathsf{pp}) \leftarrow \mathsf{K.Setup}(1^\kappa, n, t)$, all $j \in [n]$, all valid inputs r, and all $\mathcal{S} \subseteq [n]$ with $|\mathcal{S}| \geq t$ and $j \in \mathcal{S}$, we have the following:

$$\Pr\left[(\mathsf{DKdf}([\![sk]\!]_{[n]}, [j : r, \mathcal{S}]) = k) \wedge (\mathsf{DKdf}^\dagger([\![sk]\!]_{[n]}, [j : r, \mathcal{S}]) = k)\right] = 1$$

where $\mathsf{DKdf}^\dagger$ denotes a deterministic evaluation of the protocol.

DP-Correct$_{\mathcal{A}}(1^\kappa)$ GAME:

1. **set** $\mathsf{CHAL} \leftarrow 0$, $\mathsf{OUT} \leftarrow 0$.
2. **run** $([\![sk]\!]_{[n]}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\kappa, n, t)$.
3. **run** $\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$; **require** $\mathcal{C} \subseteq [n]$, $|\mathcal{C}| < t$.
4. **run** $\mathcal{A}^{O_{\text{cr-kd}}, O_{\text{cr-chal}}}(\{sk_p\}_{p \in \mathcal{C}})$.
5. **return** OUT.

ORACLE $O_{\text{cr-kd}}(j, r, \mathcal{S})$:

1. **require** $j \in \mathcal{S}$ and $r \in \{(x, \text{left}), (y, \text{right}), (z, \text{whole})\}$.
2. **run** $[j : op] \leftarrow \mathsf{DKdf}([\![sk]\!]_{[n]}, [j : r, \mathcal{S}])$.
3. **if** $j \notin \mathcal{C}$ **then return** op.

ORACLE $O_{\text{cr-chal}}(j^*, \mathsf{Type}, z^*, \mathcal{S}^*)$:

1. **require** $\mathsf{OUT} = 0$, $\mathsf{CHAL} = 0$, $\mathsf{Type} \in \{\text{left}, \text{right}, \text{whole}\}$, $j^* \in \mathcal{S}^* \setminus \mathcal{C}$, $|\mathcal{S}^*| \geq t$.
2. **set** $\mathsf{CHAL} = 1$, and $(x^*, y^*) := z^*$.
3. **if** $\mathsf{Type} = \text{left}$:
   - **run** $[j^* : lk] \leftarrow \mathsf{DKdf}([\![sk]\!]_{[n]}, [j^* : (x^*, \text{left}), \mathcal{S}^*])$.
   - **if** $lk \neq \perp$ **then set** $wk^* := \mathsf{WKGen}(lk, y^*)$.
4. **else if** $\mathsf{Type} = \text{right}$:
   - **run** $[j^* : rk] \leftarrow \mathsf{DKdf}([\![sk]\!]_{[n]}, [j^* : (y^*, \text{right}), \mathcal{S}^*])$.
   - **if** $rk \neq \perp$ **then set** $wk^* := \mathsf{WKGen}(x^*, rk)$.
5. **else if** $\mathsf{Type} = \text{whole}$:
   - **run** $[j^* : wk^*] \leftarrow \mathsf{DKdf}([\![sk]\!]_{[n]}, [j^* : (z^*, \text{whole}), \mathcal{S}^*])$.
6. **run** $wk^\dagger \leftarrow \mathsf{DKdf}^\dagger([\![sk]\!]_{[n]}, [j^* : (z^*, \text{whole}), \mathcal{S}^*])$.
7. **if** $wk^* = \perp$ **then set** $\mathsf{OUT} = 0$; **else set** $\mathsf{OUT} = (wk^* \neq wk^\dagger)$.

Definition 3.1.4 (Security of FTKD)**.** An FTKD scheme is *secure* if it satisfies both pseudorandomness and correctness as defined above, and additionally, for any PPT adversary $\mathcal{A}$ corrupting fewer than t parties, the adversary cannot distinguish derived keys from random, nor cause honest parties to derive inconsistent keys, except with negligible probability.

3.1.2 A FTKD Construction

TOOLS

- Let $G_0 = \langle g_0 \rangle$, $G_1 = \langle g_1 \rangle$, and $G_T = \langle g \rangle$ be cyclic multiplicative groups of prime order q. Assume the existence of an efficiently computable, non-degenerate bilinear pairing

$$\mathbf{e} : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}.$$

- Two cryptographic hash functions modeled as random oracles:

$$H_0 : \{0,1\}^* \to \mathbb{G}_0, \quad H_1 : \{0,1\}^* \to \mathbb{G}_1.$$

- Let $\mathsf{SSS}(sk, n, t) \to (\llbracket sk \rrbracket_{[n]}, pp_s)$ represent a Shamir's Secret Sharing scheme.

SCHEME

- $\mathsf{FTKD.Setup}(1^\lambda, n, t) \to (\llbracket sk \rrbracket_{[n]}, \mathrm{pp})$: Sample a random value $sk \xleftarrow{\$} \mathbb{Z}_q$; then execute

$$(\llbracket sk \rrbracket_{[n]}, pp_s) \leftarrow \mathsf{SSS}(sk, n, t).$$

  Set system parameters as

$$\mathrm{pp} := (g_0, G_0, g_1, G_1, g, G_T, \mathbf{e}, H_0, H_1, pp_s).$$

- $\mathsf{DKdf}(\llbracket sk \rrbracket_{[n]}, [j : \rho, \mathcal{S}]) \to [j : k/\perp]$: A two-round distributed protocol defined as follows:

  – *Round-1.* The party j broadcasts the query ρ to all parties p in $\mathcal{S}$.

  – *Round-2.* Each participant p in $\mathcal{S}$ computes a response k_p based on the query type:

$$r := \begin{cases} H_0(a) & \text{if } \rho = (a, \text{'left'}) \\ H_1(b) & \text{if } \rho = (b, \text{'right'}) \\ \mathbf{e}(H_0(a), H_1(b)) & \text{if } \rho = ((a,b), \text{'whole'}) \end{cases}$$

  and sends $k_p = r^{sk_i}$ back to j.

  Finalize. If j receives at least $t-1$ valid responses from $\mathcal{S}$, it computes

$$k := \prod_{i \in \mathcal{S}'} k_i^{\lambda_{i,\mathcal{S}'}}$$

  for a subset $\mathcal{S}' \subseteq \mathcal{S} \cup \{j\}$ of size t; otherwise, it outputs $\perp$.

- $\mathsf{WKGen}(u, \sigma) \to \mathrm{wk}$: Compute the witness key as

$$\mathrm{wk} := \begin{cases} \mathbf{e}(H_0(\sigma), u) & \text{if } \sigma = a \\ \mathbf{e}(u, H_1(\sigma)) & \text{if } \sigma = b \end{cases},$$

Theorem 3.1 ([CGMS21]). *Let* FTKD *be the flexible threshold key-derivation scheme constructed above. Then, under the Bilinear Decisional Diffie-Hellman (BDDH) assumption and in the random oracle model,* FTKD *satisfies both correctness and pseudorandomness.*

Overall Security of FTKD encompasses both of the above: it ensures that the scheme is robust against adversaries who corrupt fewer than t servers, cannot distinguish derived keys from random, and cannot cause honest parties to derive inconsistent keys except with negligible

probability [CGMS21]. Based on a distributed version of left/right constrained PRFs [BW13], the FTKD construction in [CGMS21] has been shown to be secure under the Bilinear Bilinear Decisional Diffe-Helman (BDDH) assumption in the random oracle model.

These security notions are not only theoretically sound, but also address the operational realities of large-scale, distributed key-management deployments, as highlighted by recent efforts in both academia and industry [AMMR18, CGMS21].

Proof. See appendix 5.1 in Christodorescu et al. (2021).

3.2 Group Commitment

Motivation. In modern threshold cryptographic systems, especially those designed to encrypt large datasets such as ATSE, it is necessary to efficiently and securely bind a set of messages (or data items) to a single compact cryptographic object. Traditional commitment schemes are designed for individual messages, but when encrypting or authenticating a whole group of messages in one batch to achieve amortization and efficiency, a more powerful primitive is needed, the Group Commitment Scheme.

Introduction. A group commitment scheme allows a client to generate a short commitment that cryptographically binds an entire group (or vector) of messages. This commitment ensures that no adversary can later open the commitment to a different set of messages (binding), while still allowing efficient verification and selective opening of individual messages (hiding and extractability). In the context of ATSE, group commitments are crucial for enabling the privileged encryptor to commit to a large batch of messages in a single interaction, ensuring that the derived group key and all subsequent ciphertexts are tightly linked to the committed dataset. This prevents malicious clients from forging ciphertexts or tampering with the batch after the commitment phase, thus preserving both authenticity and integrity at scale [AMMR18]. The group commitment scheme in this work is typically instantiated using Merkle tree techniques [Mer88], which offer efficient commitment and verification even for large groups, and support the selective opening of individual messages without revealing the entire dataset. This makes group commitments a foundational tool for scalable, secure, and auditable encryption in enterprise and cloud settings.

3.2.1 Definition

Definition 3.2.1. A *group commitment scheme* consists of a tuple of PPT algorithms $(\mathsf{GSetup}, \mathsf{GCommit}, \mathsf{CardVer}, \mathsf{GVer})$ with the following syntax:

- $\mathsf{GSetup}(1^\kappa) \to \mathsf{pp}$: The setup algorithm outputs public parameters pp, including a description of the hash function H.
- $\mathsf{GCommit}(\mathsf{pp}, \mathbf{m}) \to (v, \mathbf{x}, \mathbf{f})$: Commits to a message vector $\mathbf{m}$, returning a group commitment v, unique commitment paths $\mathbf{x}$, and openings $\mathbf{f}$.

- $\mathsf{CardVer}(\mathsf{pp}, (v, \mathbf{p})) \in \{0,1\}$: Verifies if the committed cardinality matches $\mathbf{p}$.
- $\mathsf{GVer}(\mathsf{pp}, (v, x), (m, f)) \in \{0,1\}$: Verifies whether a message-opening pair matches the commitment and its unique path.

To see the security properties (binding, hiding, correctness, and cardinal binding) of the group commitment scheme defied above, we refer section 6 of the main article [CGMS21]. Now we give the group commitment construction from Merkle tree.

3.2.2 The Merkle Tree Group Commitment Construction

This construction uses a Merkle tree to commit to a vector of messages compactly. Each message is hashed at the leaf, internal nodes combine hashes and labels, and the root forms the group commitment.

- $\mathsf{GSetup}(1^\kappa)$: Sample a hash function $H : \{0,1\}^* \to \{0,1\}^\kappa$ and output its description as the public parame- ters $pp := H$.
- $\mathsf{GCommit}(pp, \mathbf{m})$: Let $\mathbf{p} = |\mathbf{m}|$, $q = \lceil \log_2 \mathbf{p} \rceil$, and sample $\mathbf{p}$ random values $w_1, \ldots, w_{\mathbf{p}}$. Compute leaf hashes h_0^s and labels v_0^s for $s = 1 \to 2^q$:

$$h_0^s := \begin{cases} H(s \,\|\, \mathbf{m}_s \,\|\, w_s) & \text{if } s \le \mathbf{p}, \\ H(s \,\|\, 0^\lambda) & \text{if } s > \mathbf{p} \end{cases}$$

$$v_0^s := \begin{cases} 1 & \text{if } s \le \mathbf{p}, \\ 0 & \text{if } s > \mathbf{p} \end{cases}$$

For levels $r = 1 \to q$, compute:

$$h_r^s := H\left(v_{r-1}^{2s-1} \,\|\, h_{r-1}^{2s-1} \,\|\, v_{r-1}^{2s} \,\|\, h_{r-1}^{2s}\right)$$

$$v_r^s := v_{r-1}^{2s-1} + v_{r-1}^{2s}$$

Define the group commitment:

$$v := (\mathbf{p}, h_q^1)$$

The opening for position s:

$$f_s := (s, w_s)$$

and the unique commitment path:

$$x_s := \left((h_0^{\mathsf{sib}(s,0)}, \ldots, h_{q-1}^{\mathsf{sib}(\lceil s/2^r \rceil, r)}), (v_0^{\mathsf{sib}(s,0)}, \ldots, v_{q-1}^{\mathsf{sib}(\lceil s/2^r \rceil, r)})\right)$$

- $\mathsf{GVer}(pp, (v, x), (m, f))$: Parse f as (s, w), x as sequences of hashes and labels, v as $(\mathbf{p}, h)$.

Compute:

$$h_0 := H(s \,\|\, m \,\|\, w), \quad v_0 := 1$$

Let u be the binary representation of s, with $u[r]$ its r-th bit.

For $r = 1 \to q$:

$$\text{if } u[r] = 0 : \begin{cases} h_r := H(v_{r-1} \,\|\, h_{r-1} \,\|\, w_{r-1} \,\|\, n_{r-1}) \\ v_r := v_{r-1} + w_{r-1} \end{cases}$$

$$\text{else: } \begin{cases} h_r := H(w_{r-1} \,\|\, n_{r-1} \,\|\, v_{r-1} \,\|\, h_{r-1}) \\ v_r := w_{r-1} + v_{r-1} \end{cases}$$

Accept if $h_q = h$ and $v_q = \mathbf{p}$.

- $\mathsf{CardVer}(pp, (v, \mathbf{p}'))$: Parse v as $(\mathbf{p}, h)$. Return 1 if $\mathbf{p} = \mathbf{p}'$, else 0.

Theorem 3.2 ([CGMS21]). *Let H be a random oracle. Then the construction described above satisfies the security properties of group commitment. Hence a valid group commitment scheme.*

Proof. See section 6.1 in Christodorescu et al. (2021) for proof.

3.3 Amortized Threshold Symmetic-key Encryption

In this section we precisely introduce ATSE. Prior to presenting ATSE scheme based on FTKD due to Christodorescu et al. [CGMS21], we first give the formal definition and security notions.

3.3.1 Definition

Definition 3.3.1. A trio of algorithms and protocols (Setup, DGEnc, and DKdf) provide an ATSE that satisfies the consistency property stated below.

– $\mathsf{Setup}(1^\kappa, n, t) \to (\llbracket sk \rrbracket_{[n]}, \mathrm{pp})$:
Setup is a randomized algorithm that generates public parameters pp and N secret keys $sk_1, \ldots, \mathrm{sk}_n$. Party i receives the i-th secret key, sk_i. The following protocols and algorithms all take pp as an implicit input.

– $\mathsf{DGEnc}(\llbracket sk \rrbracket_{[n]}, [j : \mathbf{m}, \mathcal{S}]) \to [j : \mathbf{c}/\ \perp]$: DGEnc is a distributed group-encryption protocol which enables a party j to encrypt any vector of messages $\mathbf{m} = (m_1, m_2, \ldots, m_N)$ to create a vector of ciphertexts $\mathbf{c} = (c_1, c_2, \ldots, c_N)$, or $\perp$ in the event that it fails.

– $\mathsf{DistDec}(\llbracket sk \rrbracket_{[n]}, [j : c, \mathcal{S}]) \to [j : m/\ \perp]$ a distributed protocol allow a party j to decrypt a single ciphertext c with the assistance of parties in a set S. j outputs a message m (or $\perp$ to indicate failure) at the conclusion of the protocol.

Consistency: Provided that all participants act honestly, there exists a negligible function negl such that, for any collection of messages $\mathbf{m} = (m_1, \ldots, m_n)$ and any $i \in [N]$, the following probability is at least $1 - \text{negl}(\lambda)$:

$$\Pr\left[[j' : m_i] \leftarrow \text{DistDec}([\![sk]\!]_{[n]}, [j' : c_i, S']) \;\middle|\; \mathbf{c} \leftarrow \text{DGEnc}([\![sk]\!]_{[n]}, [j : \mathbf{m}, S])\right]$$

3.3.2 Message Privacy

In the Chosen-Plaintext Attack (CPA) game, which formalizes message privacy in non-interactive cryptographic systems, adversaries can create encryption queries before and after receiving a challenge ciphertext. This challenge requires distinguishing between encryptions of two adversary-chosen messages. The threshold setting introduces additional complexities by necessitating structured query mechanisms to evaluate message privacy in environments where multiple parties interact during encryption and decryption.

A critical requirement involves ensuring that decryptors cannot infer information about other ciphertexts when decrypting a specific ciphertext — even those within the same group. Simultaneously, encryptors must derive group-specific encryption keys without compromising the privacy of ciphertexts belonging to other groups. Specifically, an encryptor owning keys for one group must remain unable to generate valid encryption keys for groups they do not control, thereby preventing unauthorized ciphertext creation.

In this distributed context, message privacy also demands protection against adversaries attempting to exploit collaborative protocols. We observe that, in addition to the encryption and challenge oracles, the attacker has access to two distinct decryption oracles. Importantly, in the ATSE setting, the outputs of these decryption oracles are not directly returned to the adversary. This models realistic adversarial capabilities in distributed protocols where an adversary may corrupt some servers and attempt to gain information indirectly from protocol transcripts or execution patterns.

In the case of the ATSE scheme, Christodorescu et al. formalize these considerations by allowing two decryption oracles alongside a single encryption oracle within the security game. This setting accounts for the following adversarial scenarios:

- If an honest party is under the influence or partial control of an adversary, it may be coerced to generate encryption queries of adversary-chosen messages.
- Similarly, if an honest party is compromised, it might be manipulated to generate decryption queries for adversary-chosen ciphertexts.
- Finally, the framework ensures that even in such adversarially controlled scenarios, an honest party does not learn any unintended information about other messages — whether from the same encryption group or not — beyond the plaintext it is intended to decrypt.

To rigorously formalize this property, we now define the notion of message privacy for the ATSE scheme in terms of an indistinguishability game. Informally, the game ensures that no

PPT adversary can distinguish between the encryptions of two chosen message vectors with non-negligible advantage, even when granted controlled oracle access as specified.

The formal definition of message privacy for the ATSE scheme is given below, followed by a description of the security game and its associated oracles. The adversary interacts with the system through these oracles to issue encryption, decryption, and challenge queries under well-defined constraints that model the realistic capabilities of adversaries in threshold encryption systems. Here is the formal definition of *message privacy* of the ATSE scheme and the oracles.

Definition 3.3.2. An ATSE scheme satisfies message privacy if for all adversaries of PPT $\mathcal{A}$, there exists a negl such that

$$|\Pr[AT\text{-}MsgPriv_{\mathcal{A}}(1^\kappa, 0) = 1] - P[AT\text{-}MsgPriv_{\mathcal{A}}(1^\kappa, 1) = 1]| < \mathsf{negl}(\kappa)$$

where the **AT-MsgPriv**$_{\mathcal{A}}$ game and the component oracles are described below.

AT-MsgPriv$(1^\kappa, b)$ Game

1. **set** $\mathsf{CHAL} \leftarrow 0$.
2. **set** $c_L \leftarrow \emptyset$.
3. **set** $I \leftarrow \emptyset$.
4. **run** $([\![sk]\!]_{[n]}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\kappa, n, t)$.
5. **run** $C \leftarrow \mathcal{A}(\mathsf{pp})$.
6. **require** $C \subseteq [n]$ and $|C| < t$.
7. **run** $i' \leftarrow \mathcal{A}^{O_{\text{at-mp-enc}}, O_{\text{at-mp-dec}}, O_{\text{at-mp-chal}}, O_{\text{at-mp-pc-dec}}}(\{sk_p\}_{p \in \mathcal{C}})$.
8. **return** i'.

Oracle $O_{\text{at-mp-enc}}(j, \mathbf{m}, \mathcal{S})$:

1. **require** $j \in \mathcal{S}$.
2. **run** $[j : \mathsf{op}] \leftarrow \mathsf{DGEnc}([\![sk]\!]_{[n]}, [j : m, \mathcal{S}])$.
3. **if** $j \in s$ **then return** op.

Oracle $O_{\text{at-mp-dec}}(j, c, \mathcal{S})$:

1. **require** $j \in \mathcal{S} \setminus C$.
2. **run** $[j : \mathsf{op}] \leftarrow \mathsf{DistDec}([\![sk]\!]_{[n]}, [j : c, \mathcal{S}])$.

ORACLE $O_{\text{at-mp-chal}}(j, m_0, m_1, \mathcal{S})$:

1. **require** $j \in \mathcal{S} \setminus C$, $|m_0| = |m_1|$, and $\mathsf{CHAL} = 0$.
2. **set** $\mathsf{CHAL} \leftarrow 1$.
3. **for each** S **such that** $m_0[i] = m_1[i]$, **set** $I \leftarrow I \cup \{i\}$.
4. **run** $[j : \mathsf{op}] \leftarrow \mathsf{DGEnc}(\llbracket sk \rrbracket_{[n]}, [j : m_b, \mathcal{S}])$.
5. **set** $c_L \leftarrow \mathsf{op}$.
6. **return** c_L.

ORACLE $O_{\text{at-mp-pc-dec}}(j, c, \mathcal{S})$:

1. **require** $j \in \mathcal{S}$, $\mathsf{CHAL} = 1$, $c = c_L[i]$, and $i \in I$.
2. **run** $[j : \mathsf{op}] \leftarrow \mathsf{DistDec}(\llbracket sk \rrbracket_{[n]}, [j : c, \mathcal{S}])$.
3. **if** $j \in C$, **return** op.

The first oracle, denoted $O_{\text{at-mp-dec}}$, models situations where an adversary may attempt to decrypt chosen ciphertexts via an honest client. This reflects practical distributed systems where a malicious participant could attempt to influence decryption sessions initiated by honest parties to indirectly learn about the underlying plaintexts or cryptographic keys. The second decryption oracle, $O_{\text{at-mp-pc-dec}}$, is unique to the threshold group encryption setting and captures a more restricted scenario. This oracle is only accessible following the challenge query, and only for those ciphertext components originating from the challenge ciphertext vector c^* that correspond to message components identical in both challenge message vectors $\mathbf{m}_0$ and $\mathbf{m}_1$. This restriction preserves the indistinguishability necessary for defining message privacy in this structured multi-party setting.

3.3.3 Authenticity

We have to make sure the generated ciphertexts are "authentic", which is refer to as ciphertext integrity. Formally stated, authenticity ensures that each legitimate ciphertext must be generated by appropriately interacting with at least one honest server, provided that there are no less than t corruptions.

Direct and indirect encryption searches are allowed, much like in the message privacy game. Since the corrupt party is supposed to learn the ciphertexts produced by the former, they are obviously not regarded as forgeries. But in the indirect scenario, where an honest party starts the encryption, the security game does not provide the adversary access to the ciphertext that is produced. Therefore, if the adversary is successful in obtaining the ciphertext of an indirect encryption question, it can output it as a legitimate forgery. Therefore, even when actively

participating in the protocols, the TSE scheme must make such attacks unpredictable to him in order to prevent them.

Definition 3.3.3. An ATSE scheme satisfies (strong) authenticity if for all PPT adversary $\mathcal{A}$, there exists a negligible function negl such that,

$$\Pr[AT\text{-}Auth_{\mathcal{A}}(1^{\kappa}) = 1] < \mathsf{negl}(\kappa) \quad (\text{or} \quad \Pr[AT\text{-}Str\text{-}Auth_{\mathcal{A}}(1^{\kappa}) = 1] < \mathsf{negl}(\kappa))$$

where the security game AT-Auth and AT-Str-Auth, and the associated encryption, decryption, targeted-decryption and challenge(a.k.a. forgery) oracles are described below respectively.

AT-Auth$_{\mathcal{A}}(1^{\kappa})$ Game

1. **set** $ct := 0$ and $L_c := \emptyset$.
2. **set** $SUCC := 0$ and $CHAL := 0$.
3. **run** $(\llbracket sk \rrbracket_{[n]}, pp) \leftarrow Setup(1^{\kappa}, n, t)$.
4. **run** $C \leftarrow \mathcal{A}(pp)$;
   require $C \subseteq [n]$ and $|C| < t$.
5. **run** oracles:
   - $O_{at-au-enc}$
   - $O_{at-au-dec}$
   - $O_{at-au-tar-dec}$
   - $O_{at-au-chal}(\{sk_p\}_{p \in C})$
6. **return** $SUCC$.

Oracle $O_{at-au-enc}(j, \mathbf{m}, N, \mathcal{S})$:

require $j \in \mathcal{S}$.

1. **run** $[j : op] \leftarrow DGEnc(\llbracket sk \rrbracket_{[n]}, [j : \mathbf{m}, \mathcal{S}])$.
2. **if** $j \notin C$ **then set** $L_c := L_c \cup \{op\}$ **else set**

$$ct := ct + N|\mathcal{S} \setminus C|.$$

Oracle $O_{at-au-dec}(j, c, \mathcal{S})$:

require $j \in \mathcal{S}$.

1. **run** $[j : op] \leftarrow DistDec([\![sk]\!]_{[n]}, [j : c, \mathcal{S}])$.
2. **if** $j \notin C$ **then set**
$$ct := ct + |\mathcal{S} \setminus C|.$$

ORACLE $O_{at-au-tar-dec}(j, i, S)$:

---

require $j \in \mathcal{S} \setminus C$ and $i \in [|L_c|]$.

1. **set** $c := L_c[i]$.
2. **run** $[j : op] \leftarrow DistDec([\![sk]\!]_{[n]}, [j : c, \mathcal{S}])$.

ORACLE $O_{at-au-chal}(L_{forge})$:

---

1. **set** $k := \left\lfloor \frac{ct}{g} \right\rfloor$, where $g := t - |C|$.
2. **set** $((j_1, \mathcal{S}_1, c_1), \ldots, (j_{k+1}, \mathcal{S}_{k+1}, c_{k+1})) := L_{forge}$;;
require $CHAL = 0$ and $SUCC = 0$ and $j_1, j_2, \ldots, j_{k+1} \notin C$ and $\forall i \neq i' : c_i \neq c_{i'}$.
3. **set** $CHAL := 1$.
4. **run**
$$\{op_i \leftarrow DistDec^{\dagger}([\![sk]\!]_{[n]}, [j_i : c_i, S_i])\}_{i \in [k+1]}.$$
5. **if** $\forall i \in [k+1] : op_i \neq \perp$ **thenset** $SUCC := 1$; **else set** $SUCC := 0$.

AT-STR-AUTH$_{\mathcal{A}}(1^{\kappa})$:

---

1. **set** $ct := 0$ and $L_c := \emptyset$.
2. **set** $SUCC := 0$ and $CHAL := 0$.
3. **run** $([\![sk]\!]_{[n]}, pp) \leftarrow Setup(1^{\kappa}, n, t)$.
4. **run** $C \leftarrow \mathcal{A}(pp)$;
require $C \subseteq [n]$ and $|C| < t$.
5. **run** oracles:
   - $O_{at-au-enc}$
   - $O_{at-au-dec}$
   - $O_{at-au-tar-dec}$
   - $O_{at-str-au-chal}(\{sk_p\}_{p \in C})$

6. **return** $SUCC$.

ORACLE $O_{at-str-au-chal}(L_{forge})$:

1. **set** $k := \left\lfloor \frac{ct}{g} \right\rfloor$, where $g := t - |C|$.
2. **set** $((j_1, \mathcal{S}_1, c_1), \ldots, (j_{k+1}, \mathcal{S}_{k+1}, c_{k+1})) := L_{forge}$;
   require $CHAL = 0$ and $SUCC = 0$ and $j_1, j_2, \ldots, j_{k+1} \notin C$ and $\forall i \neq i' : c_i \neq c_{i'}$.
3. **set** $CHAL := 1$.
4. **run** $\{op_i \leftarrow DistDec([\![sk]\!]_{[n]}, [j_i : c_i, \mathcal{S}_i])\}_{i \in [k+1]}$.
5. **if** $\forall i \in [k+1] : op_i \neq \perp$ **then set** $SUCC := 1$; **else set** $SUCC := 0$.

3.3.4 Correctness

In threshold encryption schemes, the correctness property ensures that any ciphertext produced through honest execution of the encryption protocol decrypts to the original message, or signals failure if the ciphertext has been malformed or adversarially tampered with. In the setting of ATSE, this requirement becomes more intricate due to the amortized nature of the scheme and the interactive structure of distributed protocols. Specifically, correctness in ATSE must guarantee that the decryption of a given ciphertext is independent of other ciphertexts within the same encryption batch, even when multiple parties interact during encryption and decryption phases.

Moreover, it is essential to prevent adversarial parties participating in distributed decryption protocols from inferring information about other ciphertexts within the same group or across different groups. This is particularly important in scenarios where a compromised server may attempt to exploit its role in collaborative decryption to gain knowledge beyond the intended plaintext. Additionally, encryptors must be restricted to deriving encryption keys that are specific to their designated message group, ensuring that no encryptor can forge valid ciphertexts for groups outside their authority.

To capture these requirements formally, the ATSE security model incorporates structured decryption oracles within its message privacy game. These oracles reflect realistic adversarial capabilities in distributed systems, such as initiating encryption and decryption queries under adversary-chosen messages or ciphertexts, and participating as a corrupt server in decryption protocols initiated by honest clients. The correctness definition in ATSE thereby ensures both the integrity of individual decryptions and the isolation of ciphertexts across and within encryption batches, preserving the confidentiality and authenticity guarantees of the system in adversarial settings.

Definition 3.3.4 ((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 $\mathsf{negl}(\kappa)$. 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$_{\mathcal{A}}$ 1 with probability at most $\mathsf{negl}(\kappa)$. The security game AT-Correct$_{\mathcal{A}}$ and AT-Str-Correct$_{\mathcal{A}}$, and the corresponding oracles are described as follows.

AT-Correct$_{\mathcal{A}}(1^{\kappa})$ Game :

1. **set** $CHAL := 0$ and $OUT := 0$.
2. **run** $(\llbracket sk \rrbracket_{[n]}, pp) \leftarrow Setup(1^{\kappa}, n, t)$.
3. **run** $C \leftarrow \mathcal{A}(pp)$; **require** $C \subseteq [n]$ and $|C| < t$.
4. **run** oracles:
   - $O_{at-cor-enc}(\{sk_i\}_{\in C})$
   - $O_{at-cor-dec}(\{sk_i\}_{\in C})$
   - $O_{at-cor-chal}(\{sk_i\}_{\in C})$
5. **return** OUT.

Oracle $O_{at-cor-enc}(j, \mathbf{m}, \mathcal{S})$:

require $j \in \mathcal{S}$.

1. **run** $[j : op] \leftarrow DGEnc(\llbracket sk \rrbracket_{[n]}, [j : \mathbf{m}, S])$.
2. **if** $j \notin C$ **then return** op.

Oracle $O_{at-cor-dec}(j, c, \mathcal{S})$:

require $j \in S$.

1. **run** $[j : op] \leftarrow DistDec(\llbracket sk \rrbracket_{[n]}, [j : c, \mathcal{S}])$.
2. **if** $j \notin C$ **then return** op.

Oracle $O_{at-cor-chal}(j, \mathcal{S}, j', \mathcal{S}', \mathbf{m} = (m_1, m_2, \ldots, m_N), i)$:

require $j \in \mathcal{S} \setminus C$, $j' \in \mathcal{S}' \setminus C$, $i \in [N]$, $CHAL = 0$, and $OUT = 0$.

1. **set** $CHAL := 1$.
2. **run** $[j : op] \leftarrow DGEnc(\llbracket sk \rrbracket_{[n]}, [j : \mathbf{m}, \mathcal{S}])$.

3. **if** $op = \perp$ **then set** $OUT := 0$.
4. **else set** $(c_1, \ldots, c_Q) := op$**do:**
   (a) **run** $op' \leftarrow DistDec([\![sk]\!]_{[n]}, [j' : c_{i,\mathcal{S}'}])$.
   (b) **if** $op' = m_i$ **then set** $OUT := 0$; **else set** $OUT := 1$.

AT-Str-Correct$_{\mathcal{A}}(1^\kappa)$ Game :

1. **set** $CHAL := 0$ and $OUT := 0$.
2. **run** $([\![sk]\!]_{[n]}, pp) \leftarrow Setup(1^\kappa, n, t)$.
3. **run** $C \leftarrow \mathcal{A}(pp)$; **require** $C \subseteq [n]$ and $|C| < t$.
4. **run** oracles:
   - $O_{at-cor-enc}(\{sk_i\}_{\in C})$
   - $O_{at-cor-dec}(\{sk_i\}_{\in C})$
   - $O_{at-str-cor-chal}(\{sk_i\}_{\in C})$
5. **return** OUT.

Oracle $O_{at-str-cor-chal}(j, \mathcal{S}, j', \mathcal{S}', \mathbf{m} = (m_1, m_2, \ldots, m_N), i)$:

require $j \in \mathcal{S} \setminus C$, $j' \in \mathcal{S}' \setminus C$, $S_{ind} \in [N]$, $CHAL = 0$, and $OUT = 0$.

1. **set** $CHAL := 1$.
2. **run** $[j : op] \leftarrow DGEnc([\![sk]\!]_{[n]}, [j : \mathbf{m}, \mathcal{S}])$.
3. **if** $op = \perp$ **then set** $OUT := 0$.
4. **else set** $(c_1, \ldots, c_N) := op$ and **do:**
   (a) **run** $op' \leftarrow DistDec^\dagger([\![sk]\!]_{[n]}, [j' : c_i, \mathcal{S}'])$.
   (b) **if** $op' = m_i$ **then set** $OUT := 0$; **else set** $OUT := 1$.

3.3.5 A ATSE Construction

TOOLS

- A simple FTKD scheme: $\mathsf{FTKD} := (\mathsf{FTKD.Setup}, \mathsf{DKdf}, \mathsf{WKGen})$.
- Merkle-tree group commitment scheme: $(\mathsf{GSetup}, \mathsf{GCommit}, \mathsf{CardVer}, \mathsf{GVer})$.
- A PRG with polynomially many output bits.

SCHEME

- $\mathsf{Setup}(1^\kappa, n, t) \to (\mathsf{sk}_{[n]}, pp)$: Run $\mathsf{FTKD.Setup}(n, t)$ to get $((rk_1, \ldots, rk_n), pp_{\mathsf{FTKD}})$, and run $\mathsf{GSetup}(1^\kappa)$ to get pp_{com}. Set $\mathrm{sk}_i := rk_i$ for $i \in [n]$ and $pp := (pp_{\mathsf{FTKD}}, pp_{\mathsf{com}})$.
- $\mathsf{DGEnc}(\mathsf{sk}_{[n]}, [\mathsf{j} : \mathbf{m}, \mathcal{S}]) \to [j : c/\ \perp]$: Let $N := |\mathbf{m}|$. This is an interactive protocol initiated by the client (party-j):
  - Party-j computes $(\gamma, \mathbf{q}, \mathbf{p}) \leftarrow \mathsf{GCommit}(\mathsf{pp}_{\mathsf{com}}, \mathbf{m})$, where $\mathbf{p} = (p_1, \ldots, p_{\mathsf{N}})$ and $\mathbf{q} = (q_1, \ldots, q_N)$.
  - Parties in $\mathcal{S}$ interactively run DKdf to derive the group-specific left-key:

    $$[\mathsf{j} : \mathsf{gk}] \leftarrow \mathsf{DKdf}(\mathrm{sk}_{[n]}, [\mathsf{j} : (\mathsf{N}, (\mathsf{j} \| \gamma), \text{`left'})], \mathcal{S}, \mathsf{pp}).$$

    The client party-j sends $(\mathsf{j} \| \gamma \| \mathsf{N})$ to servers in $\mathcal{S}$. A server verifies (i) whether it is sent by j and (ii) whether $\mathsf{CardVer}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{S}, \mathbf{q})) = 1$; if either fails, it returns $\perp$ and aborts. Otherwise, it proceeds. If $\mathsf{gk} = \perp$, party-j outputs $\perp$, else proceeds.
  - Finally, for each $i \in [N]$, party-j:
    - Computes the message-specific whole-key:

      $$\mathsf{wk}_i := \mathsf{WKGen}(\mathsf{gk}, q_i).$$

    - Generates the ciphertext:

      $$a_i := (\mathsf{j}, \gamma, q_i, u_i)$$

      where

      $$u_i := \mathsf{PRG}(\mathsf{wk}_i) \oplus (m_i \| p_i).$$

  - Output the ciphertext tuple:

    $$c := (a_1, \ldots, a_{\mathsf{N}}).$$

- $\mathsf{DistDec}(\mathsf{sk}_{[n]}, [\mathsf{j} : c, \mathsf{S}]) \to [\mathsf{j} : \mathbf{m}/\ \perp]$: This is an interactive protocol initiated by party-j:
  - Party-j parses each a_i in c as $(\mathsf{j}, \gamma, q_i, u_i)$.
  - Parties in $\mathcal{S}$ interactively run DKdf:

    $$[\mathsf{j} : \mathsf{wk}_i] \leftarrow \mathsf{DKdf}(\mathrm{sk}_{[n]}, [\mathsf{j} : ((\mathsf{j}\|\mathsf{S}, q_i), \text{`whole'})], \mathcal{S}, \mathsf{pp}).$$

    Party-j receives wk_i; if any $\mathsf{wk}_i = \perp$, it outputs $\perp$ and aborts.
  - Decrypt:

    $$(m_i \| p_i) := \mathsf{PRG}(\mathsf{wk}_i) \oplus u_i.$$

  - Verify commitment:

    $$\text{if } \mathsf{GVer}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{S}, q_i), (m_i, p_i)) = 0, \text{ then output } \perp .$$

    If all checks pass, output $\mathbf{m} := (m_1, \ldots, m_N)$.

(Sec 7.2 in Christodorescu et al. 2021)

Theorem 3.3 ([CGMS21]). *The above ATSE scheme described is (strongly-) secure if the underlying simple FTKD is (strongly-)secure.*

Proof. See appendix E.2 of [CCMS21] for a detailed proof of Corretness, Message Privacy and Authenticity of ATSE. □

4

ATSE for Tabular Data

This chapter presents the adaptation of the ATSE scheme for efficient encryption of large tabular datasets. Motivated by the inefficiencies of traditional threshold symmetric-key encryption schemes when applied to fine-grained data, the proposed approach enables a privileged client to encrypt entire tables with a single interaction with key servers, while maintaining strong privacy, authenticity, and fine-grained access control.

Section 4.1 formalizes the ATSE scheme for matrix data, extending the vector-based definitions. Section 4.1.2 adapts the security notions—message privacy, correctness, and authenticity—to the tabular setting. Section 4.1.3 details the construction, leveraging FTKD, Merkle tree group commitments, and a PRG. We conclude with proofs demonstrating that the scheme's security follows from the underlying primitives, establishing ATSE as a scalable and secure solution for threshold encryption of tabular data.

4.1 ATSE for Tabular Data

Motivation. TSE schemes, such as DiSE, provide strong security to encrypt sensitive data by distributing key material across multiple servers and requiring server interaction for each encryption or decryption operation. However, this approach becomes inefficient when handling large tabular datasets, as repeated interactions for each cell or row introduce significant latency and computational overhead.

ATSE scheme for tabular data addresses this inefficiency by allowing a privileged client, such as a data ingestion pipeline, to encrypt entire tables or large blocks of data with just a single round of interaction with the key servers. Each cell in the table is encrypted using a unique, locally derived key, while the scheme maintains the same strong authenticity and privacy guarantees as traditional TSE for individual decryptors, who must still interact with the servers for each decryption. This makes ATSE especially suitable for enterprise applications where large volumes of tabular data need to be encrypted efficiently, while still enforcing fine-grained access control and auditability for downstream analytics.

In summary, ATSE for tabular data delivers the security of distributed threshold encryption with the performance and scalability required for modern, data-intensive environments.

4.1.1 Definition

The definition of ATSE scheme for matrix (or tabular) data is the same as the definition of ATSE scheme for a message vector except to replace the message vector $\mathbf{m} = (m_1, m_2, \ldots, m_N)$ by the matrix $\mathbf{M} = (m_{ij})$. So, we did not rewrite the same definition again to make it simple and compact.
Consistency: If all parties behave honestly, there exists a negligible function negl for which the following probability is at least $1 - \mathsf{negl}(\kappa)$ for any set of messages $\mathbf{M} = (m_{ij})_{r\times s}$, any $i \in [r]$ & $j \in [s]$:

$$\Pr\left[[k' : m_{ij}] \leftarrow \text{DistDec}([[sk]]_{[n]}, [k' : c_{ij}, \mathcal{S}']) \;\middle|\; \mathbf{c} = (c_{ij} \leftarrow \text{DGEnc}([[sk]]_{[n]}, [k : \mathbf{M}, \mathcal{S}])\right]$$

4.1.2 Security Definitions

All the security definitions are the same as in the previous ATSE scheme with a slight change in the games and oracles.

4.1.2.1 Message Privacy

The game and oracles of (strong) message privacy for this scheme have the same set of instructions in each protocol except replacing message vector by a matrix,, and the notation is inherited from the matrix to complete the set of instructions (e.g. I is a set of pairs of the form (i, j))(corresponds to) I in $O_{\text{at-mp-chal}}$, $O_{\text{at-str-mp-chal}}$ and $O_{\text{at-mp-pc-dec}}$.

4.1.2.2 Correctness

The game and oracles of (strong) correctness for this scheme have the same set of instructions in each protocol except replacing message vector by a matrix in every games and oracles.

4.1.2.3 Authenticity

Same as above like message privacy and correctness definition, we need to change the following.

- As message privacy and correctness.
- Replace N by $r \times s$ in $O_{at-au-enc}$ oracle.

4.1.3 Our Construction

TOOLS

- FTKD Scheme (containing the algorithms FTKD.Setup, DKdf, WKGen).
- Markle-tree group commitment scheme (contains the algorithms GSetup, GCommit, CardVer, GVer).
- A secure PRG with polynomially many output bits.

SCHEME

- $\mathsf{Setup}(1^\kappa, n, t) \rightarrow ([[sk]]_{[n]}, pp)$: Compute the distributed key $[[sk]]_{[n]} := (sk_1, sk_2, \ldots, sk_n)$, provide the public parameter pp_{FTKD} using $\mathsf{FTKD.Setup}(1^\lambda, n, t)$ and run GSetup to get pp_{comm}. $pp = (pp_{FTKD}, pp_{comm})$.
- $\mathsf{DGEnc}([[sk]]_{[n]}, pp, [k : \mathbf{M} = (m_{ij})_{r\times s}, \mathcal{S}]) \rightarrow ([k : \mathbf{C}/\perp])$: $\mathbf{M}$ is a tabular data (i.e. a matrix) of size $r \times s$ and the protocol initiated by the party k.
  - *Round 1:* Party k compute group commitments row-wise as well as column-wise as follows:

    $$(\sigma_i, \mathbf{q}_i, \mathbf{p}_i) \leftarrow \mathsf{GCommit}(\mathbf{M}_{i*}, pp_{comm}) \qquad i = 1 \rightarrow r$$

    $$\&(\delta_j, \mathbf{v}_j, \mathbf{u}_j) \leftarrow \mathsf{GCommit}(\mathbf{M}_{*j}, pp_{comm}) \qquad j = 1 \rightarrow s$$

    where $\mathbf{q}_i, \mathbf{v}_j$ are unique commitment vector and $\mathbf{p}_i, \mathbf{u}_j$ are opening vectors. Party j send the row commitments σ_i's $(k, (\sigma_1, \sigma_2, \ldots, \sigma_r), s)$ with no. of column in data matrix $\mathbf{M}$ to all the parties in $\mathcal{S}$.
  - *Round 2:* A server, on receiving the input, checks (i) whether it is sent by party k and (ii) $\mathsf{CardVer}(pp_{comm}, (\sigma, \mathrm{r}))= 1$; if either fails, then it returns $\perp$ and aborts; otherwise it continues with the protocol. Using DKdf protocol each party p compute $H_0(\sigma_i)^{sk_p}$'s on the received commitments and send back to the party k. If gorup key gk=$\perp$, then party k outputs $\perp$, otherwise it executes the next step.
  - *Finally*, party k locally compute the message specific whole keys by $w_{ij} := e(gk_i, H_1(\delta_j))$ for $i = 1 \rightarrow r$ & $j = 1 \rightarrow s$ and generate the ciphertext $c_{ij} = (\sigma_i, \delta_j, \mathbf{q}_i, \mathbf{v}_j, e_{ij})$ on the message m_{ij} and $e_{ij} := \mathsf{PRG}(w_{ij}) \oplus (m_{ij}||p_i||u_j)$.

  Output: $\mathbf{C} := (c_{i,j})$.
- $\mathsf{DGDec}([[sk]]_{[n]}, pp, [k : c, \mathcal{S}] \rightarrow ([k : m/\perp]))$: This protocol initiated by the party k.
  - Party k parse ciphertext c as $(\sigma, \delta, q, v, e)$.

– Parties in $\mathcal{S}$ interactively runs an instance of DKdf:

$$[k : wk] \leftarrow \mathsf{DKdf}([[sk]]_{[n]}, [k : ((\sigma, \delta), `whole), \mathcal{S}])$$

– If $wk \neq \perp$, then it output $\perp$, otherwise it executes the next step.

– Decrypt e as $(m||p||u) = \mathsf{PRG}(wk) \oplus e$

– Verify the commitment running $\mathsf{GVer}(pp_{com}, (\sigma, q), (m, p))$ and $\mathsf{GVer}(pp_{com}, (\delta, v), (m, u))$; if one of them return 0 then output $\perp$, else output m.

Theorem 4.1. *Our ATSE scheme described in this section, is (strongly)- secure if the underlying simple FTKD is (strongly-)secure.*

Correctness. The correctness of this ATSE scheme is achieved by binding of underlying group commitment scheme. The proof is analogs to proof of correctness of ATSE (see Appendix E.2 in [CGMS21]) app except here the correctness is due to two binding of group commitments of row and columns where the message lies. Hence we can check it using GVer by opening the unique commitment pair twice (once for column and another for row). Therefore we get more than what we need to achieve correctness. Hence this assure the correctness.

Message Privacy. There is a slight difference in this scheme compare to ATSE by Agwaral et. al. While computing the group specific left keys (i.e column wise group specific keys) our scheme does not include the id of encryption initiated party unlike the former. The proof of message privacy for the former (ATSE)(see Appendix E.2 in [CGMS21]) does not depend on the above but on pseudorandomness of message specific whole key using FTKD protocol while rely on BDDH assumption. Hence we can use same argument to support our claim and the only modification needed to create four types of each $r \times s$ hybrids games insead of 4 types of each N hybrid games to show indistinguishable of encryption of two different tabular data of same size. This assure that that this scheme satisfies (strong) message privacy.

Authenticity. The authenticity of the ATSE scheme for tabular data follows by adapting the proof for the vector case (see Appendix E.2 in [CGMS21]) to the matrix setting. Specifically, the message vector $\mathbf{m} = (m_1, \ldots, m_n)$ is replaced by a matrix $\mathbf{M} = (m_{ij})$ of size $r \times s$, and all corresponding oracles and games are adjusted accordingly.

The scheme employs dual group commitments: row-wise commitments σ_i for each row $i \in [r]$ and column-wise commitments δ_j for each column $j \in [s]$. These commitments bind the messages in both dimensions, providing a stronger binding guarantee than a single vector commitment. In the encryption protocol, the encryptor obtains group keys gk_i for each row via the FTKD scheme and derives message-specific whole keys $w_{ij} = e(gk_i, H_1(\delta_j))$ for each message m_{ij}. The ciphertexts include the corresponding row and column commitments, unique commitment vectors, and encrypted message with masking via a PRG keyed by w_{ij}.

The proof proceeds in the following steps:

- *Reduction to CardAuthGame:* We first reduce the standard authenticity game $\mathsf{AT\text{-}Auth}_{\mathcal{A}}$ to a modified game $\mathsf{CardAuthGame}_{\mathcal{A}}$, which additionally tracks the number of forgeries per

group commitment and aborts if the number of forgeries for a commitment exceeds its cardinality. This reduction is justified by the cardinality-binding property of the group commitment scheme.

- *Transition to Uniqueness Game:* We show that the modified game $\mathsf{CardAuthGame}_{\mathcal{A}}(1^{\kappa})$ is computationally indistinguishable from the further restricted game $\mathsf{AT\text{-}Auth\text{-}Uniq}_{\mathcal{A}}(1^{\kappa})$:

$$\mathsf{CardAuthGame}_{\mathcal{A}}(1^{\kappa}) \approx_{\mathrm{a}} \mathsf{AT\text{-}Auth\text{-}Uniq}_{\mathcal{A}}(1^{\kappa})$$

  where the adversary only wins if all forgeries are unique in their first three components (row, column, and commitment path).

- *Reduction to FTKD Pseudorandomness:* Now we bind the winning probability of $\mathcal{A}$ in the game $\mathsf{AT\text{-}Auth\text{-}Uniq}_{\mathcal{A}}$ by the winning probability of another PPT adversary $\mathcal{B}$ that attempts to win the pseudorandomness game $\mathsf{DP\text{-}PR}_{\mathcal{B}}$. The reduction $\mathcal{B}$ simulates the authenticity game for $\mathcal{A}$, using its own FTKD pseudorandomness challenge oracle to answer key derivation queries and embedding the challenge key in the challenge ciphertext. If $\mathcal{A}$ can forge a valid ciphertext, then $\mathcal{B}$ can distinguish the FTKD challenge key from random, contradicting the pseudorandomness of FTKD.

These steps apply to the modified games and oracles for the ATSE scheme instantiated for tabular data, as the matrix structure only changes the indexing and the dual use of group commitments. The security properties of the underlying FTKD and group commitment schemes, along with the PRG, remain unchanged.

Hence, the authenticity of the ATSE scheme for tabular data follows by the same reductions and arguments as in the vector case, concluding the proof.

Bibliography

[AMMR18] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed symmetric-key encryption. pages 1993–2010, 2018.

[BSA+25] Mariarosaria Barbaraci, Noah Schmid, Orestis Alpos, Michael Senn, and Christian Cachin. Thetacrypt: A distributed service for threshold cryptography. *arXiv preprint arXiv:2502.03247*, 2025.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. pages 280–300, 2013.

[CCMS21] Kelong Cong, Daniele Cozzo, Varun Maram, and Nigel P. Smart. Gladius: LWR based efficient hybrid public key encryption with distributed decryption. pages 125–155, 2021.

[CGMS21] Mihai Christodorescu, Sivanarayana Gaddam, Pratyay Mukherjee, and Rohit Sinha. Amortized threshold symmetric-key encryption. pages 2758–2779, 2021.

[DMSS24] Pousali Dey, Pratyay Mukherjee, Swagata Sasmal, and Rohit Sinha. HiSE: Hierarchical (threshold) symmetric-key encryption. Cryptology ePrint Archive, Paper 2024/158, 2024.

[dS22] Luan Cardoso dos Santos. *Design, Cryptanalysis and Protection of Symmetric Encryption Algorithms*. PhD thesis, University of Luxembourg, Esch-sur-Alzette, Luxembourg, 2022.

[DWZ+23] Sheng Deng, Zhiwei Wang, Xuebin Zhang, Yaping Lin, and Yiming Zhang. Bctc-ksm: A blockchain-assisted threshold cryptography for key security management in power iot data sharing. *Computers & Electrical Engineering*, 109:108996, 2023.

[Ebr24] Ehsan Ebrahimi. Security strengthening of threshold symmetric schemes. Cryptology ePrint Archive, Paper 2024/1377, 2024.

[Ham23] Maryum Hamid. Advanced secret handling in kubernetes application with hashicorp vault. Master's thesis, KTH Royal Institute of Technology, 2023.

[HAP18] Yotam Harchol, Ittai Abraham, and Benny Pinkas. Distributed ssh key management with proactive rsa threshold signatures. *IACR Cryptology ePrint Archive*, (2018/389), 2018.

[Has24] HashiCorp. Key management secrets engine. https://github.com/hashicorp/vault/blob/main/website/content/docs/secrets/key-management/index.mdx, 2024.

[KS22] Cecilia Boschini Komlo and Omer Shlomovits. Threshold cryptography in practice: From research to adoption. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 3102–3105, 2022.

[LLWL23] Meng Li, Yao Liu, Zhiwei Wang, and Yaping Lin. Incremental threshold scheme enabled iot group key management for symmetric key encryption. *OSTI Technical Reports*, 2023.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. pages 369–378, 1988.

[MGB+24] Florian Le Mouël, Maxime Godon, Renaud Brien, Erwan Beurier, Nora Boulahia-Cuppens, and Frédéric Cuppens. Trustless distributed symmetric-key encryption. arXiv preprint arXiv:2408.16137, 2024.

[Muk20] Pratyay Mukherjee. Adaptively secure threshold symmetric-key encryption. Cryptology ePrint Archive, Paper 2020/1329, 2020.

[Nat22] National Institute of Standards and Technology. Threshold cryptography. https://csrc.nist.gov/projects/threshold-cryptography, 2022. Accessed: 2025-07-09.

[NB23] Devika Kalathil Nandalal and Ramesh Bhakthavatchalu. Design of programmable hardware security modules for enhancing blockchain based security framework. *International Journal of Electrical and Computer Engineering (IJECE)*, 13(3):3178–3191, 2023.

[PBC] Introduction to pairing-based cryptography.

[SHS] Suitability of hardware security modules for securing remote access in high-performance environments.

[SY23] Kiarash Sedghighadikolaei and Attila Altay Yavuz. A comprehensive survey of threshold signatures: Nist standards, post-quantum cryptography, exotic techniques, and real-world applications. *arXiv preprint arXiv:2311.05514*, 2023.

[Urm17] Raoul-Gabriel Urma. Infrastructure secret management software overview. https://gist.github.com/binarymist/66206419df712bd738c3d664542157d8, 2017.

[VM16] V. Venukumar and S. S. Manvi. A survey of applications of threshold cryptography. *International Journal of Information Security and Privacy*, 10(4):53–66, 2016.

[VSB21] Giovanni Vigna, Elaine Shi, and Michael Backes, editors. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2021. Association for Computing Machinery.

[WH20] Xunhua Wang and Ben Huson. Robust distributed symmetric-key encryption. Cryptology ePrint Archive, Paper 2020/1001, 2020.

[ZLA+23] Jin Zhang, Zhiqiang Liu, Jiyong An, Shuang Wang, and Yanan Wang. Threshold fully homomorphic encryption scheme based on symmetric encryption. *Symmetry*, 17(5):737, 2023.