Dynamic Threshold Key Encapsulation with a Transparent Setup

Joon Sik Kim[1], Kwangsu Lee[*2], Jong Hwan Park[3], and Hyoseung Kim[*4]

[1]Korea University
[2]Sejong University
[3]Sangmyung University
[4]Hallym University

Abstract

A threshold key encapsulation mechanism (TKEM) facilitates the secure distribution of session keys among multiple participants, allowing key recovery through a threshold number of shares. TKEM has gained significant attention, especially for decentralized systems, including blockchains. However, existing constructions often rely on trusted setups, which pose security risks such as a single point of failure, and are limited by fixed participant numbers and thresholds. To overcome this, we propose a dynamic TKEM with a transparent setup, allowing for a flexible selection of recipients and thresholds without relying on trusted third parties in the setup phase. In addition, our construction does not rely on pairing operations. We prove the security of our TKEM under the decisional Diffie-Hellman assumption, ensuring selective chosen-ciphertext security and decapsulation consistency. Our proof-of-concept implementation highlights the practicality and efficiency of this approach, advancing the field of threshold cryptography.

1 Introduction

A threshold key encapsulation mechanism (TKEM) is a useful primitive in threshold cryptosystems, which distributes a session key among multiple parties. In the concept of TKEM with threshold parameters (n,t), the session key is split into n shares and encapsulated as a ciphertext header. Collecting at least t shares allows decapsulation of the header by reconstructing the original session key. TKEMs must fulfill two security requirements: chosen-ciphertext attack (CCA) security and decapsulation consistency (DC). Briefly, CCA security ensures that no adversary with fewer than t shares can gain any useful information about a session key corresponding to a given ciphertext header. A weaker notion, known as selective CCA security, applies only to a specific set of recipients chosen in advance. DC guarantees that any valid set of t shares out of n will consistently reconstruct the same session key.

Dynamic TKEMs allow the parameters (n,t) to be determined during encapsulation. This flexibility ensures that the number of participating parties and the required shares for decapsulation can change as needed. Furthermore, a transparent setup allows for the generation of public parameters without any secret trapdoor, ensuring that no single party can compromise the system. This is crucial for mitigating the risk of

[*]The two authors are the corresponding authors of this paper.

a single point of failure, thus establishing TKEMs as essential in distributed systems such as blockchain and decentralized finance (DeFi) [10, 20].

Despite the importance of TKEM, only a few results on dynamic threshold cryptosystems have been proposed (e.g., [13, 20, 28]), focusing on threshold public key encryption (TPKE). TPKE is analogous to TKEM in that both require collaboration among multiple parties to reconstruct a message (in TPKE) or a key (in TKEM). Although they can be converted into each other, TKEM is advantageous for transmitting large messages when integrated with a data encapsulation mechanism [30].

1.1 Our results

We propose a dynamic TKEM with a transparent setup, which is constructed without the need for pairing operations. Moreover, the proposed TKEM meets the security requirements, which are proven based on the decisional Diffie-Hellman assumption and underlying cryptographic primitives, in the random oracle model. We remark that this is the first result on threshold cryptosystems, as shown in Table 1. To do this, we focus on the following steps:

Dynamic Threshold: To achieve this, we employ the approach originating from [28], which utilizes a public key encryption (PKE) scheme, non-interactive zero-knowledge (NIZK) proof systems, and a polynomial-based secret sharing scheme. The secret sharing scheme is generally used to support the (n,t)-threshold setting. More precisely, a polynomial f of degree $t-1$ is used to share the secret $s_0 = f(0)$ by splitting it into n shares as $f(idx_1), \ldots, f(idx_n)$, where idx_i is assigned to a user i. Any t shares out of n can be used to reconstruct the secret by interpolation: $s_0 = \sum L_i(0) f(idx_i)$, where $L_i(0)$ is the Lagrange coefficient. To be dynamic, such a polynomial and its shares are now generated during encapsulation. Each share is encrypted using the PKE scheme under its corresponding public key and then included in the ciphertext header. The resulting ciphertext header additionally contains an NIZK proof, ensuring that the ciphertext header is well-formed; that is, the corresponding secret is correctly shared, and the shares are properly encrypted. Note that the construction in [28], following the above approach, employs a pairing-based PKE with a non-interactive opening (PKENO) scheme and a Groth-Sahai proof system [21], which relies on a trusted setup that we aim to avoid.

Transparent Setup: As we explained before, Groth-Sahai proof systems are problematic because they require a trusted setup to guarantee that public structured reference strings are generated without a trapdoor for real-world use. Since the trusted setup demands significant computational effort, and such strings can be subverted, as shown in [4], we address this problem by eliminating Groth-Sahai proof systems throughout our construction. One could consider applying NIZK proof systems transformed by the Fiat-Shamir heuristic [18], where the generation of strings is publicly verifiable, yielding a transparent setup. This is, however, not easily achieved, as Groth-Sahai proof systems are designed to prove complex mathematical structures; for example, proving both a group element $M \in \mathbb{G}$ and a scalar $f(idx_1) \in \mathbb{Z}_p$ such that $Mg^{f(idx_1)} \in \mathbb{G}$, where $\mathbb{G}$ is a finite group of order p. This cannot be handled by Fiat-Shamir transformed systems. Hence, we design our TKEM so that only scalars need to be proven while ensuring the well-formedness of our ciphertext header. To this end, we focus on the work of [15], originally used for constructing signature-based witness encryption. This literature demonstrates the way to prove that polynomial-based shares as exponents over a group $\mathbb{G}$, as demonstrated above, are correct to reconstruct the secret via Fiat-Shamir transformed NIZK proof systems. However, in [15], pairing operations are still needed for proving other aspects of well-formedness, such as the structure of the BLS scheme [6]. We carefully extend this proof system to be an entirely pairing-free construction, as described below.

Pairing-Free Construction: Pairing operations are relatively more expensive than exponentiations over a

Table 1: Comparison with the existing dynamic threshold cryptosystems

	Size			Dynamic	Transparent	Pairing	Method	Security	Assumption
	PP	*PK*&*SK*	*CT*	Threshold	Setup	Free		Model	
DP08 [13]	$O(N)$	$O(1)$	$O(1)$	△	X[†]	X	KEM	ROM	MSE-DDH
SES16a [28]	$O(1)$	$O(1)$	$O(n)$	O	O	X	PKE[§]	STM	DBDH/DLIN
SES16b [28]	$O(1)$	$O(1)$	$O(n)$	O	X	X	PKE	STM	DLIN
GKPW24 [20]	$O(N)$	$O(1)$	$O(1)$	△	X	X	PKE	ROM	GGM
Ours	$O(1)$	$O(1)$	$O(n)$	O	O	O	KEM	ROM	DDH

'O' indicates that the corresponding property is satisfied, while '△' indicates that it is satisfied with restrictions based on N (the maximum number of n participants). 'X' indicates that the corresponding property is not satisfied. ROM and STM refer to the random oracle model and the standard model, respectively. † requires key distribution with a higher level of trust. § indicates the conceptual difference where public verification of the well-formedness of CT is not allowed. Cryptographic assumptions in the last column are involved in the selective CCA security.

conventional group. To eliminate the dependency on pairings, we employ a technique for double ElGamal encryption with shared randomness, which is a well-known solution for constructing a CCA secure PKE scheme. This decision is mainly due to the fact that the structure of such ElGamal ciphertexts can be efficiently proven by statements for the equality of discrete logarithms. These statements can be ensured by using Fiat-Shamir transformed NIZK systems that we aim to utilize. Furthermore, the security of the employed PKE scheme contributes to the selective CCA security of TKEM, as shown in [28]. Based on this, our ciphertext header contains variant ElGamal cipheretexts to encrypt shares of participants. Finally, we elaborately incorporate statements to prove its well-formedness (i.e., that the encrypted shares are all correct). A detailed description will be demonstrated in Section 3.2.

In fact, our approach results in a TKEM producing a ciphertext header of size $O(n)$, where n is the number of participants, leading to computational inefficiency. This comes from the fact that neither ElGamal ciphertexts nor NIZK proofs can be aggregated. Nevertheless, the proposed TKEM could be a step towards future constructions with constant-size ciphertext headers. Moreover, to the best of our knowledge, no existing construction offers the features we have outlined above.

1.2 Related Works

Threshold cryptography was first introduced in [14], and since then, subsequent research has explored various fields, including KEM, PKE, and signature, as follows.

Dynamic Threshold KEM & PKE: The first dynamic threshold TKEM was constructed by Delerablée and Pointcheval [13], referred to here as DP08. While providing constant-size ciphertexts regardless of the number of participants, their construction relies on pairing operations. Moreover, DP08 has limitations in several respects: the dynamic threshold property is restricted as the maximum number of n participants must be fixed during the initial setup stage. Additionally, a trusted setup is required, involving a master key to distribute n secret keys. Regarding security, their selective CCA security is proven based on the multi-sequence of exponent Diffie-Hellman (MSE-DDH) assumption, which is a q-type assumption and may not hold against the attacks as shown in [9]. Constructing TPKE is another research topic in threshold cryptography. Notably, two fully dynamic threshold TPKE schemes, denoted as SES16a and SES16b, were presented in [28]. Both are proven secure based on the standard assumption, such as the decision linear (DLIN) assumption and the decisional bilinear Diffie-Hellman (DBDH) assumption, in the standard model. This makes their constructions theoretically stronger than those proven in the random oracle model (ROM). However, both depend on pairing operations. In particular, SES16b presents a concrete construction following the design paradigm described earlier (by utilizing Groth-Sahai systems), thus requiring a trusted setup. On the other hand, SES16a provides a generic construction whose instantiation allows a transparent setup. This originates from the required primitives, such as one-time signatures, commitment schemes, and PKENO, none of which need the help of a trusted setup. However, the syntax of SES16a differs slightly from others, even though TPKE is compatible with TKEM. The main difference is that the well-formedness of ciphertexts (in the context of TPKE) could not be publicly verified. Recently, Garg et al. [20] proposed a remarkable TPKE scheme, denoted as GKPW24, achieving constant-size ciphertexts. In contrast to DP08, this particularly removes the need for distributed key generation. This advantage is largely due to the succinctness of the Kate–Zaverucha–Goldberg polynomial commitment scheme [23] combined with the Fiat-Shamir heuristic in the ROM. However, such a commitment scheme relies on structured reference strings (SRS) that embed a trapdoor, thus requiring trust to ensure the trapdoor is not leaked. Indeed, in GKPW24, revealing the trapdoor can undermine even the semantic security of the scheme. Moreover, its security is proven in the generic group model, which is viewed as less robust than the standard model and ROM. Note that all the constructions mentioned earlier, including our TKEM, are secure against selective CCA.

Threshold Signatures: Extensive research has extended Schnorr signatures to construct threshold signature (TS) schemes, including [2,3,5,11,24,27,31], which are proven secure under the discrete logarithm problem in the ROM. However, these schemes require a trusted setup, either by executing a distributed key generation protocol or by designating a specific party to generate keys for all participants. Conversely, some pairing-based TS schemes [12, 19] enable a silent setup setting, where keys are generated locally and the setup is performed without interaction. Nevertheless, as in [20], a certain level of trust remains necessary to generate an SRS. Another line of research has investigated lattice-based TS schemes for post-quantum security (e.g., [1, 8, 16, 17, 22, 26]), though none of these constructions achieve a transparent setup.

2 Preliminaries

In this section, we provide the necessary prior knowledge to understand our paper.

2.1 Notations

For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1,\dots,n\}$. For $a,b \in \mathbb{Z}$ such that $a < b$, $[a,b]$ denotes the set $\{a,\dots,b\}$. We especially use $\mathbb{Z}_p$ to denote $[0,p]$ when p is a prime. A small bold letter means a vector, whereas a large bold letter means a matrix. Given a vector $\mathbf{v}$, $\mathbf{v}^\top$ means the transposed $\mathbf{v}$. The meaning of $a \xleftarrow{\$} \mathbb{S}$ is the uniformly random sampling of the element a from the set $\mathbb{S}$.

2.2 Complexity Assumptions

Assumption 1. (Decisional Diffie-Hellman assumption, DDH). Let $\mathbb{G}$ be a cyclic group of prime order p and g be a random generator of $\mathbb{G}$. We say that the DDH assumption holds, if for any probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$, the following advantage is negligible:

$$\begin{aligned}\mathbf{Adv}_{DDH}(\lambda) := |&\Pr[\mathcal{A}(\mathbb{G},p,g,g^\alpha,g^\beta,g^{\alpha\beta})] = 1 : g \xleftarrow{\$} \mathbb{G}, \alpha,\beta \xleftarrow{\$} \mathbb{Z}_p] \\ &- \Pr[\mathcal{A}(\mathbb{G},p,g,g^\alpha,g^\beta,D)] = 1 : g,D \xleftarrow{\$} \mathbb{G}, \alpha,\beta \xleftarrow{\$} \mathbb{Z}_p]|\end{aligned}$$

2.3 Lagrange Interpolation

Lagrange interpolation is a way to recover shared secret values efficiently. Let $f(\gamma) = \sum_{j=0}^{t-1} s_j \gamma^j$ be a $(t-1)$-degree polynomial and let $(\gamma_1, f(\gamma_1)), \ldots, (\gamma_n, f(\gamma_n))$ be n points evaluated by the polynomial with $n > t$. Given any t points out of the n points, we can recover the secret value $s_0 = f(0)$: Specifically, without loss of generality, these t points are represented as $(\gamma_1, f(\gamma_1)), \ldots, (\gamma_t, f(\gamma_t))$. Then we can compute $s_0 = \sum_{i \in [t]} f(\gamma_i) L_i(0)$ where $L_i(0) = \prod_{j \in [t], j \neq i} \frac{-\gamma_j}{\gamma_i - \gamma_j}$ is the Lagrange basis at the point 0.

2.4 Reed-Solomon Codes

Reed-Solomon codes are denoted as $RS_{n+1,t}[\gamma]$ for a vector $\gamma = (\gamma_0, \ldots, \gamma_n)^\top \in \mathbb{Z}_p^{n+1}$, where $t \in [n]$. For each $(t-1)$-degree polynomial $f(\gamma) = \prod_{j=0}^{t-1} s_j \gamma^j$, we let $\mathbf{s} = (s_0 \ldots, s_{t-1})^\top \in \mathbb{Z}_p^t$ and $\mathbf{f} = (f(\gamma_0), \ldots, f(\gamma_n))^\top$. Then $RS_{n+1,t}[\gamma]$ consists of two matrices $\mathbf{G} = (\gamma_i^j)_{i,j} \in \mathbb{Z}_p^{(n+1)\times t}$ and $\mathbf{H} = (\frac{1}{\prod_{\ell \in [0,n], \ell \neq j} \gamma_j - \gamma_\ell} \gamma_j^i)_{i,j} \in \mathbb{Z}_p^{(n-t+1)\times(n+1)}$. The former is a generator matrix since $\mathbf{G} \cdot \mathbf{s} = \mathbf{f}$, and the latter is a parity-check matrix since $\mathbf{H} \cdot \mathbf{G} = \mathbf{0}$. We adopt such Reed-Solomon codes to verify that $\mathbf{f}$ was correctly generated by $\mathbf{G}$, as in [15]. Note that $\mathbf{f}$ is correctly generated only if the equation $\mathbf{H} \cdot \mathbf{f} = \mathbf{H} \cdot \mathbf{G} \cdot \mathbf{s} = \mathbf{0}$ holds.

2.5 Non-Interactive Zero-Knowledge Proofs

Non-interactive zero-knowledge (NIZK) proofs are defined with an NP-language $\mathcal{L}_{\mathcal{R}} = \{s \in \{0,1\}^* | \exists w : (s,w) \in \mathcal{R}\}$, where $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ is efficiently recognizable relation. If $s \in \mathcal{L}_{\mathcal{R}}$, then we call such a s true statement for a witness w; otherwise, a false statement.

The NIZK proof system includes the following algorithms:

- **Prove**(CRS, s, w): It takes a common reference string CRS, a statement s, and a witness w as input, and outputs a proof π.
- **Verify**(CRS, s, π): It takes a common reference string CRS, a statement s, and a proof π as input, and outputs 1 when s is true and 0 otherwise.

Moreover, the NIZK-proof system should satisfy the following properties:

- *Completeness*: A verifier should successfully verify the proof generated by an honest prover who has a witness.
- *Soundness*: For a false statement s, a cheating prover cannot falsely convince the honest verifier that s is true.
- *Zero-knowledge*: There is a simulator S, not knowing a witness, which can generate a simulated proof that is indistinguishable from a real one.

In the case of non-interactive zero knowledge of proof of knowledge (NIZK-PoK), the extractability is stronger than the soundness, of which description is as follows:

- *Extractability*: Whenever a (potentially cheating) prover produces two valid proofs with respect to a statement, an efficient extractor can extract the witness from the information available to the adversary.

3 Dynamic TKEM with Transparent Setup

In this section, we introduce the definition and security models of a dynamic TKEM with transparent setup. Hereafter, for the sake of clarity, we will refer to it simply as a TKEM unless otherwise stated. We then propose our construction.

3.1 Definitions

We begin with the syntax of a TKEM. Our syntax is defined as a modification of dynamic TPKE [28], resulting in the TKEM concept. To ensure a transparent setup, no secret information is embedded in the **Setup** algorithm. Each user generates their own public and secret keys by running the **KeyGen** algorithm. The **Encap** algorithm produces a ciphertext header and a session key based on the specified threshold numbers t and n. Notably, an encapsulator (i.e., a user running the algorithm) can chooses t and n in advance each time the the algorithm is run, thereby reflecting the dynamic threshold setting. Due to the **CHVerify** algorithm, one can check if a given ciphertext header is well-formed or not. A legitimated user can decapsulate a given ciphertext header by running the **ShareDecaps** algorithm, which yields a (decapsulated) share. Each share can be verified through the **ShareVerify** algorithm. With a collection of t or more verified shares, the **Combine** algorithm recovers and outputs the session key. These algorithms are formally described as follows:

Definition 3.1 (Threshold Key Encapsulation Mechanism, TKEM)**.** A TKEM consists of the following six algorithms:

Setup$(1^\lambda) \to PP$: The setup algorithm takes as input a security parameter λ and outputs the public parameters PP.

KeyGen$(PP) \to (PK, SK)$: The key generation algorithm takes as input the public parameters PP, and it outputs a public key PK and a private key SK.

Encaps$(PP, PL, t) \to (CH, K)$: The encapsulation algorithm takes as input the public parameters PP, a list of public keys $PL = (PK_1, \ldots, PK_n)$, and a threshold value t. It outputs a ciphertext header CH and a session key K.

CHVerify$(PP, PL, CH) \to \{0,1\}$: The ciphertext header verify algorithm takes as input the public parameters PP, a list of public keys $PL = (PK_1, \ldots, PK_n)$, and a ciphertext header CH. It outputs 1 if a ciphertext header is valid and 0 otherwise.

ShareDecaps$(PP, PL, CH, SK_i) \to \mu_i$: The share decapsulation algorithm takes as input the public parameters PP, a list of public keys $PL = (PK_1, \ldots, PK_n)$, a ciphertext header CH, and a private key SK_i. It outputs a decapsulated share μ_i.

ShareVerify$(PP, CH, \mu_i, PK_i) \to \{0,1\}$: The share verification algorithm takes as input the public parameters PP, a ciphertext header CH, a decapsulated share μ_i, and a public key PK_i. It outputs 1 if the share is valid and 0 otherwise.

Combine$(PP, PL, CH, \{\mu_i\}_{i\in S}) \to K$: The combining algorithm takes as input the public parameters PP, a list of public keys $PL = (PK_1, \ldots, PK_n)$, a ciphertext header CH, and a set of decapsulated shares $\{\mu_i\}_{i\in S}$. It outputs a session key K or $\perp$.

A TKEM is correct if the following conditions are satisfied: Let n and t be integers such that $1 \leq t \leq n$. For any PP generated by $\mathbf{Setup}(1^{\lambda})$, any $(PK_1, SK_1), \ldots, (PK_n, SK_n)$ generated by $\mathbf{KeyGen}(PP)$, and any (CH, K) generated by $\mathbf{Encaps}(PP, PL, t)$ for $PL = (PK_1, \ldots, PK_n)$, it is required that

- $\mathbf{CHVerify}(PP, PL, CH) = 1$.
- For any $PK_i \in PL$, we have that $\mu_i = \mathbf{ShareDecaps}(PP, PL, CH, SK_i)$ and $\mathbf{ShareVerify}(PP, CH, \mu_i, PK_i) = 1$.
- For any $S = \{i_1, \ldots, i_t\} \subseteq [n]$ such that $|S| = t$ and $PK_{i_j} \in PL$, we have that $\{\mu_i = \mathbf{ShareDecaps}(PP, PL, CH, SK_i)\}_{i \in S}$ and $\mathbf{Combine}(PP, PL, CH, \{\mu_i\}_{i \in S}) = K$.

Now, we consider the IND-CCA security of a TKEM, implying that the adversary cannot obtain any information about a session key corresponding to a given ciphertext header. In the security game, a challenger $\mathcal{C}$ provides an honest public key list to an adversary $\mathcal{A}$. Then $\mathcal{A}$ submits a corrupted public key list to $\mathcal{C}$. Moreover, $\mathcal{A}$ has access to a share decapsulation oracle, which enables to obtain a share of the ciphertext header. The goal of $\mathcal{A}$ is to distinguish between the real session key and a randomly chosen key. Further details are as follows.

Definition 3.2 (IND-CCA Security)**.** The IND-CCA security game, denoted as $\mathbf{G}_{\mathcal{A}}^{\textit{IND-CCA}}$, between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ is demonstrated as follows:

Setup: $\mathcal{C}$ obtains PP by running $\mathbf{Setup}(1^{\lambda})$ and gives PP to $\mathcal{A}$. It generates $(PK_1, SK_1), \ldots, (PK_{n_1}, SK_{n_1})$ for honest users by running $\mathbf{KeyGen}(PP)$. It then gives $\mathcal{U}_H = \{PK_1, \ldots, PK_{n_1}\}$ to $\mathcal{A}$. Next, $\mathcal{A}$ submits $\mathcal{U}_C = \{PK_{n_1+1}, \ldots, PK_{n_1+n_2}\}$ of corrupted users.

Query 1: To obtain shares for honest users, $\mathcal{A}$ can make a share decapsulation query that includes a ciphertext header CH, a list of public keys $PL = \{PK'_1, \ldots, PK'_n\} \subseteq \mathcal{U}_H \cup \mathcal{U}_C$, and a public key $PK_i \in \mathcal{U}_H$. If $\mathbf{CHVerify}(PP, PL, CH) \neq 1$, then $\mathcal{C}$ outputs $\perp$. Otherwise, $\mathcal{C}$ returns a share μ_i that is obtained by running $\mathbf{ShareDecaps}(PP, PL, CH, SK_i)$ to $\mathcal{A}$.

Challenge: $\mathcal{A}$ provides a list of challenge public keys $PL^* = \{PK_1^*, \ldots, PK_{n^*}^*\} \subseteq \mathcal{U}_H \cup \mathcal{U}_C$ and a challenge threshold $t^* \in [n^*]$ such that $|PL^* \cap \mathcal{U}_C| \leq t^* - 1$. $\mathcal{C}$ selects a random K_0^* and obtains (CH^*, K_1^*) by running $\mathbf{Encaps}(PP, PL^*, t^*)$. Next, it chooses a random bit $b \in \{0,1\}$ and gives the challenge ciphertext tuple (CH^*, K_b^*) to $\mathcal{A}$.

Query 2: $\mathcal{A}$ may additionally request share decapsulation queries and $\mathcal{C}$ handles these queries as in **Query 1**, except that $\mathcal{A}$ is not allowed to request a query for the challenge ciphertext header CH^*.

Guess: Finally, $\mathcal{A}$ submits a bit $b' \in \{0,1\}$. $\mathcal{C}$ outputs 1 if $b = b'$. Otherwise, it outputs 0.

The advantage of $\mathcal{A}$ in the security parameter λ is defined as $\mathbf{Adv}_{\textit{TKEM}}^{\textit{IND-CCA}}(\lambda) = \left|\Pr[\mathbf{G}_{\mathcal{A}}^{\textit{IND-CCA}} = 1] - \frac{1}{2}\right|$ where the probability is taken over all the randomness of the game. A TKEM is IND-CCA secure if for any PPT adversary $\mathcal{A}$, $\mathbf{Adv}_{\textit{TKEM}}^{\textit{IND-CCA}}(\lambda)$ is negligible.

Remark 1 (Selective IND-CCA). We proved our TKEM in the selective IND-CCA security model, where the selective security is considered a weaker notion. The main difference between the selective security game and the full security game (Definition 3.2) is that PL^* is now defined as $PL^* = PL_C^* \cup PL_H^*$. Here, PL_C^* consists of corrupted public keys chosen by $\mathcal{A}$ in the setup phase to be used in the challenge phase. Specifically, $\mathcal{A}$ having PP submits a challenge threshold t^* and $PL_C^* = \{PL_1^*, \ldots, PL_{t^*-1}^*\}$ to $\mathcal{C}$. After receiving $\mathcal{U}_H$, $\mathcal{A}$

then submits PL_R to $\mathcal{C}$, such that $PL_R \cup PL_C^*$ becomes $\mathcal{U}_C$ (i.e., a list of corrupted public keys). During the challenge phase, $\mathcal{A}$ submits PL_H^*, which defines PL^* as described. We denote the advantage of $\mathcal{A}$ in the selective IND-CCA game as $\mathbf{Adv}_{TKEM}^{SE\text{-}IND\text{-}CCA}$.

The decapsulation consistency (DC) is also a crucial security requirement of a TKEM. The DC security ensures that, for any valid shares generated from the same ciphertext header, their combination yields the same session key. Similar to the IND-CCA security, an adversary $\mathcal{A}$ against DC may compromise some public keys. Therefore, in the DC security game, $\mathcal{A}$ finally submits a ciphertext header and two distinct sets of shares derived from the header. The goal of $\mathcal{A}$ is that all shares in both sets are valid, and their combination results in different keys.

Definition 3.3 (DC Security)**.** The DC security game, denoted as $\mathbf{G}_{\mathcal{A}}^{DC}$, between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ is demonstrated as follows:

Setup: $\mathcal{C}$ obtains PP by running $\mathbf{Setup}(1^\lambda)$ and gives PP to $\mathcal{A}$. It generates key pairs $(PK_1, SK_1), \ldots, (PK_{n_1}, SK_{n_1})$ for honest users by running $\mathbf{KeyGen}(PP)$. It then gives $\mathcal{U}_H = \{PK_1, \ldots, PK_{n_1}\}$ to $\mathcal{A}$. Next, $\mathcal{A}$ submits $\mathcal{U}_C = \{PK_{n_1+1}, \ldots, PK_{n_1+n_2}\}$ of corrupted users.

Query: To obtain shares for honest users, $\mathcal{A}$ can make a share decapsulation query that includes a ciphertext header CH, a list of public keys $PL = \{PK_1, \ldots, PK_n\} \subseteq \mathcal{U}_H \cup \mathcal{U}_C$, and a public key $PK_i \in \mathcal{U}_H$. Then $\mathcal{C}$ returns a share μ_i that is obtained by running $\mathbf{ShareDecaps}(PP, PL, CH, SK_i)$ to $\mathcal{A}$.

Forge: Finally, $\mathcal{A}$ outputs a ciphertext header CH^* for a list of public keys $PL^* = \{PK_1^*, \ldots, PK_{n^*}^*\} \subseteq \mathcal{U}_H \cup \mathcal{U}_C$ and a threshold $t^* \in [n^*]$, and two sets of decapsulated shares $\{\mu_i\}_{i\in S}$ and $\{\mu_j'\}_{j\in S'}$ such that $t^* \leq |S| = |S'| \leq n^*$ and $S, S' \subseteq [n^*]$. $\mathcal{C}$ outputs 1 if the following three conditions are satisfied: 1) $\mathbf{Combine}(PP, PL^*, CH^*, \{\mu_i\}_{i\in S}) \neq \mathbf{Combine}(PP, PL^*, CH^*, \{\mu_j'\}_{j\in S'})$, 2) for all $i \in S$ and all $j \in S'$, $\mathbf{ShareVerify}(PP, CH^*, \mu_i, PK_i^*) = 1 \wedge \mathbf{ShareVerify}(PP, CH^*, \mu_j', PK_j^*) = 1$, and 3) $\mathbf{CHVerify}(PP, PL^*, CH^*) = 1$. Otherwise, it outputs 0.

The advantage of $\mathcal{A}$ in the security parameter λ is defined as $\mathbf{Adv}_{TKEM}^{DC}(\lambda) = \Pr[\mathbf{G}_{\mathcal{A}}^{DC} = 1]$ where the probability is taken over all the randomness of the game. A TKEM is DC secure if for any PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible.

3.2 Proposed TKEM

3.2.1 High-level description

To construct our TKEM, we begin by combining the ElGamal PKE scheme and Fiat-Shamir transformed NIZK-proof systems. Let the public key and the secret key of a user (indicated by idx_i) be $PK_i = X_i = g^{x_i}$ and $sk_i = x_i$, respectively. Due to the dynamic setting, an encapsulator chooses threshold values t and n to define a $(t-1)$-degree polynomial $f(\gamma) = \sum_{j=0}^{t-1} s_j \gamma^j$. Consider that each $f(idx_i)$ is a share for reconstructing the secret $s_0 = f(0)$. We have the following structure:

$$C = (K \cdot g^{s_0}, g^r, \{A_i = g^{f(idx_i)} \cdot X_i^r\}_{i\in[n]}),$$

where K is a session key hidden with g^{s_0} and the tuple (g^r, A_i) is an ElGamal ciphertext of $g^{f(idx_i)}$ under X_i. Using the Lagrange interpolation, one can derive g^{s_0} from at least t shares included in $g^{f(idx_i)}$. These shares in exponent can be obtained by eliminating $X_i^r = (g^r)^{x_i}$ through the ElGamal decryption process. However, since the employed ElGamal scheme is not secure against IND-CCA, the structure of C inherits this weak security. We therefore apply the Naor-Yung transformation [25] to the encryption scheme, which is a classical solution for achieving CCA-security. This transformation results in the double key setting where $PK_i = (X_i = g^{x_i}, Y_i = g^{y_i})$ and $sk_i = (x_i, y_i)$. Consequently, the elements $\{B_i = g^{f(idx_i)} \cdot Y_i^r\}_{i\in[n]}$ are additionally included in C. Now, it is required to prove that $(K \cdot g^{f(0)}, g^r, \{A_i, B_i\})$ is well-formed. However, this is not easily done due to the structure involving merged the double ElGamal ciphertexts with the hidden session key. To address this, we adopt the approach of Mcfly [15] by letting $K = h^r$. This allows us to use NIZK-WF (see Chapter 3.3.2), an efficient Fiat-Shamir transformed NIZK system that proves the same r is used in the exponent, thereby ensuring the well-formedness that we require. Note that the public parameter for this NIZK can be publicly generated, and hence, no trapdoor is embedded therein.

For additional security reasons, we require two NIZK systems: NIZK-PoK (see Chapter 3.3.1) that proves knowledge of discrete logarithms and NIZK-EDL (see Chapter 3.3.3) that proves the equality of discrete logarithms. Regarding IND-CCA, our security proof must use secret keys of corrupted users to generate a challenge ciphertext header, but these keys are committed and submitted by an adversary. Therefore, extractions of the secrets are necessary from proofs of knowledge. To be DC secure, each share (encrypted under PK_i) should be obtained by following the decryption with sk_i, and if so, $X_i^r = A_i / g^{f(idx_i)}$ can be computed correctly. This is ensured by checking the equality of the discrete logarithms of X_i and X_i^r on g and g^r, respectively.

3.2.2 Construction

Our TKEM is described as follows:

TKEM.Setup(1^λ): It first generates a cyclic group $\mathbb{G}$ of prime order p with a random generator $g \in \mathbb{G}$. It selects a random element $h \in \mathbb{G}$. Next, it chooses four hash functions $H_0, H_1, H_2, H_4 : \{0,1\}^* \to \mathbb{Z}_p$ and $H_3 : \{0,1\}^* \to \mathbb{Z}_p^\ell$. Finally, it outputs public parameters $PP = (p, \mathbb{G}, g, h, H_0, H_1, H_2, H_3, H_4)$. We implicitly set $CRS_{PoK} = (p, \mathbb{G}, g, H_1)$ and $CRS_{WF} = (p, \mathbb{G}, g, h, H_2, H_3)$.

TKEM.KeyGen(PP): It first selects random exponents $x, y \in \mathbb{Z}_p$ and computes $X = g^x$ and $Y = g^y$. It generates a proof $\pi =$ **NIZK-PoK.Prove**$(CRS_{PoK}, (X = g^x \wedge Y = g^y), (x, y))$ for the proof of knowledge (x, y). Finally, it outputs a public key $PK = (X, Y, \pi)$ and a private key $SK = (x, y)$.

TKEM.Encaps(PP, PL, t): Let $PL = \{PK_1, \dots, PK_n\}$ where $PK_i = (X_i, Y_i, \pi_i)$. It proceeds as follows:

1. For each $\pi_i \in PK_i$, it checks that **NIZK-PoK.Verify**$(CRS_{PoK}, (X = g^x \wedge Y = g^y), \pi_i) = 1$. If one of these checks fails, it outputs $\perp$.
2. It chooses random exponents $s_0, \dots, s_{t-1} \in \mathbb{Z}_p$ and defines a $(t-1)$-degree polynomial $f(\gamma) = \sum_{j=0}^{t-1} s_j \cdot \gamma^j$ such that $f(0) = s_0$.
3. For $PK_i \in PL$, it calculates $H_0(PK_i) = idx_i$ and $f(idx_i)$.
4. It selects random exponent $r \in \mathbb{Z}_p$ and builds $CL = (C_1, C_2, (A_1, B_1), \dots, (A_n, B_n), t)$ such that $C_1 = h^r g^{s_0}, C_2 = g^r, A_i = g^{f(idx_i)} X_i^r$, and $B_i = g^{f(idx_i)} Y_i^r$ for all $i \in [n]$.
5. Next, it generates a proof $\psi =$**NIZK-WF.Prove**$(CRS_{WF}, (CL, PK_1, \dots, PK_n)^1, r)$ for the well-formedness of CL.
6. Finally, it outputs a ciphertext header $CH = (CL, \psi)$ and a session key $K = h^r$.

[1]This is an informal statement meaning that CL is well-formed with respect to $PK_1, \dots, PK_n$. A formal description is provided in Section 3.3.2

TKEM.CHVerify(PP, PL, CH): : Let $PL = \{PK_1, \dots, PK_n\}$ and $CH = (CL, \psi)$, where $CL = (C_1, C_2, (A_1, B_1), \dots, (A_n, B_n), t)$. It outpus 1 if **NIZK-WF.Verify**$(CRS_{WF}, (CL, PK_1, \dots, PK_n), \psi) = 1$ and 0 otherwise.

TKEM.ShareDecaps(PP, PL, CH, SK_i) : Let $PL = \{PK_1, \dots, PK_n\}$ and $CH = (CL, \psi)$, where $CL = (C_1, C_2, (A_1, B_1), \dots, (A_n, B_n), t)$, and $SK_i = (x_i, y_i)$. To decapsulate a share, it proceeds as follows:

1. It implicitly sets $CRS_{EDL} = (p, \mathbb{G}, g, C_2, H_4)$.
2. It computes $D_i = A_i \cdot (C_2)^{-x_i}$ and generates a proof $\sigma_i =$ **NIZK-EDL.Prove**$(CRS_{EDL}, (X_i = g^{x_i} \wedge A_i/D_i = (g^r)^{x_i} \wedge Y_i = g^{y_i} \wedge B_i/D_i = (g^r)^{y_i}), (x_i, y_i))$ for the equality of discrete logarithms.
3. Finally, it outputs a decapsulated share $\mu_i = (D_i, \sigma_i)$.

TKEM.ShareVerify(PP, CH, μ_i, PK_i) : Let $CH = (CL, \psi)$, where $CL = (C_1, C_2, (A_1, B_1), \dots, (A_n, B_n), t)$, $\mu_i = (D_i, \sigma_i)$, and $PK_i = (X_i, Y_i, \pi_i)$. It implicitly sets $CRS_{EDL} = (p, \mathbb{G}, g, C_2, H_4)$. If **NIZK-EDL.Verify** $(CRS_{EDL}, (X_i = g^{x_i} \wedge A_i/D_i = (g^r)^{x_i} \wedge Y_i = g^{y_i} \wedge B_i/D_i = (g^r)^{y_i}), \sigma_i) = 1$, then outputs 1. Otherwise it outputs 0.

TKEM.Combine($PP, PL, CH, \{\mu_i\}_{i \in S}$) : Let $PL = \{PK_1, \dots, PK_n\}$ and $CH = (CL, \psi)$, where $CL = (C_1, C_2, (A_1, B_1), \dots, (A_n, B_n), t)$, $\mu_i = (D_i, \sigma_i)$, and $|S| = t$. To derive a session key, it proceeds as follows:

1. It outputs a session key $K = C_1 \cdot \prod_{j \in S} D_j^{-L_j(0)}$ where $L_j(0) = \prod_{k \in S, k \neq j} (-idx_k)/(idx_j - idx_k)$ is a Lagrange coefficient.

The correctness of our TKEM is given by the completeness of used NIZK proof systems and the following equation:

$$\begin{aligned} C_1 \cdot \prod_{j \in [t]} D_j^{-L_j(0)} &= h^r g^{f(0)} \cdot \prod_{j \in S} (g^{f(idx_j)})^{-L_j(0)} = h^r g^{f(0)} \cdot g^{\sum_{j \in S} -f(idx_j) L_j(0)} \\ &= h^r g^{f(0)} \cdot g^{-f(0)} = h^r = K. \end{aligned}$$

Remark 2. Note that this setup algorithm can be **transparent** by ensuring that such generations do not require any secret information. Specifically, the generation for $(g, \mathbb{G})$ is deterministic and public, and the element h can be generated by a map-to-point hash function. This can be publicly verified given the specification of the function.

Remark 3. Since our TKEM supports a full dynamic property, the threshold parameters (t, n) are determined by the encapsulator. In practice, the output lengths of H_3 and a parity check matrix $\mathbf{H}$ are decided based on (t, n). Furthermore, both the verifier and the decapsulator can use (t, n) to generate the same H_3 and $\mathbf{H}$ as used by the encapsulator.

3.3 Underlying NIZK

In this section, we describe three underlying NIZK proof systems that are constructed using the Fiat-Shamir transformation in the random oracle model: NIZK-PoK, NIZK-WF, and NIZK-EDL. Notably, our NIZK-WF system is a modification of [15] eliminating the need for pairings, whereas the other systems rely on the well-known NIZK system for a generalized representation [7]. Thus, we carefully demonstrate that the NIZK-WF satisfies all properties described in Section 3.3. In the following description, the notations used retain the same meanings as previously defined.

3.3.1 NIZK for the Proof of Knowledge (NIZK-PoK)

The underlying NIZK-PoK system for the relation $\mathcal{R}_{PoK} = \{(s_{PoK}, w_{PoK})\}$, where $s_{PoK} = (X = g^x \wedge Y = g^y)$ and $w_{PoK} = (x, y)$, is described as follows:

NIZK-PoK.Prove$(CRS_{PoK}, s_{PoK}, w_{PoK})$: Let $CRS_{PoK} = (p, \mathbb{G}, g, H_1)$. It first selects random exponents $r_1, r_2 \in \mathbb{Z}_p$ to compute $R_1 = g^{r_1}$ and $R_2 = g^{r_2}$. Next, it obtains $e = H_1(X, Y, R_1, R_2)$ and calculates $z_1 = r_1 + ex$ and $z_2 = r_2 + ey$. It outputs a proof $\pi = (e, z_1, z_2)$.

NIZK-PoK.Verify$(CRS_{PoK}, s_{PoK}, \pi)$: For $\pi = (e, z_1, z_2)$, it computes $R_1 = g^{z_1} \cdot X^{-e}$ and $R_2' = g^{z_2} \cdot Y^{-e}$ and checks $e \overset{?}{=} H_1(X, Y, R_1, R_2)$. If this check succeeds, it outputs 1. Otherwise, it outputs 0.

According to [7], the NIZK-PoK system guarantees the properties of completeness, zero-knowledge, and extractability. Here, as mentioned in Section 3.3, the extractability is considered rather than the soundness.

3.3.2 NIZK for the Proof of Well-Formedness (NIZK-WF)

Let a list of ciphertext elements be denoted as $CL = (C_1, C_2, (A_1, B_1), \ldots, (A_n, B_n), t)$ such that $C_1 = h^r g^{f(0)}$, $C_2 = g^r, A_i = g^{f(idx_i)} X_i^r$, and $B_i = g^{f(idx_i)} Y_i^r$. We say that CL is well-formed for our TKEM if and only if the following conditions are satisfied:

1. The same exponent r is used for all elements in CL.
2. For all $i \in [n]$, both A_i and B_i are ElGamal ciphertexts of the same $f(idx_i)$.
3. Each $f(idx_i)$ is a valid secret share of the secret $f(0)$.

Fortunately, the conditions 1 and 2 might be handled with the existing techniques for NIZK-EDL and for the double ElGamal encryption with the shared randomness, respectively. We thus focus on the final condition: By the Reed-Solomon (RS) codes, anyone given $(0, idx_1, \ldots, idx_n)$ can obtain a parity check matrix $\mathbf{H} \in \mathbb{Z}_p^{(n-t+1)\times(n+1)}$ such that $\mathbf{H} \cdot \mathbf{f} = \mathbf{0}$, where $\mathbf{f} = (f(0), f(idx_1), \ldots, f(idx_n))^\top$. Assume that a vector $\mathbf{v} \in \mathbb{Z}_p^{n-t+1}$ has the uniformly random distribution via the random oracle. The fact that $\mathbf{v}^\top \cdot \mathbf{H} \cdot \mathbf{f} = \mathbf{v}^\top \cdot \mathbf{0} = 0$ implies that secret shares are all valid with overwhelming probability. Based on these, there is a way to check whether $\mathbf{f}$ is composed of valid secret shares: From CL and $\mathbf{w} = (w_0, \ldots, w_n)^\top$, we can derive the following components:

$$C_A = C_1^{w_0} \prod_{i=1}^{n} A_i^{w_i},\ C_B = C_1^{w_0} \prod_{i=1}^{n} B_i^{w_i},\ H_A = h^{w_0} \prod_{i=1}^{n} X_i^{w_i},\ H_B = h^{w_0} \prod_{i=1}^{n} Y_i^{w_i}. \tag{1}$$

Assume that h^r in C_1, X_i^r in A_i, and Y_i^r in Y_i are well-formed, which partially satisfies the condition 1 above. We then have the following relations:

$$C_A = (h^r g^{f(0)})^{w_0} \prod_{i=1}^{n} (g^{f(idx_i)} X_i^r)^{w_i} = (h^{w_0} \prod_{i=1}^{n} X_i^{w_i})^r g^{f(0)w_0 + \sum_{i=1}^{n} f(idx_i) w_i} = H_A^r \cdot g^{\mathbf{w}^\top \mathbf{f}},$$

$$C_B = (h^r g^{f(0)})^{w_0} \prod_{i=1}^{n} (g^{f(idx_i)} Y_i^r)^{w_i} = (h^{w_0} \prod_{i=1}^{n} Y_i^{w_i})^r g^{f(0)w_0 + \sum_{i=1}^{n} f(idx_i) w_i} = H_B^r \cdot g^{\mathbf{w}^\top \mathbf{f}}.$$

If we set $\mathbf{w}^\top = \mathbf{v}^\top \cdot \mathbf{H}$ then $\mathbf{w}^\top \cdot \mathbf{f} = \mathbf{v}^\top \cdot \mathbf{H} \cdot \mathbf{f} = 0$, which results in $C_A = H_A^r \wedge C_B = H_B^r$. Let $P_i = A_i / B_i$ and $Q_i = X_i / Y_i$ for all $i \in [n]$. The remaining condition (including the condition 2) is now ensured by $C_2 = g^r \wedge P_i = Q_i^r$ for the same r. Consequently, we only need to verify that the exponent r is consistently used for g, H_A, H_B, and P_i, thereby completing the verification of all conditions simultaneously. To formally prove this argument, we provide the following lemma.

Lemma 3.1. *Let $CL = (C_1, C_2, (A_1, B_1), \ldots (A_n, B_n), t)$ be a list of ciphertext elements. If there exists an exponent r such that $(C_2 = g^r \wedge C_A = H_A^r \wedge C_B = H_B^r \wedge P_1 = Q_1^r \wedge \ldots \wedge P_n = Q_n^r)$, then CL is well-formed with probability of at least $1 - 3/p$.*

Proof. We demonstrate that if CL is not well-formed, then such an exponent r exists with negligible probability. Consider that CL is defined as $\big(h^{r_0} g^{f_0(0)}, g^r, (g^{f_1(idx_1)} X_1^{r_1}, g^{f_1'(idx_1)} Y_1^{r_1'}), \ldots, (g^{f_n(idx_n)} X_n^{r_n}, g^{f_n'(idx_n)} Y_n^{r_n'}), t\big)$. If, for all $i \in [n]$, the event E_1 where $r = r_i = r_i'$ and the event E_2 where $f_0 = f_i = f_i'$ occur together, CL is considered well-formed. Hence, when CL is not well-formed, there are three distinct cases: *(Case 1)* $\neg E_1 \wedge \neg E_2$, *(Case 2)* $\neg E_1 \wedge E_2$, and *(Case 3)* $E_1 \wedge \neg E_2$.

In Case 1, from the equations in (1), the equation $\log_g C_A - \log_g H_A^r = 0$ gives the following:

$$\begin{aligned}\log_g C_A - \log_g H_A^r &= \log_g h \cdot w_0 \cdot r_0 + w_0 \cdot f_0(0) + \sum_{i=1}^{n} w_i (x_i r_i + f_i(idx_i)) - (\log_g h \cdot w_0 + \sum_{i=1}^{n} w_i x_i) \cdot r \\ &= (\log_g h \cdot r_0 + f_0(0) - \log_g h \cdot r) \cdot w_0 + \sum_{i=1}^{n} (x_i r_i + f_i(idx_i) - x_i r) \cdot w_i = 0. \qquad (2)\end{aligned}$$

Similarly, we have the following equation:

$$\log_g C_b - \log_g H_b^r = (\log_g h \cdot r_0 + f_0(0) - \log_g h \cdot r) \cdot w_0 + \sum_{i=1}^{n} (y_i r_i' + f_i'(idx_i) - y_i r) \cdot w_i = 0. \qquad (3)$$

To achieve $C_A = H_A^r$ and $C_b = H_b^r$ without controlling the random oracle H_3, all of terms in (2) and (3) must be zero. This is because $\mathbf{w} = (w_0, \ldots, w_n)^\top$ was determined by $\mathbf{v} = H_3(CL)$. However, in this way, CL has the structure $(h^r, g^r, \{X_i^r, Y_i^r\}, t)$, which is still well-formed with respect to the zero polynomial (i.e., all coefficients are zero). This contradicts the assumption that CL is not well-formed. Hence, there must be at least one non-zero term in (2). This implies that $C_A = H_A^r$ holds with a probability of at most $1/p$ (due to the Schwartz-Zippel Lemma [29, 32]).

In Case 2, we can have $f_0(0) w_0 + \sum_{i=1}^{n} f_i(idx_i) w_i = 0$, i.e., $\mathbf{w}^\top \cdot \mathbf{f} = 0$, where $\mathbf{f} = (f_0(0), f_1(idx_1), \ldots, f_n(idx_n))$. Hence, the equation (2) can be simplified as $(\log_g h \cdot r_0 - \log_g h \cdot r) \cdot w_0 + \sum_{i=1}^{n} (x_i r_i - x_i r) \cdot w_i = 0$, where at least one non-zero term exists, because E_1 does not happen. Therefore, $C_A = H_A^r$ holds with probability of at most $1/p$.

In Case 3, assuming that $P_i = A_i / B_i = (X_i / Y_i)^r = Q_i^r$, we have $f_i(idx_i) = f_i'(idx_i)$ as

$$\log_g P_i - \log_g Q_i^r = (f_i(idx_i) + x_i r_i - f_i'(idx_i) - y_i r_i') - r(x_i - y_i) = f_i(idx_i) - f_i'(idx_i) = 0.$$

From this, there must some i such that $f_0 \neq f_i$ of $\mathbf{f}$ (defined as before). Let $\mathbf{H}$ be a parity check matrix of the RS codes, with respect to $\mathbf{f}' = (f(0), f(idx_1), \ldots, f(idx_n))^\top$, where $f = f_0$. Then $C_A = H_A^r$ results in $\mathbf{w}^\top \cdot \mathbf{f} = \mathbf{v}^\top \cdot \mathbf{H} \cdot \mathbf{f} = 0$ with $\mathbf{f} \neq \mathbf{f}'$. However, this holds when the randomly sampled $\mathbf{v}$ unfortunately leads to this equality. The probability that such a bad $\mathbf{v}$ is sampled is at most $1/p$.

□

Based on these, for the underlying NIZK-WF system, we can define the relation $\mathcal{R}_{WF} = \{(s_{WF}, w_{WF})\}$, where $s_{WF} = (C_2 = g^r \wedge C_A = H_A^r \wedge C_B = H_B^r \wedge P_1 = Q_1^r \wedge \ldots \wedge P_n = Q_n^r)$ and $w_{WF} = r$. However, for the sake of simplicity in our TKEM explanation, we informally define the alternative statement $s_{WF}' = (CL, PK_1, \ldots, PK_n)$ to mean that CL is well-formed with respect to $PK_1, \ldots, PK_n$. In this way, the following proving and verification algorithms of the NIZK-NF system start by computing C_A, C_B, and $\{P_i, Q_i\}$.

NIZK-WF.Prove$(CRS_{WF}, s'_{WF}, w_{WF})$: Let $CRS_{WF} = (p, \mathbb{G}, g, h, H_2, H_3)$, $CL = (C_1, C_2, (A_1, B_1), \ldots, (A_n, B_n), t) \in s'_{WF}$, and $PK_i = (X_i, Y_i, \pi_i) \in s'_{WF}$. To generate a proof, it proceeds as follow:

1. It prepares a parity check matrix $\mathbf{H} \in \mathbb{Z}_p^{(n-t+1)\times(n+1)}$ of the RS codes. Next, it obtains a random vector $\mathbf{v} = H_3(CL, n-t+1) \in \mathbb{Z}_p^{n-t+1}$ and computes $\mathbf{w}^\top = \mathbf{v}^\top \cdot \mathbf{H}$ where $\mathbf{w} = (w_0, \ldots, w_n)^\top$.
2. It computes $C_A = C_1^{w_0} \prod_{i=1}^n A_i^{w_i}$, $C_B = C_1^{w_0} \prod_{i=1}^n B_i^{w_i}$, $H_A = h^{w_0} \prod_{i=1}^n X_i^{w_i}$, and $H_B = h^{w_0} \prod_{i=1}^n Y_i^{w_i}$.
3. It computes $P_i = A_i / B_i$ and $Q_i = X_i / Y_i$ for $i \in [n]$.
4. It selects random $u \in \mathbb{Z}_p$ and sets $U = g^u, U_A = H_A^u, U_B = H_B^u$, and $U_i = P_i^u$ for $i \in [n]$.
5. And it obtains $e = H_2(CL, C_A, C_B, H_A, H_B, (P_1, Q_1), \ldots, (P_n, Q_n), U, U_A, U_B, U_1, \ldots, U_n)$.
6. It calculates $z = u + er$ and outputs a proof $\psi = (e, z)$.

NIZK-WF.Verify$(CRS_{WF}, s'_{WF}, \psi)$: Let $CL = (C_1, C_2, (A_1, B_1), \ldots, (A_n, B_n), t)$, $PK_i = (X_i, Y_i, \pi_i)$, and $\psi = (e, z)$. It proceeds as follows;

1. It prepares a parity check matrix $\mathbf{H} \in \mathbb{Z}_p^{(n-t+1)\times(n+1)}$ of the RS codes. It obtains a random vector $\mathbf{v} = H_3(CL, n-t+1) \in \mathbb{Z}_p^{n-t+1}$ and computes $\mathbf{w}^\top = \mathbf{v}^\top \cdot \mathbf{H}$ where $\mathbf{w} = (w_0, \ldots, w_n)^\top$.
2. It computes $C_A = C_1^{w_0} \prod_{i=1}^n A_i^{w_i}$, $C_B = C_1^{w_0} \prod_{i=1}^n B_i^{w_i}$, $H_A = h^{w_0} \prod_{i=1}^n X_i^{w_i}$, and $H_B = h^{w_0} \prod_{i=1}^n Y_i^{w_i}$.
3. It computes $P_i = A_i / B_i$ and $Q_i = X_i / Y_i$ for $i \in [n]$.
4. It computes $U' = g^z \cdot C_2^{-e}$, $U'_A = H_A^z \cdot C_A^{-e}, U'_B = H_B^z \cdot C_B^{-e}$, and $U'_i = Q_i^z \cdot P_i^{-e}$ for $i \in [n]$.
5. It checks $e \overset{?}{=} H_2(CL, C_A, C_B, H_A, H_B, (P_1, Q_1), \ldots, (P_n, Q_n), U', U'_A, U'_B, U'_1, \ldots, U'_n)$.
6. If the check succeeds, it outputs 1. Otherwise, it outputs 0.

Completeness. Let $\mathbf{f} = (f(0), f(idx_1), \ldots, f(idx_n))^\top$ be a vector of polynomial values that are honestly generated and embedded in ciphertext elements CL. We have $\mathbf{v}^\top \cdot \mathbf{H} \cdot \mathbf{f} = \mathbf{w}^\top \cdot \mathbf{f} = f(0)w_0 + \sum_{i=1}^n f(idx_i)w_i = 0$ since $\mathbf{H} \cdot \mathbf{f} = 0$ where $\mathbf{H}$ is the parity check matrix. Thus, we obtain the following equations

$$
\begin{aligned}
C_A &= C_1^{w_0} \prod_{i=1}^n A_i^{w_i} = \left(h^r g^{f(0)}\right)^{w_0} \prod_{i=1}^n \left(g^{f(idx_i)} X_i^r\right)^{w_i} = \left(h^{w_0} \prod_{i=1}^n X_i^{w_i}\right)^r g^{f(0)w_0 + \sum_{i=1}^n f(idx_i)w_i} = H_A^r, \\
C_B &= C_1^{w_0} \prod_{i=1}^n B_i^{w_i} = \left(h^r g^{f(0)}\right)^{w_0} \prod_{i=1}^n \left(g^{f(idx_i)} Y_i^r\right)^{w_i} = \left(h^{w_0} \prod_{i=1}^n Y_i^{w_i}\right)^r g^{f(0)w_0 + \sum_{i=1}^n f(idx_i)w_i} = H_B^r, \\
Q_i &= A_i / B_i = (g^{f(idx_i) X_i^r}) / (g^{f(idx_i) Y_i^r}) = (X_i / Y_i)^r = P_i^r \text{ for } i \in [n].
\end{aligned}
$$

If $e = H_2(CL, C_A, C_B, H_A, H_B, (Q_1, P_1), \ldots, (Q_n, P_n), U, U_A, U_B, U_1, \ldots, U_n)$ and $z = u + er$, then it is easy to check that the following verification equations are satisfied as

$$
\begin{aligned}
&g^z = g^u \cdot (g^r)^e = U \cdot C_2^e, \;\; H_A^z = H_A^u \cdot (H_A^r)^e = U_A \cdot C_A^e, \;\; H_B^z = H_B^u \cdot (H_B^r)^e = U_A \cdot C_B^e, \\
&Q_i^z = Q_i^u \cdot (Q_i^r)^e = U_i \cdot ((X_i / Y_i)^r)^e = U_i \cdot (P_i)^e \text{ for } i \in [n].
\end{aligned}
$$

Soundness. Due to Lemma 3.1, the NIZK-WF system essentially proves the equality of discrete logarithms, as in our NIZK-EDL system. Therefore, according to [7], it is clear that this system guarantees the soundness property.

Zero-Knowledge. This property can be easily achieved for a simulated proof $\psi = (U, U_A, U_B, U_1, \ldots, U_n, z)$ if a simulator first selects random exponents $e, z \in \mathbb{Z}_p$ and sets elements $U = g^z \cdot C_2^{-e}$, $U_A = H_A^z \cdot C_A^{-e}$, $U_B = H_B^z \cdot C_B^{-e}$ and $U_i = Q_i^z \cdot P_i^{-e}$ for $i \in [n]$ by programming the random oracle. This simulated proof is identically distributed to the real one.

Table 2: Differences between Hybrid games from $\mathbf{G}_0$ to $\mathbf{G}_5$

Game	Session Key		Ciphertext Header				NIZK	Decap.
	K_0^*	K_1^*	C_1^*	$B_{t^*}^*$	$\dots$	$B_{n^*}^*$		Keys
$\mathbf{G}_0$	R	h^r	$h^r g^{f(0)}$	$g^{f(idx_{t^*})} Y_{t^*}^r$	$\dots$	$g^{f(idx_{n^*})} Y_{n^*}^r$	*Real*	x_i
$\mathbf{G}_1$	R	h^r	$h^r g^{f(0)}$	$g^{f(idx_{t^*})} Y_{t^*}^r$	$\dots$	$g^{f(idx_{n^*})} Y_{n^*}^r$	*Sim*	x_i
$\mathbf{G}_2$	R	h^r	$h^r g^{f(0)}$	R_{t^*}	$\dots$	R_{n^*}	*Sim*	x_i
$\mathbf{G}_3$	R	h^r	$h^r g^{f(0)}$	R_{t^*}	$\dots$	R_{n^*}	*Sim*	y_i
$\mathbf{G}_4$	R	h^r	R_1	R_{t^*}	$\dots$	R_{n^*}	*Sim*	y_i
$\mathbf{G}_5$	R	R_0	R_1	R_{t^*}	$\dots$	R_{n^*}	*Sim*	y_i

In our selective IND-CCA security proof, a challenge ciphertext header $CH^* = ((C_1^*, C_2^*, (A_1^*, B_1^*), \dots, (A_{n^*}^*, B_{n^*}^*), t^*), \psi^*)$ related a session key K_b^* ($b \in \{0,1\}$) changes from game $\mathbf{G}_0$ to $\mathbf{G}_5$, with the varying components indicated by grey boxes. Each of $R, R_0, \dots R_{n^*}$ denotes random replacement. In the NIZK column, *Real* indicates that all NIZK proofs given to an adversary are generated as real ones, while *Sim* indicates that they are generated by a zero-knowledge simulator. The Decap.keys column represents the secret key used to respond to share decapsulation queries. Components not shown in the table remain unchanged.

3.3.3 NIZK for the Proof of Equality of Discrete Logarithms (NIZK-EDL)

The underlying NIZK-EDL system for the relation $\mathcal{R}_{EDL} = \{(s_{EDL}, w_{EDL})\}$, where $s_{EDL} = (X_1 = g^x \wedge X_2 = g_2^x \wedge Y_1 = g^y \wedge Y_2 = g_2^y)$ and $w_{EDL} = (x, y)$, is described as follows:

NIZK-EDL.Prove$(CRS_{EDL}, s_{EDL}, w_{EDL})$: Let $CRS_{EDL} = (p, \mathbb{G}, g, g_2, H_4)$. It first select random exponents $u, v \in \mathbb{Z}_p$ and sets $U_1 = g^u, U_2 = g_2^u$, $V_1 = g^v, V_2 = g_2^v$. Next, it obtains $e = H_4(g, g_2, X_1, X_2, Y_1, Y_2, U_1, U_2, V_1, V_2)$ and calculates $z_1 = u + ex$ and $z_2 = v + ey$. It outputs a proof $\sigma = (e, z_1, z_2)$.

NIZK-EDL.Verify$(CRS_{EDL}, s_{EDL}, \sigma)$: Let $\sigma = (e, z_1, z_2)$. It computes $U_1' = g^{z_1} \cdot X_1^{-e}$, $U_2' = g_2^{z_1} \cdot X_2^{-e}$, $V_1' = g^{z_2} \cdot Y_1^{-e}$, and $V_2' = g^{z_2} \cdot Y_2^{-e}$ and checks that $e \overset{?}{=} H_4(g, g_2, X_1, X_2, Y_1, Y_2, U_1, U_2, V_1, V_2)$. If the check succeeds, it outputs 1. Otherwise, it outputs 0.

Since the NIZK-EDL system can be derived from [7], this system guarantees the properties of completeness, zero-knowledge, and soundness.

4 Security Analysis

In this section, we prove that our TKEM, proposed in Section 3.2, guarantees the selective IND-CCA security and the DC security.

Theorem 4.1. *The proposed TKEM is selective IND-CCA secure if the DDH assumption holds, the underlying NIZK systems are zero-knowledge, the NIZK-EDL is sound, and hash functions are modeled as random oracles.*

Proof. To prove the selective IND-CCA security of our TKEM, we define a sequence of hybrid games $\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_5$. As shown in Table 2, $\mathbf{G}_0$ is the original game defined in Definition 3.2, where an adversary $\mathcal{A}$ aims to distinguish which of $\{K_b^*\}_{b\in\{0,1\}}$ is associated with a given ciphertext header CH^*. We will show a gradual change from the initial game $\mathbf{G}_0$ to the final game $\mathbf{G}_5$, which is not detectable by any PPT algorithm (i.e., a distinguisher). In $\mathbf{G}_5$, the challenged CH^* is independent of both K_0^* and K_1^*. From the adversary's standpoint, there is no information that could break this security.

Let CH^* be composed of $((C_1^*, C_2^*, \{A_i^*, B_i^*\}_{i=1}^{n^*}, t^*), \psi^*)$ and let $\mathbf{S}_i$ be the event that $\mathbf{G}_i$ outputs 1. The detailed descriptions of hybrid games are given as follows:

Game $\mathbf{G}_0$: This game $\mathbf{G}_0$ is the original game. Thus, we obtain the following equation

$$\mathbf{Adv}_{TKEM}^{SE\text{-}IND\text{-}CCA}(\lambda) = \left|\Pr[\mathbf{S}_0] - 1/2\right|.$$

Game $\mathbf{G}_1$: This game $\mathbf{G}_1$ is the same as $\mathbf{G}_0$ except that proofs all NIZK proofs are replaced by simulated proofs that are generated without the corresponding witness. By the zero-knowledge property of each NIZK system, we obtain the following inequation

$$\left|\Pr[\mathbf{S}_1] - \Pr[\mathbf{S}_0]\right| \leq \mathbf{Adv}_{NIZK\text{-}PoK}^{ZK}(\lambda) + \mathbf{Adv}_{NIZK\text{-}WF}^{ZK}(\lambda) + \mathbf{Adv}_{NIZK\text{-}EDL}^{ZK}(\lambda).$$

Game $\mathbf{G}_2$: This game $\mathbf{G}_2$ is the same as $\mathbf{G}_1$ except that $\{B_i\}_{i\in[t^*,n^*]}$ in CH^* are replaced by $\{R_i\}_{i\in[t^*,n^*]}$, respectively, where $R_i \xleftarrow{\$} \mathbb{G}$. From Lemma 4.2, we have the following inequation

$$\left|\Pr[\mathbf{S}_2] - \Pr[\mathbf{S}_1]\right| \leq \mathbf{Adv}_{DDH}(\lambda).$$

Game $\mathbf{G}_3$: This game $\mathbf{G}_3$ is the same as $\mathbf{G}_2$ except for handling a share decapsulation query of (CH, PL, PK_i). To respond this query, CH is now decapsulated by using $y_i \in SK_i$, instead of x_i. This modification is detectable if $\mathcal{A}$ creates a particular $CH = (\ldots, (A_i, B_i), \ldots)$ where B_i does not mirror A_i, but passes the **CHVerify** algorithm. However, this is possible with negligible probability due to the soundness property of NIZK-WF. Hence, we have the following inequation

$$\left|\Pr[\mathbf{S}_3] - \Pr[\mathbf{S}_2]\right| \leq \mathbf{Adv}_{NIZK\text{-}WF}^{Sound}(\lambda).$$

Game $\mathbf{G}_4$: This game $\mathbf{G}_4$ the same as $\mathbf{G}_3$ except that C_1 in CH^* is replaced by $R_1 \xleftarrow{\$} \mathbb{G}$. From Lemma 4.3, we obtain the following inequation

$$\left|\Pr[\mathbf{S}_4] - \Pr[\mathbf{S}_3]\right| \leq \mathbf{Adv}_{DDH}(\lambda).$$

Game $\mathbf{G}_5$: This game $\mathbf{G}_5$ is the same as $\mathbf{G}_4$ except that K_1^* is set to $R_0 \xleftarrow{\$} \mathbb{G}$. From Lemma 4.4, we obtain the following inequation

$$\left|\Pr[\mathbf{S}_5] - \Pr[\mathbf{S}_4]\right| \leq \mathbf{Adv}_{DDH}(\lambda).$$

In this final game $\mathbf{G}_5$, the advantage of $\mathcal{A}$ is zero, and we thus have $\Pr[\mathbf{S}_5] - 1/2 = 0$.

By combining all together, we can obtain the following inequation that bounds the advantage of $\mathcal{A}$ as

$$\begin{aligned}\mathbf{Adv}_{TKEM}^{SE\text{-}IND\text{-}CCA}(\lambda) &= \left|\Pr[\mathbf{S}_0] - 1/2\right| \leq \sum_{k=1}^{5} \left|\Pr[\mathbf{S}_{k-1}] - \Pr[\mathbf{S}_k]\right| + \left|\Pr[\mathbf{S}_5] - 1/2\right| \\ &\leq \mathbf{Adv}_{NIZK\text{-}PoK}^{ZK}(\lambda) + \mathbf{Adv}_{NIZK\text{-}WF}^{ZK}(\lambda) + \mathbf{Adv}_{NIZK\text{-}EDL}^{ZK}(\lambda) + \mathbf{Adv}_{NIZK\text{-}WF}^{Sound}(\lambda) + 3\mathbf{Adv}_{DDH}(\lambda).\end{aligned}$$

This completes the proof. □

Lemma 4.2. *If the DDH assumption holds, then no PPT algorithm can distinguish between $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$.*

Proof. Given a DDH tuple $(g, \hat{g}_1 = g^{\alpha}, \hat{g}_2 = g^{\beta}, D)$, we can construct a reduction $\mathcal{R}$ that serves as a challenger to an adversary $\mathcal{A}$ in the selective IND-CCA security game. $\mathcal{R}$ interacts with $\mathcal{A}$ as follows:

Setup: $\mathcal{R}$ randomly chooses $\hat{r} \in \mathbb{Z}_p$ to compute $h = g^{\hat{r}}$. $\mathcal{R}$ generates public parameters $PP = (p, \mathbb{G}, g, h, H_0, H_1, H_2, H_3, H_4)$ as in the real scheme, except that these hash functions are modeled as random oracles. After $\mathcal{R}$ sends PP to $\mathcal{A}$, the adversary responds with t^* and PL_C^* such that $|PL_C^*| = t^*$. $\mathcal{R}$ first sets $\mathcal{U}_H$ as empty. $\mathcal{R}$ generates each public key PK_i of honest users as follows:

1. It chooses $x_i, \rho_{i,1}, \rho_{i,2} \in \mathbb{Z}_p$ randomly and stores them to deal with the challenge phase.
2. It computes $X_i = g^{x_i}$ and $Y_i = \hat{g}_1^{\rho_{i,1}} g^{\rho_{i,2}}$.
3. It simulates a proof π_i of NIZK-PoK.
4. It adds $PK_i = (X_i, Y_i, \pi_i)$ to $\mathcal{U}_H$.

Clearly, $|\mathcal{U}_H|$ is polynomially bounded. Upon receiving $\mathcal{U}_H$ from $\mathcal{R}$, $\mathcal{A}$ submits PL_R leading to a public key list $\mathcal{U}_C$ of corrupted users, such that $\mathcal{U}_C = PL_C^* \cup PL_R$. For $PK_j = (X_j, Y_j, \pi_j) \in \mathcal{U}_C$, if any π_j is invalid, $\mathcal{R}$ returns $\perp$. Otherwise, $\mathcal{R}$ extracts and stores the witness (x_j, y_j) corresponding to π_j. During the simulation, $y_i \in SK_i$ is implicitly set to $y_i = \alpha\rho_{i,1} + \rho_{i,2}$, where a is unknown to $\mathcal{R}$. Since $\rho_{i,1}$ and $\rho_{i,2}$ are both chosen randomly and independently of $\mathcal{A}$, this y_i is indistinguishable from the real one.

Query 1: The hash oracles are managed by returning a random value for a new input, without simulating NIZK proofs. When $\mathcal{A}$ requests a share decapsulation query for (PL, CH, PK_i), where **CHVerify**$(PP, PL, CH) = 1$ and $PK_i \in \mathcal{U}_H$, $\mathcal{R}$ returns $\mu_i =$ **ShareDecaps**(PP, PL, CH, SK_i) using $x_i \in SK_i$, as a response.

Challenge: $\mathcal{A}$ first sends $PL_H^* \in \mathcal{U}_H$ and t^*. Let $PL^* = PL_C^* \cup PL_H^*$ be rewritten by $\{PK_1^*, \ldots, PK_{n^*}^*\}$. For all PK_i^*, $\mathcal{R}$ obtains $H_0(PK_i^*) = idx_i^*$. $\mathcal{R}$ then randomly picks $s_0, \ldots, s_{t^*-1} \in \mathbb{Z}_p$ and generates a (t^*-1)-degree polynomial $f(\gamma) = \sum_{j=0}^{t^*-1} s_j \gamma^j$. The challenge ciphertext header $CH^* = (C_1^*, C_2^*, (A_1^*, B_1^*), \ldots, (A_{n^*}^*, B_{n^*}^*), t^*, \psi^*)$ is generated as follows:

1. It computes $C_1^* = \hat{g}_2^{\hat{r}} g^{s_0}$ and $C_2^* = \hat{g}_2$.
2. For $i \in [n^*]$, it computes $A_i^* = g^{f(idx_i^*)} \hat{g}_2^{x_i}$.
3. For $i \in [t^*-1]$, it computes $B_i^* = g^{f(idx_i^*)} \hat{g}_2^{y_{i,1}}$. For $i \in [t^*, n^*]$, it computes $B_i^* = g^{f(idx_i^*)} D^{\rho_{i,1}} \hat{g}_2^{\rho_{i,2}}$
4. It simulates a proof ψ^* of NIZK-WF.

Then $\mathcal{R}$ flips a bit $b \in \{0,1\}$ to produce a session key that is a random element $K_0^* \in \mathbb{G}$ (if $\hat{b} = 0$) or $K_1^* = \hat{g}_2^{\hat{r}}$ (otherwise). $\mathcal{R}$ finally sends both CH^* and K_b^* to $\mathcal{A}$.

Query 2: Same as Query 1 with the restriction that CH^* cannot be asked as a share decapsulation query.

Guess: At the end, $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$.

If $D = g^{ab}$, $\mathcal{R}$ simulates $\mathbf{G}_1$. This is because for $i \in [t^*, n^*]$, we have $B_i^* = g^{f(idx_i^*)} D^{\rho_{i,1}} \hat{g}_2^{\rho_{i,2}} = g^{f(idx_i^*)} g^{\alpha\beta\rho_{i,1} + \beta\rho_{i,2}} = g^{f(idx_i^*)} Y_i^{\beta}$. Otherwise, each B_i^*, where $i \in [t^*, n^*]$, has a random distribution that is independent of each other. Thus, $\mathcal{R}$ simulates $\mathbf{G}_2$. Based on this, any distinguisher $\mathbf{G}_1$ and $\mathbf{G}_2$ can be converted into a DDH solver, and thereby, $|\Pr[\mathbf{S}_1] - \Pr[\mathbf{S}_2]| \leq \mathbf{Adv}_{DDH}(\lambda)$. □

Lemma 4.3. *If the DDH assumption holds, then no PPT algorithm can distinguish between $\boldsymbol{G}_3$ and $\boldsymbol{G}_4$.*

Proof. Let $\mathcal{R}$ be a reduction acting as a challenger to a selective IND-CCA adversary $\mathcal{A}$. $\mathcal{R}$ interacts with $\mathcal{A}$ as follows:

Setup: $\mathcal{R}$ first generates $PP = (\mathbb{G}, p, g, h, H_0, H_1, H_2, H_3, H_4)$, where $h = g^{\hat{r}}$ for $\hat{r} \xleftarrow{\$} \mathbb{Z}_p$, the hash functions are modeled as random oracles, and the other components are as in the real scheme. Given PP, $\mathcal{A}$ submits t^* and $PL_C^* = \{PK_1^*, \ldots, PK_{t^*-1}^*\}$. After that, for $i \in [t^*-1]$, $\mathcal{R}$ sets $H_0(PK_i^*) = idx_i^*$, where idx_i^* is chosen at random, by programming the random oracle. $\mathcal{R}$ then computes the Lagrange basis $L_i^*(\gamma) = \frac{\gamma}{idx_i^*} \prod_{j \in [t^*-1] \setminus \{i\}} \frac{\gamma - idx_j^*}{idx_i^* - idx_j^*}$ and also computes $L_0^*(\gamma) = \prod_{j \in [t^*-1]} \frac{\gamma - idx_j^*}{-idx_j^*}$. Let $\mathcal{U}_H$ be an empty set. $\mathcal{R}$ proceeds as follows:

1. It chooses random $y_i, \rho_i \in \mathbb{Z}_p$ and stores them to deal with the challenge phase.
2. It computes $X_i = \hat{g}_1^{L_0^*(idx_i)} g^{\rho_i}$ and $Y_i = g^{y_i}$.
3. It simulates a proof π_i of NIZK-PoK.
4. It adds $PK_i = (X_i, Y_i, \pi_i)$ to $\mathcal{U}_H$

In this way, although $\mathcal{R}$ does not know $x_i = L_0^*(idx_i^*)\alpha + \rho_i$, this is well simulated as ρ_i is uniformly random from $\mathcal{A}$'s point of view. Once the generation of $\mathcal{U}_H$ is completed, $\mathcal{R}$ sends it to $\mathcal{A}$. In response, PL_R is provided by $\mathcal{A}$, which results in $\mathcal{U}_C = PL_C^* \cup PL_R$. For each $PK_j = (X_j, Y_j, \pi_j) \in \mathcal{U}_C$, if π_j is valid, $\mathcal{R}$ could obtain the corresponding witnesses (x_j, y_j) by extraction of the proofs. Otherwise, $\mathcal{R}$ outputs $\perp$.

Query 1: When $\mathcal{A}$ requests for some new input to the hash oracles, $\mathcal{R}$ returns a random value, except when simulating the NIZK proofs. To respond to a share decapsulation query for (PL, CH, PK_i), where $\mathbf{CHVerify}(PP, PL, CH) = 1$ and $PK_i \in \mathcal{U}_H$, $\mathcal{R}$ uses $y_i \in SK_i$ to return $\mu_i = \mathbf{ShareDecaps}(PP, PL, CH, SK_i)$.

Challenge: $\mathcal{A}$ requests a challenge for $PL_H^* \in \mathcal{U}_H$ with t^*. Consider $PL_H^* = \{PK_{t^*}^*, \ldots, PK_{n^*}^*\}$ and $H_0(PK_i^*) = idx_i^*$. $\mathcal{R}$ randomly picks $\eta, s_1, \ldots, s_{t^*-1} \in \mathbb{Z}_p$ and defines a degree t^*-1 polynomial $f(\gamma) = L_0^*(\gamma)(\eta - \alpha\beta) + \sum_{i=1}^{t^*-1} L_i^*(\gamma) s_i$. This polynomial can be constructed by interpolating t^* points such that $f(0) = \eta - \alpha\beta$ and $f(idx_i^*) = s_i$ for all $i \in [1, t^*-1]$. Now, $CH^* = (C_1^*, C_2^*, (A_1^*, B_1^*), \ldots, (A_{n^*}^*, B_{n^*}^*), t^*, \psi^*)$ is generated as follows:

1. It generates $C_1^* = \hat{g}_2^{\hat{r}} g^{\eta} D^{-1}$ and $C_2^* = \hat{g}_2$.
2. For $i \in [1, t^*-1]$, it computes $A_i^* = g^{s_i} \hat{g}_2^{x_i}$. For $i \in [t^*, n^*]$, it computes $A_i^* = g^{L_0^*(idx_i^*)\eta + \sum_{j=1}^{t^*-1} L_j^*(idx_i^*) s_j} \hat{g}_2^{\rho_i}$.
3. For $i \in [1, t^*-1]$, it computes $B_i^* = g^{s_i} \hat{g}_2^{y_i}$. For $i \in [t^*, n^*]$, it sets $B_i^* = R_i$ where $R_i \xleftarrow{\$} \mathbb{G}$.
4. It simulates a proof ψ^* of NIZK-WF.

Recall that $\mathcal{R}$ knows (x_i, y_i), where $i \in [1, t^*-1]$, extracted during the setup phase. Finally, $\mathcal{R}$ sends CH^* and K_b^*, where K_b^* is determined as $K_0^* = R \xleftarrow{\$} \mathbb{G}$ or $K_1^* = \hat{g}_2^{\hat{r}}$, depending on a random bit $b \in \{0, 1\}$.

Query 2: Same as Query 1, with the restriction that CH^* cannot be asked as a share decapsulation query.

Guess: At the end, $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$.

Consider $C_2^* = g^r$ (i.e., $\beta = r$). The simulation of (A_i^*, B_i^*) in CH^* is correct due to the following reason:

- For $i \in [1, t^*-1]$, $A_i^* = g^{s_i}\hat{g}_2^{x_i} = g^{f(idx_i^*)}X_i^r$ and $B_i^* = g^{s_i}\hat{g}_2^{y_i} = g^{f(idx_i^*)}Y_i^r$.
- For $i \in [t^*, n^*]$, $A_i^* = g^{L_0^*(idx_i^*)\eta + \sum_{j=1}^{t^*-1} L_j^*(idx_i^*)s_j}\hat{g}_2^{\rho_i} = g^{L_0^*(idx_i^*)(\eta-\alpha\beta)+\sum_{j=1}^{t^*-1} L_j^*(idx_i^*)s_j}(g^{L_0^*(idx_i^*)\alpha}g^{\rho_i})^{\beta}$

  $= g^{L_0^*(idx_i^*)f(0)+\sum_{j=1}^{t^*-1} L_j^*(idx_i^*)f(idx_j^*)}(\hat{g}_1^{L_0^*(idx_i^*)}g^{\rho_i})^{\beta} = g^{f(idx_i^*)}X_i^r$.

If $D = g^{ab}$, $\mathcal{R}$ simulates $\mathbf{G}_3$, since $C_1^* = \hat{g}_2^{\hat{r}}g^{\eta}(D^{-1}) = (g^{\beta})^{\hat{r}}g^{\eta-\alpha\beta} = h^r g^{f(0)}$ where $f(0) = \eta - \alpha\beta$. Otherwise, C_1^* is uniformly random, resulting in $\mathcal{R}$ which simulates $\mathbf{G}_4$. Therefore, any distinguisher $\mathbf{G}_3$ and $\mathbf{G}_4$ can be converted into a DDH solver. Thus $|\Pr[\mathbf{S}_3] - \Pr[\mathbf{S}_4]| \leq \mathbf{Adv}_{DDH}(\lambda)$. □

Lemma 4.4. *If the DDH assumption holds, then no PPT algorithm can distinguish between $\mathbf{G}_4$ and $\mathbf{G}_5$.*

Proof. Given a DDH tuple $(g, \hat{g}_1 = g^{\alpha}, \hat{g}_2 = g^{\beta}, D)$, a reduction $\mathcal{R}$ interacts with a selective IND-CCA adversary $\mathcal{A}$ as follows:

Setup: $\mathcal{R}$ sets $h = \hat{g}_1$ and generates public parameters $PP = (\mathbb{G}, p, g, h, H_0, H_1, H_2, H_3, H_4)$, where hash functions are modeled as random oracles. After $\mathcal{R}$ sends PP to $\mathcal{A}$, the adversary responds with t^* and $PL_C^* = \{PK_1^*, \ldots, PK_{t^*-1}^*\}$. $\mathcal{R}$ generates $\mathcal{U}_H$, initially set as empty, by following the process:

1. It randomly chooses $x_i, y_i \in \mathbb{Z}_p$ to compute $X_i = g^{x_i}$ and $Y_i = g^{y_i}$.
2. It simulates a proof π_i of NIZK-PoK.
3. Is adds $PK_i = (X_i, Y_i, \pi_i)$ to $\mathcal{U}_H$.

Since the tuple (X_i, Y_i) is computed as in the real scheme, this simulation is correct. For $\mathcal{U}_C = PL_C^* \cup PL_R$, where PL_R is provided by $\mathcal{A}$, $\mathcal{R}$ checks if the proof π_j of $PK_j \in \mathcal{U}_C$ is valid or not. If not, $\mathcal{R}$ outputs $\perp$. Otherwise, $\mathcal{R}$ extracts the witness (x_j, y_j) from π_j.

Query 1: The hash oracles are managed normally, by returning a random value for a new input except when simulating NIZK proofs. When $\mathcal{A}$ requests a share decapsulation query for (PL, CH, PK_i), where $\mathbf{CHVerify}(PP, PL, CH) = 1$ and $PK_i \in \mathcal{U}_H$, $\mathcal{R}$ returns $\mu_i = \mathbf{ShareDecaps}(PP, PL, CH, SK_i)$, especially using $y_i \in SK_i$.

Challenge: When $\mathcal{A}$ submits PL_H^* and t^*, $\mathcal{R}$ sets $PL^* = PL_C^* \cup PL_H^* = \{PK_1^*, \ldots, PK_{n^*}^*\}$ and obtains $H_0(PK_i^*) = idx_i^*$ for all PK_i^*. After that, $\mathcal{R}$ generates a (t^*-1)-degree polynomial $f(\gamma) = \sum_{j=0}^{t^*-1} s_j\gamma^j$, where $s_0, \ldots, s_{t^*-1} \in \mathbb{Z}_p$ are chosen randomly. $CH^* = (C_1^*, C_2^*, (A_1^*, B_1^*), \ldots, (A_{n^*}^*, B_{n^*}^*), t^*, \psi^*)$ is generated as follows:

1. It sets $C_1^* = R_1$, where $R_1 \xleftarrow{\$} \mathbb{G}$, and $C_2^* = \hat{g}_2$.
2. For $i \in [n^*]$, it computes $A_i^* = g^{f(idx_i^*)}\hat{g}_2^{x_i}$.
3. For $i \in [1, t^*-1]$, it computes $B_i^* = g^{f(idx_i^*)}\hat{g}_2^{y_i}$. For $i \in [t^*, n^*]$, it sets $B_i^* = R_i$, where $R_i \xleftarrow{\$} \mathbb{G}$.
4. Finally, it simulates a proof ψ^* of NIZK-WF.

Depending on a randomly chosen $b \in \{0,1\}$, $\mathcal{R}$ determines either $K_0^* \in \mathbb{G}$ or $K_1^* = D$. $\mathcal{R}$ then sends both CH^* and K_b^* to $\mathcal{A}$.

Query 2: Same as Query 1 with the restriction that CH^* cannot be asked as a share decapsulation query.

Guess: At the end, $\mathcal{A}$ outputs a guess b'.

If $D = g^{\alpha\beta}$, $\mathcal{R}$ simulates $\mathbf{G}_4$. This is because $K_1^* = (g^{\alpha})^{\beta} = h^r$ by letting $\beta = r$. Otherwise $\mathcal{R}$ simulates $\mathbf{G}_5$. Based on this, any distinguisher $\mathbf{G}_4$ and $\mathbf{G}_5$ can be converted into a DDH solver, and thus $|\Pr[\mathbf{S}_4] - \Pr[\mathbf{S}_5]| \leq \mathbf{Adv}_{DDH}(\lambda)$. □

Theorem 4.5. *The proposed TKEM is DC secure if the underlying NIZK-EDL and NIZK-WF systems are both sound.*

Proof. Suppose that there exists a PPT adversary $\mathcal{A}$ that can break our TKEM scheme in the DC security game. We can construct a reduction $\mathcal{R}$ (acting as a challenger) that interacts with $\mathcal{A}$ as follows:

Setup : $\mathcal{R}$ first gives $PP \leftarrow \mathbf{Setup}(1^{\lambda})$ to $\mathcal{A}$. It also give $\mathcal{U}_H$ that includes a poly-bounded number of $(PK_i = (X_i, Y_i), SK_i = (x_i, y_i)) \leftarrow \mathbf{KeyGen}(PP)$. In response, $\mathcal{R}$ is given $\mathcal{U}_C$ from $\mathcal{A}$.

Query : $\mathcal{A}$ may request share decapsulation queries for (CH, PL, PK_i), where $PL \subseteq \mathcal{U}_H \cup \mathcal{U}_C$ and $PK_i \in \mathcal{U}_H$. Then, $\mathcal{R}$ sends either $\perp$ or μ_i, which can be obtained by running $\mathbf{ShareDecaps}(PP, PL, CH, SK_i)$.

Forge : $\mathcal{A}$ outputs a ciphertext header CH^* for a public key list $PL^* = \{PK_1^*, \ldots, PK_{n^*}^*\} \subset \mathcal{U}_H \cup \mathcal{U}_C$, a threshold t^*, and two sets of shares $\{\mu_i\}_{i \in S}$ and $\{\mu_j'\}_{j \in S'}$, where $t^* \leq |S| = |S'| \leq n^*$ and $S, S' \subset [n^*]$.

Assume that $\mathcal{A}$ wins with $(CH^*, PL^*, t^*, \{\mu_i\}_{i\in S}, \{\mu_j'\}_{j\in S'})$ that meets the three winning condition in Definition 3.3. Let $CH^* = (C_1^*, C_2^*, (A_1^*, B_1^*), \ldots, (A_{n^*}^*, B_{n^*}^*), \psi^*)$, $\mu_i = (D_i, \sigma_i)$, and $\mu_j' = (D_j', \sigma_j')$. Due to the condition 3), $\mathbf{CHVerify}(PP, PL^*, CH^*) = 1$. This means CH is well-formed as ψ^* of NIZK-WF is sound. Therefore, we can write $C_1^* = h^r g^{f(0)}$, $C_2^* = g^r$, $A_i^* = g^{f(idx_i^*)}(g^{x_i})^r$, and $B_i^* = g^{f(idx_i^*)}(g^{y_i})^r$ for the same $r \in \mathbb{G}$ and the same polynomial $f(\gamma)$. The condition 2) ensures that σ_i and σ_j' of NIZK-EDL are valid. Since these proofs are all sound, we have the structures $A_i^*/D_i = (C_2^*)^{x_i}$ and $A_j^*/D_j' = (C_2^*)^{x_j}$, which can be rewritten as $D_i = A_i^*/(C_2^*)^{-x_i}$ and $D_j' = A_j^*/(C_2^*)^{-x_j}$, respectively. We now show that the condition 1) never holds, which contradicts the hypothesis: Consider $K = \mathbf{Combine}(PP, PL^*, CH^*, \{\mu_i\}_{i\in S})$ and $K' = \mathbf{Combine}(PP, PL^*, CH^*, \{\mu_j'\}_{j\in S'})$. Specifically, K is computed as follows:

$$\begin{aligned} K = C_1^* \prod_{i\in S}(D_i)^{-L_i(0)} &= (h^r g^{f(0)}) \prod_{i\in S}(g^{f(idx_i^*)} X_i^r/(g^r)^{-x_i})^{-L_i(0)} \\ &= (h^r g^{f(0)}) \prod_{i\in S}(g^{f(idx_i^*)})^{-L_i(0)} = (h^r g^{f(0)}) g^{-f(0)} = h^r. \end{aligned}$$

Similarly, we can have $K' = C_1^* \prod_{j\in S'}(D_j')^{-L_j(0)} = h^r$, leading to $K = K'$.

Consequently, $\mathcal{A}$ must compromise the soundness of NIZK-WF or NIZK-EDL to win the DC security game, resulting in the advantage of $\mathcal{A}$ as follows.

$$\mathbf{Adv}_{TKEM}^{DC}(\lambda) \leq \mathbf{Adv}_{NIZK-EDL}^{Sound}(\lambda) + \mathbf{Adv}_{NIZK-WF}^{Sound}(\lambda).$$

This completes the proof. □

5 Implementation

We present a proof-of-concept implementation of our TKEM, of which goal is to demonstrate the feasibility of our scheme, focusing on time evaluations. To do this, we utilize the Miracle-core library for implementing cryptographic primitives in Python, chosen for its robustness in handling operations over the secp256k1 curve. The experiments are conducted on a Windows PC with a 3.70GHz Intel(R) i7-8700 processor and

16GB DDR4 RAM. Tables 3 and 4 show the average execution times for each algorithm, obtained by running the corresponding algorithm multiple times with n receivers, where $n \in \{100, 200, 300, 400, 500\}$, under the threshold parameter $t = n/2$.

Table 3: Time evaluation of our TKEM

n	**Setup**(ms)	**Keygen**(s)†	**Encaps**(s)	**CHVerify**(s)	**ShareDecap**(s)†	**ShareVerify**(s)†	**Combine**(s)
100	0.178	4.498	14.702	8.617	197.032	9.784	0.769
200	0.175	8.518	32.950	21.115	386.064	19.183	1.821
300	0.160	13.344	57.310	40.281	580.950	29.385	3.271
400	0.184	17.602	94.334	69.213	745.641	37.154	4.772
500	0.160	20.777	137.187	109.650	930.298	46.200	6.770

$\dagger$ is measured as total execution time over n.

As shown in Table 3, the performance results indicate that the time complexity of all algorithms, except the **Setup** algorithm, increases approximately linearly with a small number of receivers. As n increases, the results are expected to follow the theoretical complexity; for example, the execution time of the **Encaps** algorithm grows proportionally to $O(n^3)$ due to the generation of a parity check matrix. We now focus on the **Encaps** algorithm, particularly Step 5 therein, and the **CHVerify** algorithm. These are involved in the NIZK-WF algorithms, which are crucial to the contributions of our TKEM. Table 4 provides the time required to generate and verify the proofs of NIZK-WF. From this, we observe that **NIZK-WF.Prove** algorithm becomes a bottleneck in the **Encaps** algorithm, but it takes within 2 min even for $n = 500$. Note that these results are not optimized, and thus, further optimization could improve these results significantly.

Table 4: Time evaluation of the employed NIZK-WF algorithms

n	**WF.Prove**(s)	**WF.Verify**(s)
100	7.505	8.617
200	18.538	21.115
300	35.842	40.281
400	65.168	69.213
500	102.583	109.650

6 Conclusion

In this paper, we have proposed a dynamic TKEM with a transparent setup that avoids the use of pairings. We have also proven the selective IND-CCA security and decapsulation consistency of our TKEM under the DDH assumption and the security of Fiat-Shamir transformed NIZK systems, respectively, in the ROM. The dynamic property of our TKEM allows a flexible selection of participants and thresholds, enhancing adaptability in decentralized systems. One limitation of our TKEM is that the size of the ciphertext header grows linearly with the number of receivers, $O(n)$. Future work will focus on optimizing the ciphertext header size to further improve the practicality of our approach. This research contributes to the advancement of threshold cryptography.

Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00518, Blockchain privacy preserving techniques based on data encryption).

References

[1] Shweta Agrawal, Damien Stehlé, and Anshu Yadav. Round-optimal lattice-based threshold signatures, revisited. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, volume 229 of *LIPIcs*, pages 8:1–8:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[2] Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner, and Chenzhi Zhu. Twinkle: Threshold signatures from DDH with full adaptive security. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024, Part I*, volume 14651 of *Lecture Notes in Computer Science*, pages 429–459. Springer, 2024.

[3] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022*, volume 13510 of *Lecture Notes in Computer Science*, pages 517–550. Springer, 2022.

[4] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. Nizks with an untrusted CRS: security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 777–804, 2016.

[5] Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Paper 2022/833, 2022. `https://eprint.iacr.org/2022/833`.

[6] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[7] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 1997.

[8] Rutchathon Chairattana-Apirom, Stefano Tessaro, and Chenzhi Zhu. Partially non-interactive two-round lattice-based threshold signatures. Cryptology ePrint Archive, Paper 2024/467, 2024. `https://eprint.iacr.org/2024/467`.

[9] Jung Hee Cheon. Discrete logarithm problems with auxiliary inputs. *J. Cryptol.*, 23(3):457–476, 2010.

[10] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.

[11] Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 678–709. Springer, 2023.

[12] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023*, pages 356–370. ACM, 2023.

[13] Cécile Delerablée and David Pointcheval. Dynamic threshold public-key encryption. In David A. Wagner, editor, *Advances in Cryptology - CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 317–334. Springer, 2008.

[14] Yvo Desmedt. Threshold cryptography. *Eur. Trans. Telecommun.*, 5(4):449–458, 1994.

[15] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. Mcfly: Verifiable encryption to the future made practical. In Foteini Baldimtsi and Christian Cachin, editors, *Financial Cryptography and Data Security - FC 2023*, volume 13950 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2023.

[16] Thomas Espitau, Shuichi Katsumata, and Kaoru Takemure. Two-round threshold signature from algebraic one-more learning with errors. Cryptology ePrint Archive, Paper 2024/496, 2024. `https://eprint.iacr.org/2024/496`.

[17] Thomas Espitau, Guilhem Niot, and Thomas Prest. Flood and submerse: Distributed key generation and robust threshold signature from lattices. Cryptology ePrint Archive, Paper 2024/959, 2024. `https://eprint.iacr.org/2024/959`.

[18] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[19] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. Cryptology ePrint Archive, Paper 2023/567, 2023. `https://eprint.iacr.org/2023/567`.

[20] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. Cryptology ePrint Archive, Paper 2024/263, 2024. `https://eprint.iacr.org/2024/263`.

[21] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2008.

[22] Kamil Doruk Gür, Jonathan Katz, and Tjerand Silde. Two-round threshold lattice-based signatures from threshold homomorphic encryption. In Markku-Juhani O. Saarinen and Daniel Smith-Tone, editors, *Post-Quantum Cryptography - PQCrypto 2024, Part II*, volume 14772 of *Lecture Notes in Computer Science*, pages 266–300. Springer, 2023.

[23] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[24] Chelsea Komlo and Ian Goldberg. FROST: flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020*, volume 12804 of *Lecture Notes in Computer Science*, pages 34–65. Springer, 2020.

[25] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 427–437. ACM, 1990.

[26] Rafaël Del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarinen. Threshold raccoon: Practical threshold signatures from standard lattice assumptions. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024, Part II*, volume 14652 of *Lecture Notes in Computer Science*, pages 219–248. Springer, 2024.

[27] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: robust asynchronous Schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022*, pages 2551–2564. ACM, 2022.

[28] Yusuke Sakai, Keita Emura, Jacob C. N. Schuldt, Goichiro Hanaoka, and Kazuo Ohta. Constructions of dynamic and non-dynamic threshold public-key encryption schemes with decryption consistency. *Theor. Comput. Sci.*, 630:95–116, 2016.

[29] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[30] Victor Shoup. Iso 18033-2: An emerging standard for public-key encryption. *http://shoup. net/iso/*, 2004.

[31] Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 628–658. Springer, 2023.

[32] Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation, EUROSAM '79*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.