Security Strengthening of Threshold Symmetric Schemes*

Ehsan Ebrahimi

Department of Computer Science & SnT
University of Luxembourg

Abstract. In this paper, we study the security definitions of various threshold symmetric primitives. Namely, we analyze the security definitions for threshold pseudorandom functions, threshold message authentication codes and threshold symmetric encryption. In each case, we strengthen the existing security definition, and we present a scheme that satisfies our stronger notion of security. In particular, we propose *indifferentiability* definition and *IND-CCA2* definition for a threshold pseudorandom function and a threshold symmetric encryption scheme, respectively. Moreover, we show that these definitions are achievable. Notably, we propose the first *IND-CCA2* secure threshold symmetric encryption scheme.

Keywords: Threshold Pseudorandom Function, Threshold Message Authentication Code, Threshold Symmetric Encryption

1 Introduction

A cryptographic secret key is a piece of information that is used in a cryptographic system as an input to a sub-algorithm whose action remains hidden to an observer that does not possess this secret-key. That is, the security of the system relies on the secrecy of the secret key and if an attacker compromises this secret key, the whole system renders not-secure. One way to protect the secret-key is Threshold Cryptography [DF89, SDFY94, NPR99, MPS+02, AMMR18, BGG+18, CGMS21, LST21, DMV22, ASY22, BS23] in which the secret key is split among multiple parties in order to protect the key. For a threshold functionality $\mathcal{F}$ with the parameters n, t, each party P_i for $i \in [n]$ receives its share of the secret key sk_i and is able to compute a partial evaluation of $\mathcal{F}$ on an input x ($\mathcal{F}_i(x)$). Then any coalition set S with at least t members is able to compute the evaluation of $\mathcal{F}$ on x using $\{\mathcal{F}_i(x)\}_{i\in S}$. In addition, any corrupt coalition set with less than t members should not be able to eventuate $\mathcal{F}(x)$. That is, the adversary needs to compromise at least t parties to gain information.

In contrast to threshold public-key functionalities [DF89], the threshold symmetric primitives are proposed recently and they are quite young. Consequently, their security definitions and constructions are not well-studied yet. An example would be a symmetric encryption method introduced in 2018 [AMMR18],

* A discussion about post-quantum security is included in the Appendix A.

lacking any CCA security definition. In this paper, we analyze and strengthen the existing security definitions for threshold symmetric primitives, and in each case, we propose a construction to satisfy our security notion. Namely, we analyze the security definitions for threshold pseudorandom functions, threshold message authentication codes and threshold symmetric encryption. In particular, we propose *indifferentiability* definition and *IND-CCA2* definition for a threshold pseudorandom function and a threshold symmetric encryption scheme, respectively.

To best of our knowledge, there is no threshold symmetric encryption scheme which satisfies our *IND-CCA2* security notion (see Section 5.2), that is, we aim to propose the first the first *IND-CCA2* secure threshold symmetric encryption scheme.

1.1 Our Contribution

- We propose a stronger security notion for a threshold pseudorandom function. Our definition is simulation based and it is called *indifferentiability*.
- We show that our *indifferentiability* definition is strictly stronger than the *pseudorandomness* definition [AMMR18].
- We show that the NPR threshold pseudorandom function [NPR99] satisfies our definition.
- We strengthen the unforgeability definition for a randomized threshold message authentication code.
- We propose an *IND-CCA2* security definition for a threshold symmetric encryption which is an improvement of the *message privacy* definition proposed in [AMMR18].
- We show that our *IND-CCA2* security definition is achievable.

1.2 Technical Overview

In this subsection we present a technical overview of our work.

1.3 Threshold Pseudorandom Function

Definitional Discussion. We argue that the existing security definition for a threshold pseudorandom function TPRF in [AMMR18] is not aligned with our intuitive expectation of the "pseudorandomness" [KL07]. That is, the pseudorandomness definition in [AMMR18] (Definition 9) is reminiscent of the *unpredictability* of pseudorandom number generators rather than the "pseudorandomness". In more details, in Definition 9 the adversary may submit a challenge input x for which he has not collected enough partial information to compute $\mathrm{TPRF}(x)$. Then, the challenger either responds to x with $\mathrm{TPRF}(x)$ or with a random value. Finally, the adversary's goal is to distinguish between these two cases. However, in the usual definition of "pseudorandomness" for a function f, an adversary that has oracle access to either f or a truly random function,

should not tell the difference.

Our Proposed Definition: Indifferentiability. We strengthen Definition 9 in this paper. Intuitively, we desire a security definition that guarantees a threshold pseudorandom function TPRF and its partial evaluations for non-corrupted keys look like truly random functions. That is, assuming that TPRF is constructed from the function family $\{f_{\mathsf{sk}_i}\}_i$, TPRF and f_{sk_i} for a non-corrupted key sk_i should be indistinguishable from truly random functions F and f, respectively.

We embed this intuition in our *indifferentiability* definition (Definition 14). Namely, in the real world the adversary is allowed to query both TPRF and $\{f_{\mathsf{sk}_i}\}_i$ functions. In the ideal world, we require the existence of a stateful simulator Sim that can simulate the real world interaction without using the non-corrupted keys with the lazy sampling approach.[1] In our convention, the simulator Sim should satisfy the following two conditions:

- Sim is stateful and it starts with empty databases. Whenever the simulator answers a query, it stores the input/output of this query in its corresponding database.
- Sim simulates the random functions f_i and F with the lazy sampling approach. That is, the simulator **has to** return a uniformly random value if the output of a query is not determined from its database.

Implication and Achievability. We show that our indifferentiability definition is strictly stronger than the pseudorandomness definition in [AMMR18] (see Theorem 2):

- Vaguely speaking, when there exists a stateful simulator Sim that using the lazy sampling approach can simulate the adversarial queries, then the challenge query x in the pseudorandomness definition (Definition 9) can be responded by a random value since the adversary does not have enough information to compute $\mathrm{TPRF}(x)$.
- We propose a TPRF which is secure with respect to Definition 9 and it is insecure with respect to our definition. The high-level idea is to commit to a value $\mathrm{TPRF}(x)$ in the setup phase using a hiding and binding commitment scheme, in a way that this commitment can only be opened if one learns t partial evaluation of TPRF on x. The security with respect to Definition 9 holds by the hiding property of the commitment scheme. However, for an adversary which first queries x to $\mathrm{TPRF}(x)$, then it queries t partial evaluation of TPRF on x, there is no successful simulator with respect to our definition. The reason is that the simulator has to return a random value y to respond to the first query. And the adversary checks if $[y = \mathrm{TPRF}(x)]$ once it has enough information to open the commitment. By the hiding and binding property of the commitment scheme, the simulator can not bypass this check.

[1] The lazy sampling approach can be seen as an efficient simulation of truly random functions F and f in our definition.

We show that our definition is achievable. Specifically, we show that the NPR threshold pseudorandom function [NPR99] fulfills our *indifferentiability* definition if the underlying function family $\{\mathtt{PRF}_{\mathsf{sk}_i}\}_i$ is pseudorandom. See Theorem 3 for more details.

1.4 Threshold Message Authentication Code

Definitional Discussion. In the computational model of [MPS+02], the challenger is able to observe all pairs of messages and tags that are queried by the adversary. Then, the adversary wins the existential forgery game if he is able to output a new valid pair of message and tag. In other words, in their model the adversary is not able to actively participate in the forgery game with the help of a corrupt party. We argue that this is not a realistic restriction, because the adversary can initiate an evaluation of a `tag` on a message m with the help of a corrupt party. Therefore, the resulting `tag` would not be accessible by the challenger, specifically, for a randomized threshold message authentication code. Currently, we are not aware of a threshold message authentication code that is randomized, though, we present our definition for completeness and for the prospect of inventing such a construction in the future. (Note that randomized message authentication codes are well-motivated and constructed [JJV02, DKPW12].)

Our Existential Unforgeability Definition. We embed this adversarial scenario in our existential unforgeability definition. We use the same idea that appears in the authenticity definition for a threshold symmetric encryption in [AMMR18]. That is, for the queries that are initiated by a corrupt party j, the challenger increments a counter $\mathbf{ct}$ by the total number of non-corrupt shares that j receives. The queries that are initiated by a non-corrupt party can be stored in a list $\mathbf{L}$ by the challenger. The challenger defines $g := t - |C|$ to be the gap between the threshold value t and the number of corrupt parties. At the end, the adversary wins if he outputs more than $\mathbf{ct}/g$ valid pairs (m_i, tag_i) not in $\mathbf{L}$.[2] (See Definition 15 for more information.)

A threshold pseudorandom function TPRF immediately gives us a threshold message authentication code. (See Figure 3.) We show that if TPRF satisfies the pseudorandomness and correctness (Definition 8), then the protocol in Figure 3 satisfies the *existential unforgeability*.

1.5 Threshold Symmetric Encryption

Definitional Discussion. The *message privacy* definition (Definition 12 [AMMR18]) for a threshold symmetric encryption scheme consists of `indirect decryption queries` that are allowed to be initiated only by a non-corrupt party and their outputs are not communicated to the adversary. In other words, the adversary

[2] This idea of making q MAC queries and outputting $q+1$ valid pairs at the end has been appeared before in the context of superposition queries [BZ13a, BZ13b].

is not able to initiate a decryption query with the help of a corrupt party. We argue that this is not a real-world attack scenario because an adversary that has corrupted a subset of parties can control their behaviors. Therefore, potentially the adversary can initiate a decryption query with the help of a corrupt party. We address this realistic adversarial scenario in our paper and strengthen the message privacy definition.

Our Proposed Definition: IND-CCA2 Security. We allow the adversary to make the encryption and decryption queries. These queries can be initiated by any party. In more details, in contrast to the *message privacy* definition, in our definition a corrupt party can initiate a decryption query and the output of a decryption query initiated by a non-corrupt party would be communicated to the adversary in our definition. (See Definition 16 for more details.)

The adversary submits two challenge messages m_0, m_1 and the challenger encrypts one of these two messages and sends the resulting ciphertext c^* to the adversary. The adversary's goal is to determine which of m_0, m_1 has been encrypted. Trivially, the non-corrupt parties should not participate in the decryption of c^* if a corrupt party initiates a decryption query on c^* in the post-challenge phase. (Otherwise, no encryption scheme satisfies the security definition.)

Implication and Achievability. It is obvious that our definition is stronger than the *message privacy* definition.

We argue that the well-known approach to construct an IND-CCA2 secure symmetric encryption scheme (the Encrypt-then-Mac approach) might not work in the distributed setting. At a high-level, since the verification of the message authentication code is a distributed algorithm, a corrupt party can learn a valid tag on a value m through a decryption query. In more details, let us define a threshold symmetric encryption scheme Π_{TSE} that its encryption algorithm on an input m returns $c = (c_1, c_2)$ where $c_1 := \mathrm{Enc}_{\mathrm{TSE}'}(m)$ and $c_2 := \mathrm{TM.Mac}(\mathrm{Enc}_{\mathrm{TSE}'}(m))$. The adversary can alter the challenge ciphertext c_1^* to get a valid ciphertext c_1' and initiate a decryption query with the help of a corrupt party j on (c_1', c_2). The decryption algorithm first checks if c_2 is a correct tag for c_1' or it is not. During this verification, the party j may learn about a correct tag tag on c_1'. That is, the party j may learn $\mathsf{tag} = \mathrm{TM.Mac}(c_1')$. Consequently, the adversary can submit (c_1', tag) as decryption query and use it to determine which of the two messages m_0, m_1 has been encrypted in the challenge phase.

Fortunately, we show that the Encrypt-then-Sign approach will result in an IND-CCA2 secure threshold symmetric encryption scheme. We construct a threshold symmetric encryption scheme from a threshold pseudorandom function, a symmetric encryption scheme and a signature scheme. (See Figure 4.) We show that our scheme satisfies the IND-CCA2 security if the underlying TPRF is pseudorandom, the symmetric encryption scheme is one-time IND-CPA secure and the signature scheme satisfies the strong unforgeability. (See Theorem 5 for more details.) The difference between the Encrypt-then-Sign approach with

the Encrypt-then-Mac approach is that the verification of the signature scheme is not distributed. In more details, each party evaluating an encryption on a message m signs the resulting ciphertext with its private-key. Then, anyone can check if this is a valid signature by the corresponding public-key when it is submitted as a decryption query. This bypasses the attack described above for the Encrypt-then-Mac approach.

2 Preliminaries

We say f_k is a pseudorandom function if it is indistinguishable from a truly random function for any probabilistic polynomial-time (PPT) adversary.

Definition 1 (Pseudorandom Function). *Let λ be the security parameter. Let $F := \{f_k\}_k$ be a family of keyed functions $f_k : M \to N$ where k is a bit string of length n. The size of M, N and the value n may depend on the security parameter. We say F is a pseudorandom function family if for any* PPT *adversary $\mathcal{A}$:*

$$|\Pr[b = 1 : k \xleftarrow{\$} \{0,1\}^n, b \leftarrow \mathcal{A}^{f_k}] - \Pr[b = 1 : f \xleftarrow{\$} \{all\ f : M \to N\}, b \leftarrow \mathcal{A}^{f}]| \leq \mathsf{negl}(\lambda),$$

where $\mathcal{A}^O$ has access to the oracle O.

We define one-time IND-CPA security for a symmetric encryption scheme.

Definition 2. *We say* $\mathrm{SYM} := (\mathrm{Sym.Gen}, \mathrm{Sym.Enc}, \mathrm{Sym.Dec})$ *is one-time IND-CPA secure if for any* PPT *adversary $\mathcal{A}$:*

$$|\Pr[b = 1 :\ m_0, m_1 \leftarrow \mathcal{A},\ c^* \leftarrow \mathrm{Sym.Enc}_{\mathsf{sk}}(m_0),\ b \leftarrow \mathcal{A}(c^*)] - \Pr[b = 1 :\ m_0, m_1 \leftarrow \mathcal{A},\ c^* \leftarrow \mathrm{Sym.Enc}_{\mathsf{sk}}(m_1),\ b \leftarrow \mathcal{A}(c^*)]| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{sk} \leftarrow \mathrm{Sym.Gen}(\lambda)$.

Definition 3 (Strong Unforgeability). *A signature scheme* $\mathrm{SignScheme} := (\mathrm{SGen}, \mathrm{Sign}, \mathrm{Verif})$ *is strong unforgeable if for any* PPT *adversary $\mathcal{A}$, and any* $(\mathsf{pk}, \mathsf{sk})$ *generated by* SGen*, the following holds:*

$$\Pr[\mathrm{Verif}_{\mathsf{pk}}(m, \theta) = 1 \wedge (m, \theta) \notin \mathbf{L} : (m, \theta) \leftarrow \mathcal{A}^{\mathrm{Sign}_{\mathsf{sk}}}(\mathbf{L})] \leq \mathsf{negl}(\lambda),$$

where $\mathbf{L}$ *is a list to store the $\mathcal{A}$'s signature queries to* $\mathrm{Sign}_{\mathsf{sk}}$.

Definition 4 (Commitment Scheme). *A commitment scheme consists of three polynomial-time algorithms* Gen*,* Com *and* Ver *described below.*

- *The key generating algorithm* Gen *that on the input of the security parameter 1^λ returns a public-key* pk_{com}.

– *The commitment algorithm* Com *on the inputs* pk_{com} *and a message* m *chooses a randomness* r *and returns* $c := \mathrm{Com}(\mathsf{pk}_{com}, m; r)$ *and the corresponding opening information* ω.
– *The verification algorithm* Ver *on the inputs* pk_{com}, c, ω *and* m, *either accepts* ($b = 1$) *or rejects* ($b = 0$).

The scheme has the correctness property, that is, the verification algorithm returns 1 *with the probability* 1 *if* c, ω *are the output of* Com*:*

$$\Pr[b = 1 : \mathsf{pk}_{com} \leftarrow \mathrm{Gen}(1^\lambda), (c, \omega) \leftarrow \mathrm{Com}(\mathsf{pk}_{com}, m), b \leftarrow \mathrm{Ver}(\mathsf{pk}_{com}, c, \omega, m)] = 1.$$

We define hiding and binding properties of a commitment scheme against a PPT adversary.

Definition 5. *We say a commitment scheme* $(\mathrm{Gen}(1^\lambda), \mathrm{Com}, \mathrm{Ver})$ *is statistically hiding if for any* $\mathsf{pk}_{com} \leftarrow \mathrm{Gen}(1^\lambda)$, *for any two messages* m_1, m_2 *and for any distinguisher* $\mathcal{D}$

$$|\Pr[\mathcal{D}(\mathsf{pk}_{com}, c_1) = 1 : (c_1, \omega_1) \leftarrow \mathrm{Com}_{\mathsf{pk}_{com}}(m_1)] - \Pr[\mathcal{D}(\mathsf{pk}_{com}, c_2) = 1 : (c_2, \omega_2) \leftarrow \mathrm{Com}_{\mathsf{pk}_{com}}(m_2)]| \leq \mathtt{neg}(\lambda).$$

Definition 6. *A commitment scheme* $(\mathrm{Gen}(1^\lambda), \mathrm{Com}, \mathrm{Ver})$ *is computationally binding if for any commitment* c, *and any* PPT *adversary* $\mathcal{A}$

$$|\Pr[\mathrm{Ver}(\mathsf{pk}_{com}, c, m_1, \omega_1) = 1 \wedge \mathrm{Ver}(\mathsf{pk}_{com}, c, m_2, \omega_2) = 1 \wedge m_1 \neq m_2 : \mathsf{pk}_{com} \leftarrow \mathrm{Gen}(1^\lambda), (m_1, \omega_1, m_2, \omega_2) \leftarrow \mathcal{A}(c, \mathsf{pk}_{com})]| \leq \mathtt{neg}(\lambda).$$

2.1 Threshold Pseudorandom Function

Definition 7 (Threshold Pseudorandom Function [AMMR18]). *A threshold pseudorandom function* TPRF *is a tuple of three algorithms* TPRF = (Setup, Eval, Combine) *described below:*

– *The* Setup *algorithm generates* n *secret keys* $(\mathsf{sk}_1, \cdots, \mathsf{sk}_n)$ *and public parameters* pp. *The* i*-th secret key* sk_i *is given to party* i. *That is,* $((\mathsf{sk}_1, \cdots, \mathsf{sk}_n), pp) \leftarrow \mathrm{Setup}(1^\lambda, n, t)$.
– *The* Eval *algorithm generates pseudo-random shares for a given value* x. *Party* i *computes the* i*-th share* z_i *for a value* x *by running* Eval *with* sk_i, x *and* pp. *That is,* $z_i \leftarrow \mathrm{Eval}(\mathsf{sk}_i, x, pp)$.
– *The Combine algorithm combines the partial shares* $\{z_i\}_{i \in S}$ *from parties in the set* S *to generate a value* z. *If the algorithm fails, its output is denoted by* $\perp$. $\mathrm{Combine}(\{(i, z_i)\}_{i \in S}; pp) =: z / \perp$.

The TPRF = (Setup, Eval, Combine) *has to fulfill the consistency property in which with a high probability* Combine *executed on two different sets* $\{(i, z_i)\}_{i \in S}$, $\{(j, z_j)\}_{j \in S'}$ *(generated honestly for two sets* S, S' *of size at least* t*) will result into the same value.*

Definition 8 (Correctness [AMMR18]). *A threshold pseudorandom function* TPRF = (Setup, Eval, Combine) *is correct if for all* PPT *adversary* $\mathcal{A}$ *there exists a negligible function* neg *such that the following game outputs* 0 *with a probability at most* $\mathsf{neg}(\lambda)$.

- ***Initialization.*** *Run* Setup *algorithm to generate* n *secret keys and public information* pp: $((\mathsf{sk}_1, \cdots, \mathsf{sk}_n), pp) \leftarrow \text{Setup}(1^\lambda, n, t)$. *Send* pp *to* $\mathcal{A}$.
- ***Corruption.*** *Receive the set of corrupt parties* C *from* $\mathcal{A}$, *where* $|C| < t$. *Give the secret keys* $\{\mathsf{sk}_i\}_{i \in C}$ *of these parties to* $\mathcal{A}$.
- ***Evaluation.*** *In response to* $\mathcal{A}$*'s evaluation query* (Eval, x, i) *for some* $i \in [n] \setminus C$, *return* $\text{Eval}(\mathsf{sk}_i, x, pp)$ *to* $\mathcal{A}$. *Repeat this step as many times as* $\mathcal{A}$ *desires.*
- ***Computation.*** *When* $\mathcal{A}$ *sends a set* S *of size at least* t, *an input* x^* *and shares* $\{(i, z_i^*)\}_{i \in S \cap C}$, *compute* $z_j \leftarrow \text{Eval}(\mathsf{sk}_j, x^*, pp)$ *for any* $j \in S$ *and compute* $z_i' \leftarrow \text{Eval}(\mathsf{sk}_i, x^*, pp)$ *for any* $i \in S \setminus C$. *Also, compute* $z := \text{Combine}(\{j, z_j\}_{j \in S}, pp)$ *and* $z^* := \text{Combine}(\{i, z_i'\}_{i \in S \setminus C} \cup \{(i, z_i^*)\}_{i \in S \cap C}, pp)$. *Finally, output* 1 *if* $z^* \in \{z, \perp\}$. *Otherwise, output* 0.

We present the pseudorandomness definition from [AMMR18].

Definition 9 (Pseudorandomness [AMMR18]). *A threshold function* TPRF = (Setup, Eval, Combine) *is pseudorandom if for all* PPT *adversaries* $\mathcal{A}$, *there exists a negligible function* neg *such that*

$$\Pr[\texttt{PseudoRanGame}(1^\lambda, 0) = 1] - \Pr[\texttt{PseudoRanGame}(1^\lambda, 1) = 1] \leq \mathsf{neg}(\lambda),$$

where `PseudoRanGame` *is defined below.*

$\texttt{PseudoRanGame}(1^\lambda, b)$:

- ***Initialization.*** *Run* $\text{Setup}(1^\lambda, n, t)$ *to get* $((\mathsf{sk}_1, \cdots, \mathsf{sk}_n), pp)$. *Give* pp *to* $\mathcal{A}$. *Initialize a list* $L := \emptyset$ *to record the set of values for which* $\mathcal{A}$ *may know the* TPRF *outputs.*
- ***Corruption.*** *Receive the set of corrupt parties* C *from* $\mathcal{A}$, *where* $|C| < t$. *Give the secret keys* $\{\mathsf{sk}_i\}_{i \in C}$ *of these parties to* $\mathcal{A}$. *Define the corruption gap as* $g := t - |C|$.
- ***Pre-challenge evaluation queries.*** *In response to* $\mathcal{A}$*'s evaluation query* (Eval, x, i) *for some* $i \in [n] \setminus C$, *return* $\text{Eval}(\mathsf{sk}_i, x, pp)$ *to* $\mathcal{A}$. *Repeat this step as many times as* $\mathcal{A}$ *desires.*
- ***Build the list.*** *Add an* x *to* L *if* $|\{i | \mathcal{A} \text{ made a } (\text{Eval}, x, i) \text{query}\}| \geq g$. *In other words, if* $\mathcal{A}$ *contacts at least* g *honest parties on a value* x, *it has enough information to compute the* TPRF *output on* x.
- ***Challenge.*** $\mathcal{A}$ *outputs* $(x^*, S, \{(i, z_i^*)\}_{i \in U})$ *such that* $|S| \geq t$ *and* $U \subseteq S \cap C$. *If* $x^* \in L$, *output* 0 *and stop. Let* $z_i \leftarrow \text{Eval}(sk_i, x, pp)$ *for* $i \in S \setminus U$ *and* $z^* := \text{Combine}(\{(i, z_i^*)\}_{i \in U} \cup \{(i, z_i)\}_{i \in S \setminus U}, pp)$. *If* $z^* = \perp$, *return* $\perp$. *Otherwise, if* $b = 0$, *return* z^* *and if* $b = 1$, *return a uniformly random value.*
- ***Post-challenge evaluation queries.*** *Same as the pre-challenge phase except that if* $\mathcal{A}$ *makes a query of the form* (Eval, x^*, i) *for some* $i \in [n] \setminus C$ *and* i *is the* g*-th party it contacted, then output* 0 *and stop.*
- ***Guess.*** *Finally,* $\mathcal{A}$ *returns a guess* b'. *Output* b'.

2.2 Threshold Symmetric Encryption

Definition 10 (Threshold Symmetric-key Encryption [AMMR18]). *A threshold symmetric-key encryption scheme* TSE *consists of (possibly randomized) algorithms* TSE.Setup, TSE.Enc *and* TSE.Dec *described below:*

- *The algorithm* TSE.Setup *takes the security parameter as input, and outputs n secret keys $\mathsf{sk}_1, \cdots, \mathsf{sk}_n$ and public parameters pp. The i-th secret key sk_i is given to party i.*
- $\mathrm{TSE.Enc}(\llbracket \mathsf{sk} \rrbracket_{[n]}; [j : m; S]; pp) \to [j : c/\ \perp]$ *: The algorithm* TSE.Enc *is a distributed protocol through which a party j encrypts a message m with the help of parties in a set S. At the end of the protocol, j outputs a ciphertext c (or $\perp$ to denote failure). All the other parties have no output.*
- $\mathrm{TSE.Dec}(\llbracket \mathsf{sk} \rrbracket_{[n]}; [j : c; S]; pp) \to [j : m/\ \perp]$ *: The algorithm* TSE.Dec *is a distributed protocol through which a party j decrypts a ciphertext c with the help of parties in a set S. At the end of the protocol, j outputs a message m (or $\perp$ to denote failure). All the other parties have no output.*

It is required that TSE *fulfills a consistency property: For any natural numbers n, t such that $t \leq n$, all $(\llbracket \mathsf{sk} \rrbracket_{[n]}; pp)$ output by* $\mathrm{TSE.Setup}(1^\lambda)$*, for any message m, any two sets $S, S' \subsetneq [n]$ such that $|S|, |S'| \geq t$, and any two parties $j \in S, j' \in S'$, if all the parties behave honestly, then there exists a negligible function neg such that*

$$\Pr[[j' : m] \leftarrow \mathrm{TSE.Dec}(\llbracket \mathsf{sk} \rrbracket_{[n]}; [j' : c; S']; pp) | \\ [j : c] \leftarrow \mathrm{TSE.Enc}(\llbracket \mathsf{sk} \rrbracket_{[n]}; [j : m; S]; pp)] \geq 1 - \mathsf{neg}(\lambda),$$

where the probability is taken over the randomness chosen in TSE.Enc *and* TSE.Dec*.*

The consistency property does not prevent an adversary to deliberately influence an encryption of a message m through the corrupt parties to get a cipheretxt c which later can be decrypted to a different message than m. Below, the correctness property is defined to address this scenario.

Definition 11 (Correctness [AMMR18]). *We say a threshold symmetric encryption scheme* $\mathrm{TSE} := (\mathrm{TSE.Setup}, \mathrm{TSE.Enc}, \mathrm{TSE.Dec})$ *is correct if for all* PPT *adversaries $\mathcal{A}$ there exists a negligible function neg such that the following game outputs 0 with a probability at most $\mathsf{neg}(\lambda)$.*

- ***Initialization.*** *The setup algorithm* TSE.Setup *is run to get $(\llbracket \mathsf{sk} \rrbracket_{[n]}, pp)$. Send pp to $\mathcal{A}$.*
- ***Corruption.*** *When receiving a set of corrupt parties C from $\mathcal{A}$ where $|C| < t$, send $\{\mathsf{sk}_i\}_{i \in C}$ to $\mathcal{A}$.*
- ***Encryption.*** *When receiving an encryption query on (m, j, S) where $j \in S \setminus C$ and $|S| \geq t$, initiate the* TSE.Enc *distributed algorithm from the party j on inputs m, S and pp. If j outputs $\perp$, stop and return $\perp$. Otherwise, send c to $\mathcal{A}$.*

– *Decryption.* *When receiving a decryption query on* (c, j', S') *where* $j' \in S' \setminus C$ *and* $|S'| \geq t$*, initiate the* TSE.Dec *distributed algorithm from the party* j' *on inputs* c, S' *and* pp*.*
– *Output.* *Return* 1 *if and only if* j' *outputs* m *or* $\perp$*.*

We say TSE *is* strongly-correct *if we differentiate the output as follows. The game outputs* 1 *if and only if:*

– *when all parties in* S' *behave honestly, then* j' *outputs* m*.*
– *when corrupt parties in* S' *deviate from the protocol, then* j' *outputs* m *or* $\perp$*.*

We import the *message privacy* from [AMMR18].

Definition 12 (Message Privacy). *A* TSE := (TSE.Setup, TSE.Enc, TSE.Dec) *satisfies* message privacy *if for all* PPT *adversaries* $\mathcal{A}$*, there exists a negligible function* neg *such that*

$$\Pr[\mathtt{MsgPriv}_{\mathrm{TSE},\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathtt{MsgPriv}_{\mathrm{TSE},\mathcal{A}}(1^\lambda, 1) = 1] \leq \mathsf{neg}(\lambda),$$

where $\mathtt{MsgPriv}_{\mathrm{TSE},\mathcal{A}}$ *is defined below.*

$\mathtt{MsgPriv}_{\mathrm{TSE},\mathcal{A}}(1^\lambda, b)$:

– *Initialization.* *Run* TSE.Setup$(1^\lambda, n, t)$ *to get* $((\mathsf{sk}_1, \cdots, \mathsf{sk}_n), pp)$*. Give* pp *to* $\mathcal{A}$*.*
– *Corruption.* *Receive the set of corrupt parties* C *from* $\mathcal{A}$*, where* $|C| < t$*. Give the secret keys* $\{\mathsf{sk}_i\}_{i \in C}$ *of these parties to* $\mathcal{A}$*.*
– *Pre-challenge encryption queries.* *In response to* $\mathcal{A}$*'s encryption query* (Encrypt, j, m, S) *where* $j \in S$ *and* $|S| \geq t$*, run an instance of the protocol* TSE.Enc *with* $\mathcal{A}$*. Note that when* $j \notin C$*, party* j *runs* TSE.Enc *and returns the output ciphertext to* $\mathcal{A}$*. Repeat this step as many times as* $\mathcal{A}$ *desires.*
– *Pre-challenge indirect decryption queries.* *In response to* $\mathcal{A}$*'s decryption query* (Decrypt, j, c, S) *where* $j \in S \setminus C$ *and* $|S| \geq t$*, party* j *initiates* TSE.Dec *with inputs* c *and* S*. Repeat this step as many times as* $\mathcal{A}$ *desires.*
– *Challenge.* $\mathcal{A}$ *outputs* $(j^*; m_0; m_1; S^*)$ *where* $|m_0| = |m_1|$*,* $j^* \in S^* \setminus C$ *and* $|S^*| \geq t$*. Party* j^* *initiates* TSE.Enc *with inputs* m_b *and* S^* *and sends the resulting ciphertext* c^* *to* $\mathcal{A}$*.*
– *Post-challenge encryption queries.* *Same as the pre-challenge phase.*
– *Post-challenge indirect decryption queries.* *Same as the pre-challenge indirect decryption queries.*
– *Guess.* *Finally,* $\mathcal{A}$ *returns a guess* b'*. Output* b'*.*

2.3 Threshold MAC

We define a threshold message authentication code below. Our definition is different from the definition in [MPS+02] since we present the verification algorithm as a distributed algorithm as well.

Definition 13 (Threshold Message Authentication Code (TMAC)). *A* TMAC *is a tuple of three algorithms* $\mathrm{TMAC} = (\mathrm{TM.Setup}, \mathrm{TM.Mac}, \mathrm{TM.Verif})$ *described below:*

- *The algorithm* $\mathrm{TM.Setup}$ *takes the security parameter as input, and outputs* n *secret keys* $\mathsf{sk}_1, \cdots, \mathsf{sk}_n$ *and public parameters* pp*. The* i*-th secret key* sk_i *is given to party* i*.*
- $\mathrm{TM.Mac}([\![\mathsf{sk}]\!]_{[n]}; [j : m; S]; pp) \to [j : \mathsf{tag}/\ \perp]$ *: The algorithm* $\mathrm{TM.Mac}$ *is a distributed protocol through which a party* j *computes a tag on a message* m *with the help of parties in a set* S*. At the end of the protocol,* j *outputs a tag* tag *(or* $\perp$ *to denote failure).*
- $\mathrm{TM.Verif}([\![\mathsf{sk}]\!]_{[n]}; [j : (m, \mathsf{tag}); S]; pp) \to [j : b]$ *: The algorithm* $\mathrm{TM.Verif}$ *is a distributed protocol through which a party* j *verifies a tag* tag *on a message* m *with the help of parties in a set* S*. At the end of the protocol,* j *outputs a bit* b *which indicates accept* ($b = 1$) *or reject* ($b = 0$)*.*

It is required that TMAC *fulfills a consistency property: For any natural numbers* n, t *such that* $t \leq n$*, all* $([\![\mathsf{sk}]\!]_{[n]}; pp)$ *output by* $\mathrm{TM.Setup}(1^\lambda)$*, for any message* m*, any two sets* $S, S' \subsetneq [n]$ *such that* $|S|, |S'| \geq t$*, and any two parties* $j \in S, j' \in S'$*, if all the parties behave honestly, then there exists a negligible function* neg *such that*

$$\Pr[[j' : 1] \leftarrow \mathrm{TM.Verif}([\![\mathsf{sk}]\!]_{[n]}; [j' : (m, \mathsf{tag}); S']; pp) | \\ [j : \mathsf{tag}] \leftarrow \mathrm{TM.Mac}([\![\mathsf{sk}]\!]_{[n]}; [j : m; S]; pp)] \geq 1 - \mathsf{neg}(\lambda),$$

where the probability is taken over the randomness chosen in $\mathrm{TM.Mac}$ *and* $\mathrm{TM.Verif}$*.*

3 TPRF from a PRF

In this section, first, we strengthen the pseudorandomness security definition (Definition 9). And then we show that the well-known NPR's threshold pseudorandom function [NPR99] satisfies our security definition if the underlying function family is pseudorandom.

Construction of a threshold pseudorandom function (TPRF) in [NPR99] is based on any pseudorandom function. In the setup phase, for $d := \binom{n}{k-1}$, random numbers $k_1, \cdots, k_d$ are chosen. Let $D_1, \cdots, D_d$ be the d distinct $(n-t+1)$-sized subsets of $[n]$. Then, the i-th random number is given to all parties in the set D_i. The TPRF is defined as $F_k(x) = \oplus_{i=1}^{d} \mathtt{PRF}_{k_i}(x)$, where $\mathtt{PRF}$ is any pseudorandom function. We call this NPR threshold function. (See Figure 1.) Since all the d keys are needed to compute F_k, no set S of parties with less than t members can compute F_k by itself since at least one of the $D_1, \cdots, D_d$ subsets does not intersect with S.[3]

[3] Note that d is the total number of $(n-t+1)$-sized subsets of $[n]$ and it is the total number of $(t-1)$-sized subsets of $[n]$. Then the map $D_i \to [n] \setminus D_i$ is a permutation between these subsets. Therefore, any subset S of size $t-1$ (or less) will not intersect with at least one $D_j (:= [n] \setminus S)$.

Public Parameters: Let $\mathtt{PRF} : \{0,1\}^k \times \{0,1\}^\lambda \to \{0,1\}^{\lambda'}$ be a pseudorandom function.

- $\mathsf{Setup}(1^\lambda, n, t) \to (\mathsf{sk}_1, \cdots, \mathsf{sk}_n; pp)$. For any $i \in [d]$, choose $k_i \xleftarrow{\$} \{0,1\}^k$ where $d = \binom{n}{n-t+1}$. Let $D_1, \cdots, D_d$ be the d distinct $(n-t+1)$-sized subset of $[n]$. For any $i \in [n]$, let $\mathsf{sk}_i := \{k_j | i \in D_j \text{ for } j \in [d]\}$. Set $pp := (f, D_1, \cdots, D_d)$ and gives (sk_i, pp) to the party i.
- $\mathsf{Eval}(\mathsf{sk}_i, x; pp) \to z_i$. For any $k \in \mathsf{sk}_i$, computes $h_{i,k} := \mathtt{PRF}_k(x)$. Set $z_i := \{h_{i,k}\}_{k \in \mathsf{sk}_i}$.
- $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}; pp) := z$. If $|S| < t$, it returns $\perp$. Parse $z_i = \{h_{i,k}\}_{k \in \mathsf{sk}_i}$. We define a set $\{h'_1, \cdots, h'_{d'}\} := \cup_{i \in S}\{h_{i,k}\}_{k \in \mathsf{sk}_i}$. If $d' > d$, returns $\perp$. Otherwise, set $z := \oplus_{i \in [d']} h'_i$.

Fig. 1. The NPR threshold pseudorandom function.

3.1 Our Definition: Indifferentiability

The Pseudorandomness definition in [AMMR18] (Definition 9) is reminiscent of the *unpredictability* of pseudorandom number generators. However, if one wants to reflect on the definition of a *pseudorandom function* to the threshold setting, a threshold pseudorandom function TPRF(:= (Setup, Eval, Combine)) and $\mathsf{Eval}(\cdot, \mathsf{sk}_i)$ for $i \in [n] \setminus C$ should look like random functions beyond the information that the adversary has learned through the corrupt parties and its evaluation queries. We embed this requirement in our definition.

Let C be a set of corrupt parties. The adversary in the real world is allowed to make two types of queries: 1) an evaluation query to a party $i \in [n] \setminus C$, that is, an (Eval, x, i) query. And 2) an evaluation query to TPRF itself, that is an (Eval, x) query. For any $i \in [n]$, let f_i be a truly random functions with the same domain and co-domain as $\mathsf{Eval}(\cdot, \mathsf{sk}_i)$. Let F be a truly random function with the same domain and co-domain as TPRF. We require the existence of a PPT simulator Sim that given $\{\mathsf{sk}_i\}_{i \in C}$ and oracle access to f_i and F is able to simulate the interaction of adversary with the primitive in the real world.[4] Obviously, the answers to the (Eval, x, i) and (Eval, x) queries should be consistent if the adversary queries $(\mathsf{Eval}, x, \cdot)$ beyond the threshold gap. In other words, whatever an adversary can learn from the threshold psudorandom function during the real execution of the function can be simulated by the simulator without using the non-corrupt keys.

In the definition below, we remove the use of f_i and F and instead, we put a convention for the simulator Sim:

- Sim is stateful and it starts with empty databases. Whenever the simulator answers a query, it stores the input/output of this query in its corresponding database.
- Sim simulates the random functions f_i and F with the lazy sampling approach. That is, the simulator **has to** return a uniformly random value if the output of a query is not determined from its database.

[4] This is inspired from the *Indifferentiability Framework* [MRH04].

Note that our convention to consider such a simulator Sim described above is not limiting. The reason is that in the security reductions the simulator should be an efficient algorithm. That is, one needs to implement the truly random functions f_i and F with the lazy-sampling approach. Therefore, we explicitly embed this efficiency in our definition.

Definition 14 (Indifferentiability). *A threshold pseudorandom function* $\mathrm{TPRF} = (\mathrm{Setup}, \mathrm{Eval}, \mathrm{Combine})$ *is indifferentiable from a truly random function if for any* PPT *adversaries* $\mathcal{A}$*, there exist a* PPT *(stateful) simulator* Sim *described above and a negligible function* neg *such that*

$$\Pr[\texttt{IndifGame}(1^\lambda, \textit{Real}) = 1] - \Pr[\texttt{IndifGame}(1^\lambda, \textit{Random}) = 1] \leq \mathsf{neg}(\lambda),$$

where `IndifGame` *games are defined below.*

$\underline{\texttt{IndifGame}(1^\lambda, \textit{Real}) :}$

- ***Initialization.*** *Run* $\mathrm{Setup}(1^\lambda, n, t)$ *to get* $((\mathsf{sk}_1, \cdots, \mathsf{sk}_n), pp)$*. Give* pp *to* $\mathcal{A}$*.*
- ***Corruption.*** *Receive the set of corrupt parties* C *from* $\mathcal{A}$*, where* $|C| < t$*. Give the secret keys* $\{\mathsf{sk}_i\}_{i \in C}$ *of these parties to* $\mathcal{A}$*. Define the corruption gap as* $g := t - |C|$*.*
- *The adversary makes a mix order of the following two query types:*
  - ***Evaluation queries to parties.*** *In response to* $\mathcal{A}$*'s evaluation query* (Eval, x, i) *for some* $i \in [n] \setminus C$ *return* $\mathrm{Eval}(\mathsf{sk}_i, x, pp)$ *to* $\mathcal{A}$*.*
  - ***Evaluation queries to*** TPRF***.*** *In response to a query* (Eval, x) *choose a set* $S \subseteq [n]$ *of at least* t *and return* $\mathrm{Combine}(\{(i, z_i)\}_{i \in S}; pp)$ *where* $z_i \leftarrow \mathrm{Eval}(sk_i, x, pp)$ *for* $i \in S$*.*
- ***Guess.*** *Finally,* $\mathcal{A}$ *returns a guess* b'*. Output* b'*.*

$\underline{\texttt{IndifGame}(1^\lambda, \textit{Random}) :}$

- ***Initialization.*** *Run* $\mathrm{Setup}(1^\lambda, n, t)$ *to get* $((\mathsf{sk}_1, \cdots, \mathsf{sk}_n), pp)$*. Give* pp *to* $\mathcal{A}$ *and* Sim*.*
- ***Corruption.*** *Receive the set of corrupt parties* C *from* $\mathcal{A}$*, where* $C < t$*. Give the secret keys* $\{\mathsf{sk}_i\}_{i \in C}$ *of these parties to* $\mathcal{A}$ *and* Sim*. Define the corruption gap as* $g := t - |C|$*.*
- *The simulator responds to the adversary as:*
  - ***Evaluation queries to parties.*** *In response to* $\mathcal{A}$*'s evaluation query* (Eval, x, i) *for some* $i \in [n] \setminus C$*,* $\mathsf{Sim}(\{\mathsf{sk}_i\}_{i \in C}; pp)$ *answers either using its databases or with a uniformity random value.*
  - ***Evaluation queries to*** TPRF***.*** *In response to an* $\mathcal{A}$*'s query on* (Eval, x)*,* $\mathsf{Sim}(\{\mathsf{sk}_i\}_{i \in C}; pp)$ *answers either using its databases or with a uniformity random value.*
- ***Guess.*** *Finally,* $\mathcal{A}$ *returns a guess* b'*. Output* b'*.*

Remark. It may sound like a (pathological) example, where Eval and TPRF always output a zero bit, independently of the inputs, rejects our definition. Since this zero-construction is consistent and correct and moreover, a simulator that always outputs 0 bit on any query may work. However, we emphasize that with our convention a valid simulator **has to** return a uniformly random value whenever the output of a query can not be determined from its database. Clearly, for an adversary that makes a single query to TPRF, there is no valid simulator to make these two games indistinguishable for this zero-construction.

We show that the indifferentiability definition (Definition 14) is stronger than the pseudorandomness definition (Definition 9).

First, we show that if a TPRF is pseudorandom with respect to the Definition 9, then it is not necessarily indifferentiable with respect to the Definition 14.

Theorem 1. *There exists a scheme that is secure with respect to the Definition 9 but it is not secure with respect to the Definition 14.*

Proof. Let $\mathtt{PRF} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^m$ be a pseudorandom function where n, m depends on a security parameter λ. Let ComScheme := (Gen, Com, Ver) be a commitment scheme. Let H be a random oracle. We propose a $(2,1)$-TPRF with three algorithms defined below:

1. The Setup(1^λ) algorithm chooses $\mathsf{sk}_1, \mathsf{sk}_2$ uniformly random from $\{0,1\}^k$. Then it executes Gen(1^λ) to obtain a public-key pk_{com}. It chooses a random x from $\{0,1\}^n$ and computes $y_1 = \mathtt{PRF}_{\mathsf{sk}_1}(x)$, $y_2 = \mathtt{PRF}_{\mathsf{sk}_2}(x)$, $r = H(y_1, y_2)$ and $c = \mathrm{Com}_{\mathsf{pk}_{com}}(y_1 \oplus y_2; r)$. Then it sets $pp := (\mathtt{PRF}, H, \mathsf{pk}_{com}, x, c)$. The i-th secret key sk_i is given to party i. That is, $((\mathsf{sk}_1, \cdots, \mathsf{sk}_n), pp) \leftarrow \mathrm{Setup}(1^\lambda, n, t)$.
2. The Eval algorithm generates pseudo-random shares for a given value x and it is defined as $\mathrm{Eval}(\mathsf{sk}_i, x, pp) := \mathtt{PRF}_{\mathsf{sk}_i}(x)$.
3. The Combine algorithm given two partial shares $\{z_1, z_2\}$ from parties returns $z_1 \oplus z_2$.

We sketch why this scheme is secure with respect to the Definition 9 when PRF is pseudorandom and ComScheme satisfies the hiding property. Note that our scheme is a modified NPR TPRF with the parameters $n = 2, t = 1$. Let us assume that the adversary $\mathcal{A}$ is attacking this scheme with respect to the Definition 9. Without of loss of generality, we assume that $\mathcal{A}$ is corrupting the party 1. We discuss the following cases:

1. Neither x is a learning query nor it is the challenge query.
2. x is the challenger query.
3. x is a learning query.

In the case 1 and 2, since $\mathtt{PRF}_{\mathsf{sk}_2}(x)$ is unknown to $\mathcal{A}$ and H is a random oracle, we can use a totally random value r^* instead of r. Then, by the hiding property of the commitment scheme, we can replace c with a random commitment value. The rest of the proof follows from the security of the NPR TPRF.

In the case 3, the proof follows from the security of the NPR TPRF. Because the adversary can learn y_1, y_2, r and checks if these values are consistent with c, but this does not give any extra information.

Insecurity with respect to Definition 11: First, the adversary makes an TPRF query on the input x and forces the simulator to commit to a random answer y'. Later the adversary calculates $r = H(y_1, y_1 \oplus y')$, opens the commitment c to y'', and compares y'' with y'. If y'' equals to y', the adversary guesses an interaction with the real protocol, otherwise, it is interacting with the simulator. The simulator can win only if at least one of the following two cases happens:

1. The simulator breaks the hiding property of the commitment scheme and obtains $y_1 \oplus y_2$ before answering to the TPRF on the input x, or,
2. The simulator breaks the binding property of the commitment scheme and opens the commitment value c to y' with the opening of $H(y_1, y_1 \oplus y')$.

□

Intuitively, if TPRF is indifferentiable with respect to the Definition 14, there exists a simulator Sim that without knowing the non-corrupt keys can simulate (Eval, x, i) queries. Since the adversary does not have enough information to compute (Eval, x^*) where $(x^*, S, \{(i, z_i^*)\}_{i \in U})$ is the challenge query, (that is, the simulator is not able to answer this query using its databases), the simulator Sim (by our convention) has to return a random value for the challenge query. And this shows the psudorandomness property.

Theorem 2. *If a threshold function* TPRF *is indifferentiable with respect to the Definition 14, then it is pseudorandom with respect to the Definition 9.*

Proof. **Game 0**. We start with `PseudoRanGame` game where $\mathcal{A}_P$ is an adversary to attack the psudorandomness definition. We consider a slightly different version (but equivalent) of the Definition 9. That is, a random bit b is chosen by the challenger in the game and $\mathcal{A}_P$ wins if it guesses b correctly. We show that:

$$\Pr[b = b' : b \xleftarrow{\$} \{0,1\}, b' \leftarrow \texttt{PseudoRanGame}_{\mathcal{A}_P}(1^\lambda, b)] \leq 1/2 + \mathsf{neg}(\lambda).$$

Game 1. Let $\mathcal{A}_I$ be an adversary against the Definition 14 and it runs $\mathcal{A}_P$. Below is the description of $\mathcal{A}_I$.

- `Initialization.` The adversary$\mathcal{A}_I$ chooses a random bit b. When receiving pp from its challenger, $\mathcal{A}_I$ passes pp to $\mathcal{A}_P$. After receiving a corrupt set C from $\mathcal{A}_P$, $\mathcal{A}_I$ forwards this set to its challenger. While receiving $\{\mathsf{sk}_i\}_{i \in C}$, it sends this set of keys to $\mathcal{A}_P$.
- `Pre-challenge evaluation queries.` The adversary $\mathcal{A}_I$ answers to an evaluation query $\text{Eval}(\mathsf{sk}_i, x, pp)$ using its challenger and construct a list L. That is, it adds an x to L if $|\{i | \mathcal{A}_P \text{ made a } (\text{Eval}, x, i) \text{ query}\}| \geq g$ where g is the corruption gap.

– `Challenge.` When receiving the challenge query $(x^*, S, \{(i, z_i^*)\}_{i\in U})$ such that $|S| \geq t$ and $U \subseteq S \cap C$, if x^* is in L, it stops and returns 0. Otherwise, it computes $z_i := (\text{Eval}, i, x)$ for $i \in S \cap U$. If there exist an $i \in S \cap U$ such that $z_i \neq z_i^*$, it returns $\perp$. Otherwise, it queries (Eval, x) to its oracle to receive a value z^*. If $b = 0$ it returns z^* to $\mathcal{A}_P$. If $b = 1$, the adversary$\mathcal{A}_I$ returns a random value to $\mathcal{A}_P$.
– `Post-challenge evaluation queries.` The adversary $\mathcal{A}_I$ answers to post-challenge evaluation queries same as the pre-challenge phase except that if $\mathcal{A}_P$ makes a query of the form (Eval, x^*, i) for some $i \in [n] \setminus C$ and i is the g-th party it contacted, then output 0 and stop.
– `Output.` Finally, if the output of $\mathcal{A}_P$ is b', it returns $[b = b']$.

Note that the difference between Game 0 and Game 1 is how the challenge query is responded. Therefore, these two games are indistinguishable by the correctness of the TPRF with respect to the Definition 8.

Game 2. Since TPRF is indifferentiable from a random function, there exists a stateful simulator Sim that without knowing the non-corrupt keys can simulate the answers to the queries by $\mathcal{A}_I$. The simulator chooses random elements to answer the queries whenever the output is not determined from its databases. Game 1 and Game 2 are indistinguishable by the Definition 14.

Since the adversary does not have enough information to evaluate TPRF on x^*, the simulator Sim answers to the (Eval, x^*) query with a random value. Consequently, $\mathcal{A}_P$ will receive a random value in both cases: $b = 0$ and $b = 1$. Therefore, it is clear that in this game the probability that $b' = b$ is at most $1/2$. Putting all together, this finishes the proof. □

3.2 Achievability of Indifferentiability

In this subsection we show that the NPR threshold pseudorandom function satisfies our indifferentiability definition. First we describe a stateful simulator Sim that simulates answers to the adversary's queries using only the corrupt keys. The simulator answers to (Eval, x, i) and (Eval, x) queries with random values unless for the keys that $\mathcal{A}$ knows (the keys that corrupt parties possess) or when the adversary has enough knowledge to compute $\text{NPR}(x)$. The simulator uses `PRF` to answer the queries on the corrupt keys. When the adversary has enough information to compute $\text{NPR}_{\texttt{PRF}}$ itself, the simulator needs to make the consistency between its databases.

Then based on Sim, we introduce some intermediate simulators Sim_ℓ that know ℓ non-corrupt keys beside all the corrupt keys. The difference between Sim and Sim_ℓ is the answers to (Eval, x, i) queries where i is a non-corrupt party and Sim_ℓ knows its corresponding secret key sk_i. For these queries Sim_ℓ uses `PRF`.

Finally, we start with $\texttt{IndifGame}(1^\lambda, \texttt{Real})$ game that uses a $\mathsf{Sim}_{n-|C|}$ that knows all the secret keys. Then, for each non-corrupt party i we remove its corresponding secret key from the simulator (one-by-one) in subsequent games to reach the last game that uses Sim. We show that each two subsequent games

are indistinguishable if $\mathtt{PRF}$ is a pseudorandom function family and this finishes the proof.

Theorem 3. *If* $\mathtt{PRF}$ *is a pseudorandom function family, the* $\mathrm{NPR}_{\mathtt{PRF}}$ *threshold function in Figure 1 is indifferentiable with respect to the Definition 14.*

Proof. First we present some preliminaries and a simulator Sim. Then we use this simulator Sim to prove the theorem.

Simulator $\mathsf{Sim}(\{\mathsf{sk}_i\}_{i\in C}, pp)$:
$\mathtt{Setup}$. For any input query x, let $\mathbf{L_x}$ be an empty list to store pairs (j, α_j). For any $\mathbf{L_x}$, we define $\mathbf{L_{x,J}} := \{j|\ \exists\ (j,\alpha_j) \in \mathbf{L_x}\}$. Let $\mathbf{L_{comb}}$ be an empty list to store pairs (x, y). For any $i \in C$, let $\mathbf{J}_i$ be the set of all indexes of keys in sk_i.

$\mathtt{Evaluation\ queries\ to\ parties}$. In response to a query (Eval, x, i) for an $i \in [n] \setminus C$: The simulator Sim finds all the sets $D_{j_1}, \cdots, D_{j_\ell}$ that contain i. It sets $\mathbf{I_j} := \{j_1, \cdots, j_\ell\}$. Then for any $j \in \mathbf{I_j} \cap \{\mathbf{J}_i\}_{i\in C}$, it computes $h_{i,j} := \mathtt{PRF}(k_j, x)$.

1. **While** $|\mathbf{L_{x,J}}| < \mathbf{g-1}$: For any $j \in \mathbf{I_j}$ that is not in $\{\mathbf{J}_i\}_{i\in C}$, the simulator searches over database $\mathbf{L_x}$ to find a pair (j, α_j). If Sim finds such a pair, it sets $h_{i,j} := \alpha_j$, otherwise, it chooses a random element α_j from the co-domain and sets $h_{i,j} := \alpha_j$ and it stores (j, α_j) in $\mathbf{L_x}$ and it removes j from the set $\mathbf{I_j}$. Then, **if** $|\mathbf{L_{x,J}}| = \mathbf{g-1}$ and $\mathbf{I_j}$ is non-empty, it breaks and goes to the item 2, otherwise, Sim continues with another $j \in \mathbf{I_j}$ that is not in $\{\mathbf{J}_i\}_{i\in C}$.
2. **When** $|\mathbf{L_{x,J}}| = \mathbf{g-1}$: Note that since $|\mathbf{L_{x,J}}| = g-1$ and $\mathbf{I_j}$ is not empty, there is at least one index $j \in \mathbf{I_j}$ that is not in $\mathbf{L_{x,J}}$. For this j, the simulator Sim searches over $\mathbf{L_{comb}}$ to find a pair (x, y) and if it finds such a pair, it sets $h_{i,j} := (\oplus_{m\in\mathbf{L_{x,J}}}\alpha_m) \oplus y \oplus (\oplus_{m\in\{\mathbf{J}_i\}_{i\in C}}\mathtt{PRF}(k_m, x))$. Otherwise, it chooses a random element α_j from the co-domain and sets $h_{i,j} := \alpha_j$ and it stores (j, α_j) in $\mathbf{L_x}$ and then it stores $(x, \oplus_{j\in\mathbf{L_{x,J}}}\alpha_j \oplus_{j\in\{\mathbf{J}_i\}_{i\in C}} \mathtt{PRF}(k_j, x))$ in $\mathbf{L_{comb}}$. Remove this j from $\mathbf{I_j}$. If $\mathbf{I_j}$ is empty, go to the step 4. Otherwise, it continues with the step 3.
3. **When** $|\mathbf{L_{x,J}}| > \mathbf{g-1}$. In this case, the adversary has enough information to compute the evaluation of TPRF on the value x. This means that $\mathtt{PRF}(\cdot, x)$ is evaluated on all the non-corrupt keys. Therefore, for any $j \in \mathbf{I_j}$ that is not in $\{\mathbf{J}_i\}_{i\in C}$, the simulator searches over database $\mathbf{L_x}$ to find a pair (j, α_j) and it sets $h_{i,j} := \alpha_j$.
4. Finally, the simulator sets $z_i := \{h_{i,j_m}\}_{m\in[\ell]}$ and sends z_i to the adversary.

$\mathtt{Evaluation\ queries\ to}$ TPRF. In response to a query (Eval, x), the simulator searches over database $\mathbf{L_{comb}}$ to find a pair (x, y). If it finds such a pair, it sends y to the adversary. Otherwise, it chooses a random element y from the co-domain and sends it to $\mathcal{A}$. Then, it stores (x, y) in $\mathbf{L_{comb}}$.

Fig. 2. Description of the Simulator Sim.

$\mathtt{Initialization}$. For any $i \in [d]$, choose $k_i \xleftarrow{\$} \{0,1\}^k$ where $d = \binom{n}{n-t+1}$. Let $D_1, \cdots, D_d$ be the d distinct $(n-t+1)$-sized subset of $[n]$. For any $i \in [n]$, let $\mathsf{sk}_i := \{k_j | i \in D_j \text{ for } j \in [d]\}$. Set $pp := (\mathtt{PRF}, D_1, \cdots, D_d)$ and gives (sk_i, pp) to

the party i.

`Corruption`. Receive the set of corrupt parties C from $\mathcal{A}$, where $C < t$. Give the secret keys $\{\mathsf{sk}_i\}_{i\in C}$ of these parties to $\mathcal{A}$. Define the corruption gap as $g := t - |C|$.

We describe a simulator Sim that given the set $\{\mathsf{sk}_i\}_{i\in C}$ and pp answers to $\mathcal{A}$'s queries. For any input query x, let $\mathbf{L_x}$ be an empty list to store pairs (j, α_j) where j represents the index of the key k_j and α_j is either $\mathtt{PRF}(k_j, x)$ or a random value. For any $\mathbf{L_x}$, we define $\mathbf{L_{x,J}} := \{j | \exists\ (j, \alpha_j) \in \mathbf{L_x}\}$. Let $\mathbf{L_{comb}}$ be an empty list to store pairs (x, y) where x is an input query and y is the answer to an (Eval, x) query. For any $i \in C$, let $\mathbf{J}_i$ be the set of all indexes of keys in sk_i. The simulator Sim described in Figure 2 answers to (Eval, x, i) and (Eval, x) queries with random values unless for the keys that $\mathcal{A}$ knows (the keys that corrupt parties possess) or when $|\mathbf{L_{x,J}}| \geq \mathbf{g - 1}$. The simulator Sim uses $\mathtt{PRF}$ to answer the queries on the keys in $\{\mathbf{J}_i\}_{i\in C}$. When $|\mathbf{L_{x,J}}| \geq \mathbf{g - 1}$, since the adversary has enough information to compute $\text{NPR}_{\mathtt{PRF}}$ itself, the simulator needs to make the consistency between its databases of $\mathbf{L_x}$ and $\mathbf{L_{comb}}$.

Now we use Sim to show that NPR threshold function is indifferentiable from a random function. Let $\mathbf{J_{NotC}} := \{j_1, \cdots, j_m\}$ be the set of all index of the keys that are not corrupted. That is,

$$\mathbf{K_{NotC}} := \{k_{j_1}, \cdots, k_{j_m}\} = \{k_1, \cdots, k_d\} \setminus \{\mathsf{sk}_i\}_{i\in C}.$$

Game $m + 2$. This is $\mathtt{IndifGame}(1^\lambda, \mathtt{Real})$ in which the adversary $\mathcal{A}$ is interacting with the real protocol.

Game $m + 1$. In this game, we use Sim'_m in the interaction with the adversary. The simulator Sim'_m knows all the keys and responds to any evaluation query (Eval, x, i) using $\mathtt{PRF}$ and keep a database similar to Sim. Note that when the answer can be derived from its databases, the Sim'_m is exactly as Sim. For evaluation queries to TPRF, the simulator Sim'_m is the same as the challenger with a difference that it stores the values in $\mathbf{L_{comb}}$. It is clear that Game $m+2$ and Game $m+1$ are indistinguishable since Sim'_m knows all the secret keys and replies similar to the challenger in the real game.

Game m. In this game, we use Sim_m in the interaction with the adversary. The simulator Sim_m is similar to Sim'_m unless for the evaluation queries to TPRF. For a TPRF evaluation query on an input x, if there exists a pair $(x, \cdot)$ in $\mathbf{L_{comb}}$, the simulator Sim_m uses $\mathbf{L_{comb}}$, otherwise, it returns a random value. We show that two games m and $m + 1$ are indistinguishable. Note that for an evaluation queries to TPRF on an input x, if there exists a pair $(x, \cdot)$ in $\mathbf{L_{comb}}$ derived from (Eval, x, i) queries, the answer to this query is indistinguishable by the consistency property of TPRF. When there is no pair $(x, \cdot)$ in $\mathbf{L_{comb}}$, Sim_m returns a random value y and this will effect the answer for some evaluation queries (Eval, x, i) when $|\mathbf{L_{x,J}}| \geq \mathbf{g - 1}$. Since $\mathtt{PRF}$ is pseudorandom this remains indistinguishable for the adversary. In more details, setting $h_{i,j} := (\oplus_{m\in\mathbf{L_{x,J}}}\alpha_m) \oplus y \oplus (\oplus_{m\in\{\mathbf{J}_i\}_{i\in C}}\mathtt{PRF}(k_m, x))$ when y is

chosen randomly is indistinguishable from receiving $\mathtt{PRF}(k_j, x)$ unless the adversary can find a collision for $\mathtt{PRF}$. That is, it finds two values x_1, x_2 such that $\mathtt{PRF}(k_j, x_1) = \mathtt{PRF}(k_j, x_2)$. Since a pseudorandom function is indistinguishable from a random function and a random function is collision-resistant, these two games are indistinguishable.

Game $m-1$. In this game, we use Sim_{m-1} in the interaction with the adversary. The simulator Sim_{m-1} is similar to Sim_m unless for evaluation queries that needs k_{j_m}. For these queries, Sim_{m-1} answers similar to Sim (returns a random value whenever it is needed.). We show that Game m and Game $m-1$ are indistinguishable since $\mathtt{PRF}$ is a pseudorandom function family. Let's assume that the adversary $\mathcal{A}$ distinguishes these two games with a probability ϵ. A reduction adversary $\mathcal{B}^O$ runs $\mathcal{A}$ and answers to $\mathcal{A}$'s queries similar to Sim_m unless for queries that involve the key k_{j_m}. For any evaluation query (Eval, x, i) and any combine query (Eval, x) that needs k_{j_m}, $\mathcal{B}$ uses its oracle O to reply. It is clear that when O is a pseudorandom function, $\mathcal{B}$ simulates the Game m for $\mathcal{A}$. And when O is a truly random function, $\mathcal{B}$ simulates the Game $m-1$ for $\mathcal{A}$. Therefore, the adversary $\mathcal{B}^O$ is able to distinguishes $\mathtt{PRF}_{k_{j_m}}$ from a truly random function with the probability ϵ. We can conclude that ϵ should be negligible since $\mathtt{PRF}_{k_{j_m}}$ is a pseudorandom function.

We define **Games** $m-2$, **Games** $m-3$, $\cdots$, **Games** 1, **Games** 0 which use Sim_{m-2}, $\mathsf{Sim}_{m-3}, \cdots, \mathsf{Sim}_1, \mathsf{Sim}$, respectively. Similarly, we can show that each two subsequent games are indistinguishable since $\mathtt{PRF}$ is a pseudorandom function. This finishes the proof since $m < d$ is a polynomial on the security parameter. □

4 Threshold Message Authentication Code

We define an *existential unforgeability* definition for a randomized threshold message authentication code. Currently, we are not aware of a threshold message authentication code that is randomized, and we present our definition for completeness and for the prospect of inventing such a construction in the future. (Note that randomized message authentication codes are well-motivated and constructed [JJV02, DKPW12].)

4.1 Our Definition

Definition 15 (Strong Existential Unforgeability). *We say a threshold message authentication code* TMAC := (TM. Setup, TM. Mac, TM. Verif) *satisfies* existential unforgeability *if for all* PPT *adversaries* $\mathcal{A}$*, there exists a negligible function* neg *such that*

$$Pr[\mathtt{Forge}_{\mathrm{TM.Mac},\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{neg}(\lambda),$$

where Forge *is defined below.*

$\underline{\mathtt{Forge}_{\mathrm{TMAC},\mathcal{A}}(1^\lambda)}$*:*

- ***Initialization.*** *Run* $\mathrm{TM.Setup}(1^\lambda, n, t)$ *to get* $((\mathsf{sk}_1, \cdots, \mathsf{sk}_n, \mathsf{sk}_R), pp)$*. Give* pp *to* $\mathcal{A}$*. Sets a counter* $\mathbf{ct} := 0$ *and a list* $L := \emptyset$*.*
- ***Corruption.*** *Receive the set of corrupt parties* C *from* $\mathcal{A}$*, where* $|C| < t$*. Give the secret keys* $\{\mathsf{sk}_i\}_{i \in C}$ *of these parties to* $\mathcal{A}$*. Define the gap between the threshold and the number of corrupt parties as* $g := t - |C|$*.*
- ***MAC queries.*** *In response to* $\mathcal{A}$*'s MAC query* (Mac, j, m, S) *where* $j \in S$ *and* $|S| \geq t$*, run the protocol* $\mathrm{TM.Mac}$ *with* m, S *as the inputs of* j *to get tag* tag*. If* $j \in C$*, increment* $\mathbf{ct}$ *by* $|S \setminus C|$ *(number of honest parties in* S*). Otherwise, send* (m, tag) *to* $\mathcal{A}$ *and append* (m, tag) *to* L*.*
- ***Forgery.*** *Let* $k := \lfloor \mathbf{ct}/g \rfloor$*. The adversary* $\mathcal{A}$ *outputs* (m_1, tag_1)*,* (m_2, tag_2)*,* $\cdots$*,* $(m_{k+1}, \mathsf{tag}_{k+1})$ *such that* $(m_i, \mathsf{tag}_i) \notin L$ *for every* $i \in [k+1]$ *and* $(m_i, \mathsf{tag}_i) \neq (m_j, \mathsf{tag}_j)$ *for any* $i \neq j$ *. For every* $i \in [k+1]$*, run* $\mathrm{TM.Verif}$ *on the input* (m_i, tag_i) *and a set* $|S| \geq t$ *and* $S \cap C = \emptyset$*. It output* 1 *if* $\mathrm{TM.Verif}$ *returns* 1 *for all. We assume that parties involved in the verification behaves honestly.*

Remark. For a deterministic threshold message authentication code, the definition above can be simplified since for any $\mathcal{A}$'s MAC query (Mac, j, m, S) where $j \in S$ and $|S| \geq t$, the corresponding tag can be computed and stored in L by the oracle even when $j \in C$. Therefore, in the forgery phase, the adversary outputs one pair (m, tag) and wins if $(m, \mathsf{tag}) \notin L$ and $\mathrm{TM.Verif}$ on the input (m, tag) return 1.

4.2 Achievability

Figure 3 presents a threshold message authentication code based on a threshold pseudorandom function. We show that the protocol in Figure 3 satisfies *strong existential unforgeability* with respect to Definition 15 if the underlying threshold pseudorandom function satisfies the *pseudorandomness* with respect to Definition 9. The high-level idea of the proof: from an adversary $\mathcal{A}$ that attacks the unforgeability, we construct an adversary $\mathcal{B}$ that breaks the pseudorandomness of the underlying TPRF. The reduction adversary $\mathcal{B}$ is able to track the messages that the adversary $\mathcal{A}$ might know their corresponding tags. Therefore, when $\mathcal{A}$ returns $k+1$ pairs (m_1, tag_1), (m_2, tag_2), $\cdots$, $(m_{k+1}, \mathsf{tag}_{k+1})$ at the end, $\mathcal{B}$ is able to detect at least one pair (m_i, tag_i) in which tag_i is not obtained from MAC queries since the message authentication protocol in Figure 3 is deterministic. Then, the reduction adversary $\mathcal{B}$ uses this pair (m_i, tag_i) to break the pseudorandomness of TPRF.

Theorem 4. *If* TPRF *satisfies the pseudorandomness and correctness, then the protocol in Figure 3 satisfies the strong existential unforgeability.*

Ingredients:

- An $(n; t)$-threshold pseudorandom function TPRF := (Setup, Eval, Combine)

TM. $\mathrm{Setup}(1^\lambda, n, t) \to (\mathsf{sk}_1, \cdots, \mathsf{sk}_n, \mathsf{sk}_R; pp)$. Run $\mathrm{Setup}(1^\lambda, n, t)$ to get the keys $(\mathsf{sk}_1, \cdots, \mathsf{sk}_n; pp)$. Then it sets $\mathsf{sk}_R := \{\mathsf{sk}_i\}_{i \in S}$ for a set $S \subseteq [n]$ of size at least t.

TM. $\mathrm{Mac}([\![\mathsf{sk}]\!]_{[n]}; [j : m; S]; pp) \to [j : t/\ \perp]$: To obtain a tag on a message m with the help of parties in S:

- Party j sends m to all parties in S.
- For every $i \in S$, party i runs $\mathrm{Eval}(sk_i, m; pp)$ to get tag_i, and sends it to party j.
- Party j runs Combine on $\{\mathsf{tag}_i\}_{i \in S}$ to get a value tag.

TM. $\mathrm{Verif}([\![\mathsf{sk}]\!]_{[n]}; [j : (m, \mathsf{tag}); S]; pp) \to [j : b]$:

- Party j sends m to all parties in S.
- For every $i \in S$, party i runs $\mathrm{Eval}(sk_i, m; pp)$ to get tag_i, and sends it to party j.
- Party j runs Combine on $\{\mathsf{tag}_i\}_{i \in S}$ to get a value tag'.
- Party j returns $[\mathsf{tag} = \mathsf{tag}']$.

Fig. 3. Threshold Message Authentication Code

Proof. Let $\mathcal{A}$ be a polynomial-time adversary that breaks the simplified *strong existential unforgeability*. (Since the message authentication code is deterministic.) Let $\mathcal{B}$ be a polynomial-time adversary that has oracle access to a TPRF. The adversary $\mathcal{B}$ runs $\mathcal{A}$ and answers to its queries as follows.

Description of $\mathcal{B}$: It sets a list $L := \emptyset$.

- `Corruption`. Receive the set of corrupt parties C from $\mathcal{A}$, where $|C| < t$. Forward the set C to its oracle to receive the secret keys $\{\mathsf{sk}_i\}_{i \in C}$ of these parties. Send $\{\mathsf{sk}_i\}_{i \in C}$ to $\mathcal{A}$. Define the gap between the threshold and the number of corrupt parties as $g := t - |C|$.
- `MAC queries.` When $\mathcal{A}$ makes a MAC query (Mac, j, m, S) where $j \in S$ and $|S| \geq t$, if $j \in C$, whenever it receives a query TM. $\mathrm{Mac}(\mathsf{sk}_i, m, pp)$ for an $i \in S \setminus C$, it sends (Eval, m, i) to its oracle to get a response tag_i. Then it forwards tag_i to the party j. Also, for any $i \in S \cap C$, it computes $\mathsf{tag}_i := \mathrm{Eval}(sk_i, m; pp)$. Finally, it computes $\mathrm{Combine}(\{(i, \mathsf{tag}_i)\}_{i \in S}; pp)$ to get a value tag and appends (m, tag) to L.
  When $j \notin C$, for any $i \in S \setminus C$ the adversary $\mathcal{B}$ queries (Eval, m, i) to its oracle to receive tag_i. Also, for any $i \in S \cap C$, it computes $\mathsf{tag}_i := \mathrm{Eval}(sk_i, m; pp)$. Finally, it computes $\mathrm{Combine}(\{(i, \mathsf{tag}_i)\}_{i \in S}; pp)$ to get a value tag. It sends (m, tag) to $\mathcal{A}$ and appends (m, tag) to L.
- `Attack.` When the adversary $\mathcal{A}$ outputs a pair $(m, \mathsf{tag}) \notin L$ as a forgery, the adversary $\mathcal{B}$ sends $(m, S, \emptyset)$ as a challenge query to its oracle to receive a value tag^*. If $\mathsf{tag} = \mathsf{tag}^*$, $\mathcal{B}$ returns 0. Otherwise, it returns 1.

Success probability. Note that if the advantage of $\mathcal{A}$ in outputting a forgery is ϵ, the adversary $\mathcal{B}$ is able to break the *pseudorandomness* with the probability negligibly close to $|\epsilon - 1/2^T|$ where T is the set of all possible tags. For the reason that when $b = 1$, $\mathcal{B}$'s oracle returns a random value tag^* and therefore $\mathsf{tag} = \mathsf{tag}^*$ with the probability $1/2^T$. And when $b = 0$, $\mathsf{tag} = \mathsf{tag}^*$ with the probability negligibly close to ϵ by the correctness of TPRF. □

5 Threshold Symmetric Encryption

5.1 Our definition

The `indirect decryption queries` in the *message privacy* definition (Definition 12) are allowed to be initiated only by a non-corrupt party and their outputs are not communicated to the adversary. We argue that this is not a real-world attack scenario since an adversary $\mathcal{A}$ that corrupts some parties C can control their behaviors. This means that $\mathcal{A}$ can initiate a decryption query with the help of a corrupt party in C.[5] But these decryption queries are not embedded in Definition 12.

We strengthen the *message privacy* definition on two fronts:

1. A corrupt party can initiate a decryption query.
2. The output of a decryption query initiated by a non-corrupt party will be sent to $\mathcal{A}$.

Obviously, non-corrupt parties should not participate in the decryption of the challenge ciphertext c^*, otherwise, no encryption scheme satisfies the security definition.

Definition 16 (IND-CCA2 Security). *A* TSE := (TSE. Setup, TSE. Enc, TSE. Dec) *satisfies* IND-CCA2 security *if for all* PPT *adversaries* $\mathcal{A}$*, there exists a negligible function* neg *such that*

$$\Pr[\mathtt{CCAGame2}_{\mathrm{TSE},\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathtt{CCAGame2}_{\mathrm{TSE},\mathcal{A}}(1^\lambda, 1) = 1] \leq \mathsf{neg}(\lambda),$$

where $\mathtt{CCAGame2}_{\mathrm{TSE},\mathcal{A}}$ *is defined below.*

$\mathtt{CCAGame2}_{\mathrm{TSE},\mathcal{A}}(1^\lambda, b)$:

- ***Initialization.*** *Run* TSE. Setup$(1^\lambda, n, t)$ *to get* $((\mathsf{sk}_1, \cdots, \mathsf{sk}_n), pp)$*. Give* pp *to* $\mathcal{A}$*.*
- ***Corruption.*** *Receive the set of corrupt parties* C *from* $\mathcal{A}$*, where* $|C| < t$*. Give the secret keys* $\{\mathsf{sk}_i\}_{i \in C}$ *of these parties to* $\mathcal{A}$*.*
- ***Pre-challenge encryption queries.*** *In response to* $\mathcal{A}$*'s encryption query* (Encrypt, j, m, S) *where* $j \in S$ *and* $|S| \geq t$*, run an instance of the protocol* TSE. Enc *with* $\mathcal{A}$*. Note that when* $j \notin C$*, party* j *runs* TSE. Enc *and returns the output ciphertext to* $\mathcal{A}$*. Repeat this step as many times as* $\mathcal{A}$ *desires.*

[5] This trivially breaks Definition 12 since a decryption query initiated by a corrupt party on the input c^* will reveal the bit b to $\mathcal{A}$.

- *`Pre-challenge decryption queries`. In response to $\mathcal{A}$'s decryption query* (Decrypt, j, c, S) *where $j \in S$ and $|S| \geq t$, party j initiates* TSE.Dec *with inputs c and S and it sends the resulting output to $\mathcal{A}$. Repeat this step as many times as $\mathcal{A}$ desires.*
- *`Challenge`. A outputs $(j^*; m_0; m_1; S^*)$ where $|m_0| = |m_1|$, $j^* \in S^* \setminus C$ and and $|S^*| \geq t$. Party j^* initiates* TSE.Enc *with inputs m_b and S^* and sends the resulting ciphertext c^* to $\mathcal{A}$.*
- *`Post-challenge encryption queries`. Same as the pre-challenge phase.*
- *`Post-challenge decryption queries`. Same as the pre-challenge decryption queries except for the challenge ciphertext c^*. For this ciphertext, non-corrupt parties will not participate in the decryption.*
- *`Guess`. Finally, $\mathcal{A}$ returns a guess b'. Output b'.*

IND-CCA1 Security. *It is similar to IND-CCA2 except that $\mathcal{A}$ is not allowed to make post-challenge decryption queries in the IND-CCA1 game.*

5.2 Existing Schemes

The threshold symmetric encryption in [NPR99]. In [NPR99], authors propose a threshold symmetric encryption scheme constructed from an IND-CPA symmetric encryption scheme Π and a distributed pseudorandom function F. Vaguely speaking, any party j to encrypt a message m, generates a secret key sk for the symmetric encryption scheme, then it encrypts the message m with this key sk to get a value e ($e \leftarrow \mathrm{Enc}_{\Pi}(\mathsf{sk}, m)$). Then it invokes a distributed evaluation of F on the input $j \| e$ (with a set of parties beyond the threshold) to obtain a value y. Finally, it returns the ciphertext $c := (j, e, y \oplus \mathsf{sk})$. To decrypt, it is clear that any set of parties that has at least t members can recompute y from F and $j \| e$ and consequently can obtain sk and decrypt e with $\mathrm{Dec}_{\Pi}(\mathsf{sk}, \cdot)$ to get m.

The threshold symmetric encryption in [AMMR18]. The protocol in [AMMR18] uses a commitment scheme, a threshold pseudorandom function and a pseudorandom generator. Vaguely speaking, any party j to encrypt a message m, commits to m using a fresh randomness ρ to get a value α, that is, $\alpha := \mathrm{Com}(m, pp; \rho)$. Then it invokes a distributed evaluation of F on the input $j \| \alpha$ (with a set of parties beyond the threshold) to obtain a value y. Finally, it returns the ciphertext $c := (j, \alpha, \mathtt{PRG}(y) \oplus m \| \rho)$. To decrypt, it is clear that any set of parties that has at least t members can recompute y from F and $j \| \alpha$ and consequently can obtain $m \| \rho$ and check if α is equal to $\mathrm{Com}(m, pp; \rho)$ or not.

The threshold symmetric encryption in [DMV22]. The DiAE protocol in [DMV22] uses a threshold pseudorandom function (F), a Key Derivation Function (KDF) [Kra10] and a Encryptment scheme (EC) [DGRW18]. Vaguely speaking, any party j to encrypt a message m, runs the key generating algorithm of EC to get a key k_{EC} and then it computes $(c_{EC}, t_{EC}) \leftarrow EC(k, j, m)$. Next it invokes a distributed evaluation of F on the input $j \| t_{EC}$ (with a set of parties beyond the threshold) to obtain a value y. Finally, it returns the ciphertext

$c := (j, c_{EC}, t_{EC}, e)$ where $e := KDF(y) \oplus k_{EC}$. To decrypt, it is clear that any set of parties that has at least t members can recompute y from F and $j \| t_{EC}$ and consequently can obtain k_{EC}. Therefore, it can run the decryption algorithm of EC on the inputs k_{EC}, j, t_{EC} and c_{EC} to get the message m back.

CCA Attack. We show that these encryption schemes are not IND-CCA2 secure with respect to Definition 16. An adversary given the challenge ciphertext $c^* := (j^*, e^*, F(j^* \| e^*) \oplus \mathsf{sk})$ (or $c^* := (j^*, \alpha^*, \mathtt{PRG}(F(j^* \| \alpha^*)) \oplus m_b \| \rho^*)$) can initiate a decryption query on an input (j^*, e^*, β) where $\beta \neq F(j^* \| e^*) \oplus \mathsf{sk}$ (or (j^*, α^*, β) where $\beta \neq \mathtt{PRG}(F(j^* \| \alpha^*)) \oplus m_b \| \rho^*)$) with the help of a corrupt party j_c. (Note that the honest parties would participate in the decryption since c^* is not submitted as a decryption query.) These means that j_c is able to evaluate $F(j^* \| e^*)$ (or $F(j^* \| \alpha^*)$) and decrypt the challenge ciphertext. Similarly, we can show that the DiAE protocol in [DMV22] is not CCA secure.

5.3 Encrypt-then-Mac

A known approach to construct an IND-CCA2 secure symmetric encryption scheme is the Encrypt-then-Mac method [BN08]. We discuss the difficulties of constructing an IND-CCA2 secure threshold symmetric encryption using this method below.

Discussion: why Encrypt-then-Mac may not work. As shown in [BN08], any encryption scheme that is both IND-CPA secure and INT-CTXT secure is also IND-CCA2 secure. Vaguely speaking, an encryption scheme is INT-CTXT secure if the only way to generate a valid cipheretxt is to query the encryption algorithm. (We say a ciphertext c is valid if it decrypts to a massage and not to $\perp$.) The high-level idea of the proof is as follows. Since the only way to generate a valid ciphertext is to use the encryption oracle, then a CPA reduction adversary $\mathcal{B}$ can answer to decryption queries made by a CCA adversary $\mathcal{A}$ if $\mathcal{B}$ stores all the encryption queries made by $\mathcal{A}$. However, in the distributed setting, a CCA adversary $\mathcal{A}$ may be able to generate a valid ciphertext through an decryption query that is initiated by a corrupt party. And this may prevent the reduction strategy to work.

Later, the authors in [BN08] show that the Encrypt-then-Mac approach will result in an encryption scheme that is IND-CPA secure and INT-CTXT secure if the underlying encryption scheme is IND-CPA secure and the underlying message authentication code is strongly unforgeable. And this gives us an IND-CCA2 secure encryption. We argue that the Encrypt-then-Mac approach may not work in the distributed setting. To do so, we take a threshold symmetric encryption scheme TSE$'$ that is malleable but it satisfies the *message privacy* (or it satisfies IND-CPA security notion in the threshold setting). Now for any threshold message authentication code TM. Mac with a distributed verification algorithm, we define a threshold symmetric encryption scheme Π_{TSE} such that its encryption algorithm on an input m returns $c = (c_1, c_2)$ where $c_1 := \mathrm{Enc}_{\mathrm{TSE}'}(m)$ and

$c_2 := \mathrm{TM.Mac}(\mathrm{Enc}_{\mathrm{TSE}'}(m))$. The adversary can alter c_1 to get a valid ciphertext c_1' and initiates a decryption query with the help of a corrupt party j on (c_1', c_2'). The decryption algorithm first checks if c_2' is a correct tag for c_1' or it is not. During this verification, the party j may learn about a correct tag tag on c_1'. That is, the party j may learn $\mathsf{tag} = \mathrm{TM.Mac}(c_1')$. Consequently, a valid ciphertext (c_1', tag) is generated without using the encryption oracle. This would be problematic if the adversary alters the challenge ciphertext which would help the adversary to distinguish which message has been encrypted. Therefore, it breaks the IND-CCA2 security.

For instance, we can use the Encrypt-then-Mac method and a threshold pseudorandom function TPRF to construct a threshold symmetric encryption scheme. To achieve this, any party j to encrypt a message m, generates a secret key sk for the symmetric encryption scheme, then it encrypts the message m with this key sk to get a value e ($e \leftarrow \mathrm{Enc}_{\Pi}(\mathsf{sk}, m)$). It invokes a distributed evaluation of TPRF on the input $j\|e$ (with a set of parties beyond the threshold) to obtain a value y. Then, it invokes a distributed evaluation of TPRF on the input $j\|e\|y \oplus \mathsf{sk}$ (with a set of parties beyond the threshold) to obtain a value tag. Finally, it returns the ciphertext $c := (j, e, y \oplus \mathsf{sk}, \mathsf{tag})$. To decrypt, any set of parties that has at least t members can recompute a value tag' from TPRF and $j\|e\|y \oplus \mathsf{sk}$. If $\mathsf{tag}' \neq \mathsf{tag}$, honest parties would not participate in the rest of decryption. If $\mathsf{tag}' = \mathsf{tag}$, they can recompute the value y from TPRF and $j\|e$ and consequently can obtain sk and decrypt e with $\mathrm{Dec}_{\Pi}(\mathsf{sk}, \cdot)$ to get m.

Regrettably, this threshold symmetric encryption scheme is not IND-CCA2 secure. Let $c^* = (c_1^*, c_2^*)$ where $c_1^* = (j^*, e^*, y^* \oplus \mathsf{sk}^*)$ be the challenge ciphertext. If the adversary initiates a decryption query $c = (c_1, c_2)$ where $c_1 = (j^*, e^*, \beta)$, $\beta \neq y^* \oplus \mathsf{sk}^*$ and $c_2 \neq c_2^*$ with the help of a corrupt party, the adversary can learn the evaluation of $\mathrm{TPRF}(c_1)$. Then, it initiates a decryption query $(c_1, \mathrm{TPRF}(c_1))$ with the help of a corrupt party to learn $y^* = \mathrm{TPRF}(j^*\|e^*)$. Finally, the adversary using y^* can obtain the challenge message m_b and breaks the IND-CCA2 security.

5.4 Encrypt-then-Sign

We show that the Encrypt-then-Sign approach will result in an IND-CCA2 secure threshold symmetric encryption scheme.

Theorem 5. *If* TPRF *is an (n, t)-threshold function satisfies Definition 9,* SYM *is a one-time IND-CPA secure encryption scheme and* SignScheme *satisfies the strong unforgeability, the threshold symmetric encryption in Figure 4 is IND-CCA2 secure.*

Proof. **Game 0**. We start with $\mathtt{CCAGame2}(1^\lambda, b)$ game where $\mathcal{A}_{cca}$ is an adversary to attack the CCA definition. We consider a slightly different version (but equivalent) of Definition 16. Namely, a random bit b is chosen by the challenger in the security game and the value m_b is encrypted. Then, the adversary $\mathcal{A}_{cca}$ wins if it guesses b correctly. We show that:

$$\Pr[b = b' : b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathtt{CCAGame2}_{\mathcal{A}_{cca}}(1^\lambda, b)] \leq 1/2 + \mathsf{neg}(\lambda).$$

Ingredients:

- An $(n; t)$-threshold pseudorandom function TPRF := (Setup, Eval, Combine)
- A symmetric encryption scheme SYM := (Sym. Gen, Sym. Enc, Sym. Dec)
- A signature scheme SignScheme := (SGen, Sign, Verif).

$\underline{\text{Setup}(1^\lambda, n, t) \to (\mathsf{sk}_1, \cdots, \mathsf{sk}_n; pp).}$ Run $\text{Setup}(1^\lambda, n, t)$ to get the keys $(\mathsf{sk}_1^f, \cdots, \mathsf{sk}_n^f; pp^f)$. Run $\text{SGen}(1^\lambda)$ n times to get n key pairs $((\mathsf{sk}_1^s, \mathsf{pk}_1^s) \cdots (\mathsf{sk}_n^s, \mathsf{pk}_n^s)$. For any $i \in [n]$, set $\mathsf{sk}_1 := (\mathsf{sk}_i^f, \mathsf{sk}_i^s)$. Set $pp := (pp^f, \mathsf{pk}_1^s, \cdots, \mathsf{pk}_n^s)$.

$\underline{\text{TSE. Enc}([\![\mathsf{sk}]\!]_{[n]}; [j : m; S]; pp) \to [j : c/\ \perp] :}$ To encrypt a message m with the help of parties in S:

- Party j samples $k \leftarrow$ Sym. Gen and computes $e \leftarrow \text{Sym. Enc}_k(m)$. Then it sends e to all parties in S.
- For every $i \in S$, party i runs $\text{Eval}(\mathsf{sk}_i^f, j \| e; pp^f)$ to get y_i, and sends it to party j.
- Party j runs $\text{Combine}(\{(i, y_i)\}_{i \in S}; pp^f)$ to get y or $\perp$. If $\perp$, it stops and returns $\perp$. Otherwise, it sets $c_1 := (j, e, y \oplus k)$ and computes $c_2 = \text{Sign}(\mathsf{sk}_j^s, c_1)$ and outputs $c := (c_1, c_2)$.

$\underline{\text{TSE. Dec}([\![\mathsf{sk}]\!]_{[n]}; [j : c; S]; pp) \to [j : m/\ \perp] :}$ To decrypt a ciphertext c with the help of parties in S:

- Party j checks if c_1 is of the form (j', e, β) for some $j' \in [n]$. If not, it stops and returns $\perp$. Then it computes $b = \text{Verif}(\mathsf{pk}_{j'}^s, c_1, c_2)$. If $b = 0$, it stops and returns $\perp$. Otherwise, it sends $c = (c_1, c_2)$ to all parties in S.
- For every $i \in S$, party i computes $b = \text{Verif}(\mathsf{pk}_{j'}^s, c_1, c_2)$. If $b = 0$, it stops and returns $\perp$. Otherwise, it runs $\text{Eval}(sk_i^f, j' \| e; pp^f)$ to get y_i, and sends it to the party j.
- Party j runs $\text{Combine}(\{(i, y_i)\}_{i \in S}; pp^f)$ to get y or $\perp$. If $\perp$, it aborts and returns $\perp$. Otherwise, it computes $k := \beta \oplus y$ and outputs $m := \text{Sym. Dec}_k(e)$.

Fig. 4. Threshold Symmetric Encryption: Encrypt-then-Sign

Let us assume $(c_1^* = (j^*, e^*, \beta^*), c_2^*)$ is the challenge ciphertext.

Game 1. This is similar to the Game 0 except for the post-challenge decryption queries. The decryption oracle returns $\perp$ if the adversary $\mathcal{A}_{cca}$ submits a decryption query (c_1, c_2) where $c_1 = (j^*, e^*, \cdot)$. We define a bad event $\mathcal{E}$ in which $\mathcal{A}_{cca}$ submits a decryption query $(c_1 = (j^*, e^*, \beta), c_2)$ such that $(c_1, c_2) \neq (c_1*, c_2^*)$ and $\text{Verif}(\mathsf{pk}_{j^*}^s, c_1, c_2) = 1$. Note that two Games $0, 1$ are distinguishable if the bad event $\mathcal{E}$ happens with a non-negligible probability. If $\mathcal{A}_{cca}$ is able to do so, we can easily construct a reduction adversary that breaks the strong unforgeability of the SignScheme.

Let us assume that $\mathcal{A}_{cca}$ submits a decryption query $(c_1 = (j^*, e^*, \beta), c_2)$ such that $(c_1, c_2) \neq (c_1*, c_2^*)$ and $\text{Verif}(\mathsf{pk}_{j^*}^s, c_1, c_2) = 1$ with a non-negligible probability (or the bad event $\mathcal{E}$ happens with a non-negligible probability). A reduction adversary $\mathcal{B}$ that has an oracle access to a signing algorithm $\text{Sign}_{\mathsf{sk}^s}$ with its corresponding public-key pk^s, chooses a random index j^* from $[n]$, a threshold pseudorandom function TPRF := (Setup, Eval, Combine), a symmetric encryption scheme SYM := (Sym. Gen, Sym. Enc, Sym. Dec). Then, it runs $\text{Setup}(1^\lambda, n, t)$ to get the keys $(\mathsf{sk}_1^f, \cdots, \mathsf{sk}_n^f; pp^f)$. It runs the setup algorithm $\text{SGen}(1^\lambda)$ of its signing oracle n times to get n key pairs $((\mathsf{sk}_{j_1}^s, \mathsf{pk}_{j_1}^s) \cdots (\mathsf{sk}_{j_n}^s, \mathsf{pk}_{j_n}^s)$. It sets $pp := (pp^f, \mathsf{pk}_1^s, \cdots, \mathsf{pk}_n^s)$ where $\mathsf{pk}_i^s := \mathsf{pk}_{j_i}^s$ when $i \neq j^*$, and $\mathsf{pk}_{j^*}^s := \mathsf{pk}^s$.

Then it runs $\mathcal{A}_{cca}$ with pp. When receiving the set of corrupt parties, if j^* is corrupted, $\mathcal{B}$ aborts. Otherwise, for any $i \in [n]$ where $i \neq j^*$, it sets $\mathsf{sk}_i := (\mathsf{sk}_i^f, \mathsf{sk}_i^s)$. And for j^*, it uses its signing oracle $\text{Sign}_{\mathsf{sk}^s}$. It sends the set of corrupt keys to the adversary. The adversary $\mathcal{B}$ simulates the $\mathcal{A}$'s encryption and decryption queries using its keys and its signing oracle $\text{Sign}_{\mathsf{sk}^s}$. Note that with a probability $1 - \frac{|C|}{n}$, $\mathcal{B}$ does not abort. Condition on not aborting, the adversary $\mathcal{A}_{cca}$ makes a challenge query

$$\text{TSE. Enc}([\![\mathsf{sk}]\!]_{[n]}; [j^* : m_0, m_1; S^*]; pp)$$

with a probability $1/n$. Overall, this is a non-negligible probability since the number of parties is a polynomial in the security parameter λ. The reduction adversary $\mathcal{B}$ chooses a random b, encrypts m_b and sends the resulting ciphertext to $\mathcal{A}_{cca}$.

When $\mathcal{A}_{cca}$ submits a post-challenge decryption query $(c_1 := (j^*, e^*, \beta), c_2)$ such that $(c_1, c_2) \neq (c_1*, c_2^*)$ and $\text{Verif}(\mathsf{pk}_{j^*}^s, c_1, c_2)$ returns 1, $\mathcal{B}$ return (c_1, c_2) as a new forgery. Note that the only way to access the signing algorithm $\text{Sign}_{\mathsf{sk}^s}$ is through an encryption query that is initiated by the party j^*. And in each execution of the encryption query, the party j^* generates a new value e randomly. Therefore, a valid signature on $c_1 = (j^*, e^*, \beta)$ would be a valid forgery with a high probability (the probability is not 1 since e^* may be generated twice with some negligible probability.). Overall, the reduction adversary is able to break the strong unforgeability of the underlying signature scheme if the bad event $\mathcal{E}$ happens with a non-negligible probability.

Game 2. This is similar to Game 1 unless, how the challenger responds to the challenge query $\mathrm{TSE.Enc}([\![\mathsf{sk}]\!]_{[n]}; [j^* : m_0, m_1; S^*]; pp)$. The challenger computes e^* and y^* similar to the Game 1, but it sets $\beta^* := \beta^{\$}$ for a randomly chosen $\beta^{\$}$ afterwards. Finally, it returns $c_1^* := (j^*, e^*, \beta^{\$})$ and $c_2^* := \mathrm{Sign}_{\mathsf{sk}^s_{j^*}}(c_1^*)$ to the adversary. We show that Game 1 and Game 2 are indistinguishable since the underlying threshold function TPRF satisfies Definition 9.

Let us assume that an adversary $\mathcal{A}_{dis}$ distinguishes these two games. We construct a reduction adversary $\mathcal{B}_{tprf}$ which breaks Definition 9. Here is the description of the reduction adversary $\mathcal{B}_{tprf}$:

- The reduction adversary $\mathcal{B}_{tprf}$ chooses a signature scheme SignScheme := (SGen, Sign, Verif) and a symmetric encryption scheme SYM := (Sym. Gen, Sym. Enc, Sym. Dec). It runs $\mathrm{SGen}(1^\lambda)$, n times to get n key pairs $(\mathsf{sk}_1^s, \mathsf{pk}_1^s)$, $\cdots$, $(\mathsf{sk}_n^s, \mathsf{pk}_n^s)$. When it receives pp^f from its challenger, it sets $pp := (pp^f, \mathsf{pk}_1^s, \cdots, \mathsf{pk}_n^s)$ and runs $\mathcal{A}_{dis}$ with the input pp.
- When $\mathcal{A}_{dis}$ sends a set C of corrupt parties, $\mathcal{B}_{tprf}$ forwards C to its challenger to receive $\{\mathsf{sk}_i^f\}_{i\in C}$. Then it sends $\{(\mathsf{sk}_i^f, \mathsf{sk}_i^s)\}_{i\in C}$ to $\mathcal{A}_{dis}$.
- The adversary $\mathcal{B}_{tprf}$ simulates the encryption and the decryption queries using SYM, its oracle and SignScheme.
- When $\mathcal{A}_{dis}$ outputs a challenge query $\mathrm{TSE.Enc}([\![\mathsf{sk}]\!]_{[n]}; [j^* : m_0, m_1; S^*]; pp)$, the adversary $\mathcal{B}_{tprf}$ chooses a random value b, samples a key $k^* \leftarrow \mathrm{Sym.Gen}(1^\lambda)$, computes $e^* \leftarrow \mathrm{Sym.Enc}_{k^*}(m_b)$ and sends $((j^*, e^*), S^*, \emptyset)$ to its challenger. After receiving an answer y^*, it sets $c_1^* := (j^*, e^*, y^* \oplus k^*)$ and $c_2^* := \mathrm{Sign}_{\mathsf{sk}^s_{j^*}}(c_1^*)$. Finally, it sends $c^* := (c_1^*, c_2^*)$ to $\mathcal{A}_{dis}$.
- The adversary $\mathcal{B}_{tprf}$ simulates the encryption and the decryption queries using SYM, its oracle and SignScheme unless for decryption queries of the form $c_1 = (j^*, e^*, \cdot)$. For these queries, $\mathcal{B}_{tprf}$ returns $\perp$ immediately.
- When $\mathcal{A}_{dis}$ returns a guess b', $\mathcal{B}_{tprf}$ outputs b'.

It is clear that the adversary $\mathcal{B}_{tprf}$ simulates Game 1 for $\mathcal{A}_{dis}$ when its challenger simulates $\mathtt{PseudoRanGame}(1^\lambda, 0)$ in Definition 9 since y^* is the evaluation of TPRF on the input(j^*, e^*). And $\mathcal{B}_{tprf}$ simulates Game 2 for $\mathcal{A}_{dis}$ when its challenger simulates $\mathtt{PseudoRanGame}(1^\lambda, 1)$ in Definition 9, since y^* is chosen randomly. Since the underlying the underlying threshold function TPRF satisfies Definition 9, these two games (1 and 2) are indistinguishable.

Now the success probability of a CCA adversary in the Game 2 is negligible by the one-time IND-CAP security of the underlying symmetric encryption scheme SYM, since k^* is only used once and to generate e^* which is the encryption of m_b. We skip the detailed reduction algorithm since it is straightforward and it is similar to two reductions presented above. The high level idea is a reduction adversary $\mathcal{B}_{sym}$ chooses a TPRF and a SignScheme, and runs the CCA adversary in the Game 2. Once the CCA adversary submits a challenge query $\mathrm{TSE.Enc}([\![\mathsf{sk}]\!]_{[n]}; [j^* : m_0, m_1; S^*]; pp)$, $\mathcal{B}_{sym}$ forwards m_0, m_1 to its challenger. After receiving e^*, it computes y^*, but it sets $\beta^* := \beta^{\$}$ for a randomly chosen $\beta^{\$}$. Finally, it returns $c_1^* := (j^*, e^*, \beta^{\$})$ and $c_2^* := \mathrm{Sign}_{\mathsf{sk}^s_{j^*}}(c_1^*)$ to the adversary. Afterwards, it simulates answers to the encryption and decryption queries similar

to the Game 2. Finally, $\mathcal{B}_{sym}$ returns the output of the CCA adversary. Overall, if the CCA adversary in the Game 2 guesses b with a non-negligible probability, the reduction adversary $\mathcal{B}_{sym}$ breaks the one-time IND-CAP security of SYM with a a non-negligible probability. □

6 Acknowledgment

This work is supported by the Luxembourg National Research Fund under the Junior CORE project QSP (C22/IS/17272217/QSP/Ebrahimi).

References

ABKM22. Gorjan Alagic, Chen Bai, Jonathan Katz, and Christian Majenz. Post-quantum security of the even-mansour cipher. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 458–487. Springer, Heidelberg, May / June 2022.

Agr19. Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 191–225. Springer, 2019.

AHU19. Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 269–295. Springer, Heidelberg, August 2019.

AMMR18. Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. Dise: Distributed symmetric-key encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1993–2010. ACM, 2018.

AMRS20. Gorjan Alagic, Christian Majenz, Alexander Russell, and Fang Song. Quantum-access-secure message authentication via blind-unforgeability. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 788–817. Springer, Heidelberg, May 2020.

ASY22. 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, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 8:1–8:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

ATTU16. Mayuresh Vivekanand Anand, Ehsan Ebrahimi Targhi, Gelo Noel Tabia, and Dominique Unruh. Post-quantum security of the CBC, CFB, OFB, CTR, and XTS modes of operation. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 44–63. Springer, Heidelberg, 2016.

BBC+21. Ritam Bhaumik, Xavier Bonnetain, André Chailloux, Gaëtan Leurent, María Naya-Plasencia, André Schrottenloher, and Yannick Seurin. QCB: Efficient quantum-secure authenticated encryption. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part I*, volume 13090 of *LNCS*, pages 668–698. Springer, Heidelberg, December 2021.

BDE+23. Maxime Buser, Rafael Dowsley, Muhammed Esgin, Clémentine Gritti, Shabnam Kasra Kermanshahi, Veronika Kuchta, Jason Legrow, Joseph Liu, Raphaël Phan, Amin Sakzad, Ron Steinfeld, and Jiangshan Yu. A survey on exotic signatures for post-quantum blockchain: Challenges and research directions. *ACM Comput. Surv.*, 55(12), March 2023.

BDF+11. Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Heidelberg, December 2011.

BDGM22. Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Factoring and pairings are not necessary for IO: circular-secure LWE suffices. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 28:1–28:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

BGG+18. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, August 2018.

BN08. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptol.*, 21(4):469–491, 2008.

BS23. Katharina Boudgoust and Peter Scholl. Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. *IACR Cryptol. ePrint Arch.*, page 16, 2023.

BZ13a. Dan Boneh and Mark Zhandry. Quantum-secure message authentication codes. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 592–608. Springer, Heidelberg, May 2013.

BZ13b. Dan Boneh and Mark Zhandry. Secure signatures and chosen ciphertext security in a quantum computing world. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 361–379. Springer, Heidelberg, August 2013.

CETU21. Tore Vincent Carstens, Ehsan Ebrahimi, Gelo Noel Tabia, and Dominique Unruh. Relationships between quantum IND-CPA notions. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 240–272. Springer, Heidelberg, November 2021.

CEV22. Céline Chevalier, Ehsan Ebrahimi, and Quoc Huy Vu. On security notions for encryption in a quantum world. In Takanori Isobe and Santanu Sarkar, editors, *Progress in Cryptology - INDOCRYPT 2022 - 23rd International Conference on Cryptology in India, Kolkata, India, December 11-14, 2022, Proceedings*, volume 13774 of *Lecture Notes in Computer Science*, pages 592–613. Springer, 2022.

CFHL21. Kai-Min Chung, Serge Fehr, Yu-Hsuan Huang, and Tai-Ning Liao. On the compressed-oracle technique, and post-quantum security of proofs of sequential work. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 598–629. Springer, Heidelberg, October 2021.

CGMS21. Mihai Christodorescu, Sivanarayana Gaddam, Pratyay Mukherjee, and Rohit Sinha. Amortized threshold symmetric-key encryption. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2758–2779. ACM Press, November 2021.

CMS19. Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 1–29. Springer, Heidelberg, December 2019.

DF89. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, 1989.

DFMS19. Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 356–383. Springer, Heidelberg, August 2019.

DFMS22. Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 677–706. Springer, Heidelberg, May / June 2022.

DFNS14. Ivan Damgård, Jakob Funder, Jesper Buus Nielsen, and Louis Salvail. Superposition attacks on cryptographic protocols. In Carles Padró, editor, *ICITS 13*, volume 8317 of *LNCS*, pages 142–161. Springer, Heidelberg, 2014.

DGRW18. Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 155–186. Springer, 2018.

DKPW12. Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message authentication, revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 355–374. Springer, 2012.

DMV22. Alexandre Duc, Robin Müller, and Damian Vizár. Diae: Re-rolling the dise. *IACR Cryptol. ePrint Arch.*, page 1275, 2022.

Ebr22. Ehsan Ebrahimi. Post-quantum security of plain OAEP transform. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-*

11, 2022, Proceedings, Part I, volume 13177 of *Lecture Notes in Computer Science*, pages 34–51. Springer, 2022.

EvW22. Ehsan Ebrahimi and Jeroen van Wier. Post-quantum plaintext-awareness. In Jung Hee Cheon and Thomas Johansson, editors, *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings*, volume 13512 of *Lecture Notes in Computer Science*, pages 260–285. Springer, 2022.

JJV02. Éliane Jaulmes, Antoine Joux, and Frédéric Valette. On the security of randomized CBC-MAC beyond the birthday paradox limit: A new construction. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*, volume 2365 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2002.

KL07. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

KLLN16. Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 207–237. Springer, Heidelberg, August 2016.

KM10. Hidenori Kuwakado and Masakatu Morii. Quantum distinguisher between the 3-round feistel cipher and the random permutation. In *IEEE International Symposium on Information Theory, ISIT 2010, June 13-18, 2010, Austin, Texas, USA, Proceedings*, pages 2682–2685. IEEE, 2010.

KM12. Hidenori Kuwakado and Masakatu Morii. Security on the quantum-type even-mansour cipher. In *Proceedings of the International Symposium on Information Theory and its Applications, ISITA 2012, Honolulu, HI, USA, October 28-31, 2012*, pages 312–316. IEEE, 2012.

Kra10. Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648. Springer, 2010.

LST21. Benoît Libert, Damien Stehlé, and Radu Titiu. Adaptively secure distributed PRFs from LWE. *Journal of Cryptology*, 34(3):29, July 2021.

MPS+02. Keith M. Martin, Josef Pieprzyk, Reihaneh Safavi-Naini, Huaxiong Wang, and Peter R. Wild. Threshold macs. In Pil Joong Lee and Chae Hoon Lim, editors, *Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers*, volume 2587 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2002.

MRH04. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, February 2004.

NPR99. Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and kdcs. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346. Springer, 1999.

SDFY94. Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 522–533. ACM, 1994.

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

Sim94. Daniel R. Simon. On the power of quantum computation. In *35th FOCS*, pages 116–123. IEEE Computer Society Press, November 1994.

TU16. Ehsan Ebrahimi Targhi and Dominique Unruh. Post-quantum security of the fujisaki-okamoto and OAEP transforms. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 192–216, 2016.

Unr12. Dominique Unruh. Quantum proofs of knowledge. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 135–152. Springer, Heidelberg, April 2012.

Wat06. John Watrous. Zero-knowledge against quantum attacks. In Jon M. Kleinberg, editor, *38th ACM STOC*, pages 296–305. ACM Press, May 2006.

WW21. Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 127–156. Springer, 2021.

YZ21. Takashi Yamakawa and Mark Zhandry. Classical vs quantum random oracles. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 568–597. Springer, Heidelberg, October 2021.

Zha12. Mark Zhandry. How to construct quantum random functions. In *53rd FOCS*, pages 679–687. IEEE Computer Society Press, October 2012.

Zha19. Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 239–268. Springer, Heidelberg, August 2019.

A Post-quantum Security

In this section, we discuss the security of a threshold functionality in the presence of a quantum polynomial-time adversary (QPT). We fix two terminologies:

- **Post-quantum Secure Pseudorandom Function.** We say a function f is post-quantum secure pseudorandom function if no QPT adversary with **classical access** to its oracle, can distinguish f from a truly random function.
- **Quantum Secure Pseudorandom Function.** We say a function f is quantum secure pseudorandom function if no QPT adversary with **superposition access** to its oracle, can distinguish f from a truly random function.

A.1 Discussion

The cryptographic community are reevaluating the traditional assumptions and security model in cryptography due to the rapid progress in quantum computing and the dramatic effects of a full-scale quantum computer on cryptography [Sho94]. Towards this goal, the first essential step would be replacing the quantumly broken computational assumptions with quantum-hard assumptions, though, some classical security proofs may not carry to the post-quantum setting trivially ([Wat06, Unr12]) and this replacement might not be sufficient to claim the post-quantum security. The extension of this reevaluation is a conducive line of research, where new security definitions have been developed in the *superposition-access model* [KM10, KM12, Zha12, DFNS14, BZ13a, BZ13b, ATTU16, KLLN16, AMRS20, BBC$^+$21, CETU21, ABKM22, EvW22, CEV22] and many cryptographic schemes have been proven (in)secure in this model. In the superposition-access model, a quantum adversary may be able to run the cryptographic primitive in superposition of inputs. One example is the security proofs in the quantum random oracle model [BDF$^+$11] in which the adversarial superposition queries are allowed to the random oracle.

Although, the security proofs in the quantum random oracle [BDF$^+$11, TU16, Zha19, DFMS19, CMS19, AHU19, YZ21, CFHL21, DFMS22, Ebr22] become standard and a necessity, the community may still consider a superposition access to a classical cryptographic primitive as an unnatural scenario. For instance a quote from [BDE$^+$23] for a threshold signature scheme: "Since we are considering only classical protocols, there is no real-world scenario in which an adversary would have quantum access to signature generation, so this restriction is quite reasonable". So, even with the existence of many research papers in the *superposition-access model*, often a research in this model is evaluated as a more conservative security requirement and as a resolution of the authors inquisitiveness rather than addressing a realistic threat. However, we raise some evidence that the security in the *superposition-access model* might be necessary in some settings. Namely, in a distributed setting (for instance for a threshold signature scheme) some parties are involved in a joint evaluation of a functionality (signature) and hypothetically a malicious quantum entity is able to corrupt a subset of these parties, and therefore the quantum entity has a superposition access to a partial evaluation of the functionality. So if the partial evaluation of the functionality is vulnerable to a superposition attack, then the security of the overall evaluation may be affected negatively. In other words, our observation is that we may not be able to conclude a threshold system (for instance a threshold signature scheme) post-quantum secure even if it is based on a quantum-secure computational assumption.

To back our claim, we discuss a security concern in a threshold pseudorandom function from [NPR99] in the post-quantum setting. More specifically, we consider a threshold pseudorandom function in [NPR99] that is constructed from a family of pseudorandom functions as follows. At a high-level, a threshold pseudorandom function TPRF is defined as $\mathrm{TPRF}(x) = \oplus_i \mathtt{PRF}_{\mathsf{sk}_i}(x)$, where $\{\mathtt{PRF}_k\}_k$ is a family of keyed pseudorandom functions. (The exact parameters are not

needed for our purpose in this discussion.) It is clear that one needs all the keys $\{\mathsf{sk}_i\}_i$ to evaluate TPRF on some input x. Now if $\{\mathtt{PRF}_k\}_k$ functions family is post-quantum secure (secure against a quantum adversary with the classical access to $\{\mathtt{PRF}_k\}_k$), but it is vulnerable to superposition attack, for instance if there is a hidden period s in the construction of $\{\mathtt{PRF}_k\}_k$, we can not conclude TPRF post-quantum secure. For the reason that a quantum adversary that has corrupted a party j and possesses its secret key sk_j can implement $\mathtt{PRF}_{\mathsf{sk}_j}$ in superposition of inputs and obtain s using the Simon's algorithm [Sim94]. By the construction, TPRF is periodic in s as well and therefore TPRF should not be considered as a secure threshold pseudorandom function. In other words, we can not conclude the post-quantum security of a TPRF constructed from a post-quantum secure $\{\mathtt{PRF}_k\}_k$.

Now we explain why our assumption of constructing a periodic function without including the period as a part of the secret key is reasonable. We recall a quantum attack to the 3-Round Feistel Cipher (FC) [KM10]. In [KM10], it is shown how to construct a periodic function from the first half of the Feistel Cipher's output (FC_1). In more details, $\mathrm{FC}_1(x_1, x_2) = x_2 \oplus \mathtt{PRF}_{\mathsf{sk}_2}(x_1 \oplus \mathtt{PRF}_{\mathsf{sk}_1}(x_2))$. And the periodic function f is defined as

$$f(b, x) = \begin{cases} \mathrm{FC}_1(x, \alpha) \oplus \beta & b = 0 \\ \mathrm{FC}_1(x, \beta) \oplus \alpha & b = 1 \end{cases}$$

where α and β are distinct fixed strings. It is straightforward to show that f is periodic in $1 \| \mathtt{PRF}_1(\alpha) \oplus \mathtt{PRF}_1(\beta)$. This period is not a part of the secret key and it is introduced by the construction of the Feistel Cipher itself. (See Appendix A.2 for a separation based on post-quantum obfuscation.)

Take-home Message: The take-home message from our observation is that a post-quantum secure pseudorandom function family that is not secure in the superposition-access model will not result in a post-quantum secure threshold pseudorandom function using the NPR approach.

A.2 Separation

Theorem 8.4 in [AMMR18] formally states that when $\mathtt{PRF}$ is a pseudorandom function, the NPR threshold function sketched above is a TPRF that satisfies *pseudorandomness*. We show that theorem 8.4 does not hold if we replace PPT adversary with a QPT adversary in the *pseudorandomness* definition under the existence of a post-quantum obfuscator.

Observation 1. *Let* $\{\mathtt{PRF}'_{k_i}\}_i$ *be a family of pseudorandom functions from* $\{0,1\}^n$ *to* $\{0,1\}^m$. *For each* i, *we define* $\mathtt{PRF}_{k_i}(x) := \mathtt{PRF}'_{k_i}(x) \oplus \mathtt{PRF}'_{k_i}(x \oplus s)$ *where* s *is a randomly chosen element from the domain. If there exists a post-quantum obfuscator* [6] *for the value* s, *the* NPR *threshold function constructed from this family is not post-quantum* TPRF *with respect to the Definition 9.*

[6] See [Agr19, WW21, BDGM22] for recent advances on post-quantum obfuscation.

First, we show that $\{\mathtt{PRF}_k\}_k$ is a family of pseudorandom functions if $\{\mathtt{PRF}'_k\}_k$ is a family of pseudorandom functions. Our assumption is that the secret period s is obfuscated in the program of $\mathtt{PRF}$. Then, we show that the NPR threshold function that uses $\{\mathtt{PRF}\}$ ($F_k(x) = \oplus_{i=1}^{d} \mathtt{PRF}_{k_i}(x)$) is not post-quantum secure.

Game 0. In this game, a PPT adversary $\mathcal{A}$ has oracle access to $\mathtt{PRF}_{k_i}$ for a randomly chosen key k_i. The adversary makes polynomial number of queries to its oracle and returns an output b.

Game 1. Let $f : \{0,1\}^n \to \{0,1\}^m$ be a function chosen uniformly at random from the set of all functions. In this game, the adversary has oracle access to $\mathtt{PRF}_f(x) := f(x) \oplus f(x \oplus s)$. We show that Game 0 and Game 1 are indistinguishable if $\mathtt{PRF}'_{k_i}$ is a pseudorandom function. Let $\mathcal{A}$ be an adversary that distinguishes Game 0 and Game 1 with some non-negligible probability ϵ making q queries. We construct an adversary $\mathcal{B}$ that distinguishes $\mathtt{PRF}'$ from a truly random function. Namely, the adversary $\mathcal{B}^O$ chooses a random element s, runs $\mathcal{A}$ and answers to its queries as follows. On input x received from $\mathcal{A}$, the adversary $\mathcal{B}^O$ queries x and $x \oplus s$ to its oracle O to receive $O(x)$ and $O(x \oplus s)$, respectively. Then it returns $O(x) \oplus O(x \oplus s)$ to $\mathcal{A}$. At the end, $\mathcal{B}^O$ returns $\mathcal{A}$'s output. It is clear that $\mathcal{B}^O$ distinguishes $\mathtt{PRF}'_{k_i}$ from a truly random function with a probability ϵ and making $2q$ queries to its oracle.

Game 2. Let $F : \{0,1\}^n \to \{0,1\}^m$ be a truly random function. In this game, the adversary has oracle access to F. Under the existence of a post-quantum obfuscator [WW21, BDGM22], the two games 1 and 2 are indistinguishable unless the adversary makes a query on two inputs x and $x \oplus s$ that occurs with a probability at most $q^2/2^n$ due to the *birthday attack*. This shows that $\{\mathtt{PRF}_k\}_k$ is a family of pseudorandom functions.

Quantum Attack. We show how a quantum polynomial-time adversary is able to break the Definition 9. Note that a QPT adversary $\mathcal{A}$ that has corrupted a party j can employ the Simon's algorithm on the function $\mathtt{PRF}_{k_j}$ to recover s. Then, it queries (Eval, x, i) for as many as i that is enough to compute $F_k(x)$. Then, it submits $(x \oplus s, S, \emptyset)$ where $S \subseteq [n] \setminus \{j\}$ and $|S| \geq t$ as a challenge query. It is clear that when $b = 0$, the adversary expects to receive $F_k(x)$. On the other hand, when $b = 1$, the adversary receives a random value. This breaks the Definition 9.

A.3 Effect on Our Result

Most of our result in this paper hold if we replace a PPT adversary with a QPT adversary in the security definitions and theorems. Only Theorem 3 needs further modification: That is, it holds if the underlying function family $\mathtt{PRF}$ is *Quantum Secure*. (In other words, it does not hold if we use a *post-quantum* secure $\mathtt{PRF}$.)