DiAE: Re-rolling the DiSE

Alexandre Duc[1], Robin Müller[1], and Damian Vizár[2]

[1] IICT, School of Management and Engineering Vaud, HES-SO University of Applied Sciences and Arts Western Switzerland
{alexandre.duc,robin.muller}@heig-vd.ch

[2] Swiss Center for Electronics and Microtechnology (CSEM)
damian.vizar@csem.ch

Abstract. The notion of distributed authenticated encryption was formally introduced by Agrawal et al. in ACM CCS 2018. In their work, they propose the DiSE construction building upon a distributed PRF (DPRF), a commitment scheme and a PRG. We show that most of their constructions do not meet some of the claimed security guarantees. In fact, *all the concrete instantiations* of DiSE, as well as multiple follow-up papers (one accepted at ACM CCS 2021), fail to satisfy their strongly-secure definitions. We give simple fixes for these constructions and prove their security. We also propose a *new construction* DiAE using an *encryptment* instead of a commitment. This modification dispenses with the need to buffer the entire message throughout the encryption protocol, which in turn enables implementations with constant RAM footprint and online message encryption. This is particularly interesting for constrained IoT devices. Finally, we implement and benchmark DiAE and show that it performs similarly to the original DiSE construction.

Keywords: Cryptography · Encryption · Authentication · Internet of Things.

1 Introduction

One of the fundamental problems related to the use of cryptography is ensuring the protection of keys, be it symmetric (e.g., for authenticated encryption schemes) or asymmetric (e.g., for digital signature schemes). In numerous applications, simply storing the keys in the non-volatile memory of a device does not suffice to meet the security requirements of the application. One established solution to this problem relies on specialized secure hardware (HW), such as HW security modules or secure elements. Yet, this is often cost-prohibitive and creates undesirable HW lock-in. An alternative, HW-independent approach to reducing the risk related to compromised keys is to use threshold cryptography. In a nutshell, a threshold "flavor" of a cryptographic primitive, such as a secret key encryption scheme, uses a key split into *shares* and the algorithms of the primitive, such as the encryption algorithm, become interactive protocols. The shares are distributed to $n \geq 2$ parties and evaluation of the primitive requires *interaction* of a certain subset of these parties. This is often simply any subset of size at least equal to a *threshold* $t \leq n$, such that the protocol execution does not reveal information about other parties' key shares.

Distributed encryption. The problem of distributing cryptographic primitives over multiple parties has been studied for nearly three decades [14, 36]. Most of the effort focused on threshold public key cryptography [6–9, 13, 19, 20, 22, 39]. This under-representation of secret key cryptography was addressed by Naor, Pinkas and Reingold in 1999, who proposed several constructions of a *distributed pseudorandom function* (DPRF) [28] and recently by Agrawal et al. in 2018 [2], who provided a security definition for *distributed authenticated encryption* and a construction DiSE (for Distributed Symmetric-key Encryption). Their definitions capture two levels of security. The first suffices if encryption only needs to be distributed. A stronger variant is required by most applications that also need a secure distributed decryption (or when re-encryption of a message cannot be requested). Their construction builds on a DPRF, a commitment scheme and a pseudo-random generator (PRG). In brief, a party j wishing to encrypt a plaintext m computes a commitment α on m with randomness ρ. Then, $j||\alpha$ is used as the input of an interactive evaluation of the DPRF. The DPRF's output z is fed to the PRG to derive a keystream and encrypt $m||\rho$ in a stream-cipher-like fashion. The final ciphertext is output as $(PRG(z) \oplus (m||\rho), j, \alpha)$. The decryption works analogously but verifies the integrity of the message by opening the commitment α with the message m and the randomness ρ. Agrawal et al. provide two variants of DiSE, with weaker and stronger security properties, complete with security proofs. They further provide multiple instances, based on a DPRF construction of Naor et al. that relies on the hardness of the DDH assumption, as well as a combinatorial DPRF based on AES, complete with performance benchmarks showing practical computational and communication complexity.

Applications of threshold encryption. A distributed encryption with practical complexity finds many real-world applications, as also hinted by NIST's recent activity in this domain [31]. Threshold cryptography can improve security of various applications through distributing keys and thus decreasing the risk related to their compromise in network authentication protocols like Kerberos [29], in secret management systems [21,35,40] or in Internet of Things (IoT) devices (consumer-grade or otherwise). In particular in IoT, threshold cryptography represents a HW-independent hardening of the so called Edge, i.e., the subset of an application consisting of the end-devices. These are often cost sensitive (and thus poorly secured), but at the same time exposed, leading to devastating attacks [34]. Threshold cryptography can compensate this and help protect data created and encrypted in the Edge. For example, when a group of devices uses distributed encryption to protect sensitive data (such as security camera footage) uploaded to cloud, an attacker that exfiltrates all secrets from a single device does not get the ability to decrypt past data, nor to create new, spoofed ciphertext, unless it can interact with the remaining devices using the local communication channels. In an analogous case, compromising a single device would not give perpetual access to firmware updates secured with threshold encryption. Threshold encryption also lets a group of devices act as a single entity, such as the devices in a smart household representing the household, personal electronics representing their owner, or autonomous drones acting as a swarm being just a few examples.

Threshold encryption for IoT. To be truly practical for (not only) IoT applications, a threshold encryption scheme must decouple the local message encryption from the interactive part of the protocol, such that the former can be completed before the latter, with only a constant-size state being carried over. The same in reverse order must also apply for decryption. For example, in a "smart dust" sensor mote [33] with as little as 4 KB of RAM [12] an implementation with a constant RAM footprint is absolutely vital. The DiSE construction does not allow this; the message must be committed to *before* the interactive DPRF evaluation and must be buffered to be encrypted *afterwards*.

1.1 Our Contribution

We first show that the strongly secure DiSE construction in [2] fails to meet the claimed security guaranty because their generic strongly secure DPRF construction does not meet the correctness property. As a result, this issue affects all the concrete instantiations of DiSE DPRFs, as well as recent follow-up papers [5, 11, 27]. Additionally, we show that the NIZK proof system in one of the constructions is invalid (the proof's verification always outputs 0) which renders the DPRF unusable.

We propose simple fixes that restore the security of the DPRF constructions. We then propose *a new construction* of distributed authenticated encryption using an *encryptment* [16] instead of the commitment and prove security of our new construction. We call our construction *DiAE*. The advantage of our solution is that the message encryption and DPRF protocol execution are no longer tightly interlocked. The computation (and subsequent transmission) of most of the ciphertext can take place *without the need to wait for the completion of the protocol*, such that only a constant-size state (a binding tag) needs to be preserved. This is favorable for applications that cannot buffer the entire message during the protocol. As our encryptment instance of choice allows online (a.k.a. blockwise) processing, the entire DiAE can be implemented with a constant RAM footprint.

1.2 Related Work

Threshold Symmetric Encryption schemes. DiSE [2] initiated the work on threshold symmetric encryption schemes by formalizing definitions and providing practical constructions, as well as benchmarks. Several works followed up recently, extending DiSE to the adaptive setting [27] and improving the performance when encrypting a large group of messages [11].

DPRFs. Distributed PseudoRandom Functions were first described by Micali and Sidney [26] and have then been used in constructions of other primitives [15, 28, 30] and various constructions [1, 2, 5, 11]. Recently, Libert et al. provided an adaptively secure DPRF based on the Learning With Errors (LWE) problem [25], which could give rise to post-quantum threshold authenticated encryption. Mukherjee shows that the adaptive security of DiSE depends directly on the adaptive security of the underlying DPRF [27], indicating that improvements on DPRFs could lead to straightforward improvements of DiSE.

1.3 Roadmap

Section 2 introduces notations and definitions from prior work. In Section 3, we present our attacks on DiSE's DPRF constructions. In Section 4, we fix the DPRFs and propose our new construction using encryptment. We show our implementation results in Section 5 and conclude in Section 6.

2 Preliminaries

We first explain the notations used throughout the paper, then we present the main definitions used by our construction. We provide references to additional definitions in Appendix B.

2.1 Notations

We let n and t respectively denote the number of key shares (or parties) and the threshold in secret sharing or in a threshold primitive. Given an integer $i \in \mathbb{N}$, we use $[i]$ to represent the set $\{1, 2, \dots, i\}$. We count the number of items in a set S with $|S|$ or with $\#\{1, 2\} = 2$ when building the set explicitly. We let $y \leftarrow \mathsf{F}(x)$ denote assigning the output of an algorithm F to a variable y, while $\mathsf{F}(x) \rightarrow y$ denotes the event that F outputs a value y. Sampling a variable ρ uniformly from a finite set S is denoted by $\rho \leftarrow\$ S$. The Lagrange basis polynomial (or Lagrange coefficient) using the element i of the set S is denoted as $\lambda_{0,i,S}$. The concatenation of two byte strings a and b is indicated by $a \| b$. The length in bits of a byte string a is obtained with $\mathsf{len}(a)$. In a distributed protocol, $[j : x]$ signifies that the value x is private to the party j. Similarly, $[\![sk]\!]_{[n]}$ signifies that each party i, $i \in [n]$ owns a private value sk_i.

$f : \mathbb{N} \longrightarrow \mathbb{N}$ is negligible if for all $c \in \mathbb{N}$ there exists a $n_0 \in \mathbb{N}$ such that for all integers $n \geq n_0$, we have $f(n) < n^{-c}$. The abbreviation PPT stands for for probabilistic polynomial-time.

2.2 Distributed Pseudorandom Function [2]

We explain below the concept of distributed pseudorandom functions (DPRFs) using the definitions from [2].

A distributed pseudorandom function DPRF is a tuple of three algorithms $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Combine})$:

$\mathsf{Setup}(1^k, n, t) \to (\{sk_i\}_{i=1}^n, pp_{\mathsf{DPRF}})$: Randomized algorithm that takes a security parameter 1^k, a number of parties n and a threshold t; outputs n secret shares and public parameters.

$\mathsf{Eval}(sk_i, x, pp_{\mathsf{DPRF}}) \to z_i/\bot$: Compute the partial output share z_i for a DPRF input x with the secret share sk_i and the public parameters generated by Setup.

$\mathsf{Combine}(\{(i, z_i)\}_{i\in S}, pp_{\mathsf{DPRF}}) \to z$: Combine the DPRF output shares from parties $i \in S,\ |S| \geq t$ to get the DPRF output z.

In a nutshell, a DPRF is a threshold variant of a keyed PRF where the output can only be evaluated interactively by a set of t parties.

Definition 1 (Consistency). DPRF *is* consistent *if for all* $(\{sk_i\}_{i=1}^n, pp_{\mathsf{DPRF}})$ *output by* $\mathsf{Setup}(1^k, n, t)$ *and any* $x \in \{0,1\}^*$*, any subsets* $S, S' \subseteq [n]$ *of size at least* t *used in* $\mathsf{Combine}(\{(i, z_i)\}_{i\in S}, pp_{\mathsf{DPRF}})$ *output the same* z*.*

$$\Pr\big[\mathsf{Combine}(\{(i, z_i)\}_{i\in S}, pp_{\mathsf{DPRF}}) = \mathsf{Combine}(\{(j, z_j)\}_{j\in S'}, pp_{\mathsf{DPRF}}) \neq \bot\big] = 1$$

where $z_i \leftarrow \mathsf{Eval}(sk_i, x, pp_{\mathsf{DPRF}})$ *for* $i \in S$ *and* $z_j \leftarrow \mathsf{Eval}(sk_j, x, pp_{\mathsf{DPRF}})$ *for* $j \in S'$*.*

Definition 2 (Pseudorandomness [2, Def. 5.3]). *A* DPRF *is* pseudorandom *if the output of* $\mathsf{Combine}$ *appears pseudorandom independently of the evaluations used, even if some evaluations are corrupted. No information should be gained on the output without the knowledge of at least* t *evaluations.* DPRF *is* pseudorandom *for all* PPT *adversaries if there exists a negligible function such that:*

$$\left|\Pr\big[\mathsf{PseudoRand}^{\mathcal{A}}_{\mathsf{DPRF}}(1^k, 0) = 1\big] - \Pr\big[\mathsf{PseudoRand}^{\mathcal{A}}_{\mathsf{DPRF}}(1^k, 1) = 1\big]\right| \leq \mathbf{negl}(k)$$

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

Definition 3 (Correctness [2, Def. 5.4]). DPRF *is* correct *if the output of* $\mathsf{Combine}$ *is the expected output; either there are no corruptions and the true output is computed, or the corruption is detected. This is true for all* PPT *adversaries if there exists a negligible function such that:*

$$\Pr\big[\mathsf{Correctness}^{\mathcal{A}}_{\mathsf{DPRF}}(1^k) = 1\big] \geq 1 - \mathbf{negl}(k)$$

where $\mathsf{Correctness}^{\mathcal{A}}_{\mathsf{DPRF}}(1^k)$ *is the following:*

Initialization. *Run* $\mathsf{Setup}(1^k, n, t)$ *to get* $(\{sk_i\}_{i=1}^n, pp_{\mathsf{DPRF}})$. *Give* pp_{DPRF} *to* $\mathcal{A}$.
Corruption. *Receive the set of corrupt parties* C *from* $\mathcal{A}$, *where* $|C| < t$. *Give the secret keys* $\{sk_i\}_{i\in C}$ *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* $\mathsf{Eval}(sk_i, x, pp_{\mathsf{DPRF}})$ *to* $\mathcal{A}$. *Repeat this step as many times as* $\mathcal{A}$ *desires.*
Computation. *Receive a set* S *of size at least* t, *an input* $x^\star$ *and evaluations* $\{(i, z_i^\star)\}_{i\in S\cap C}$ *from* $\mathcal{A}$. *Let* $z_j \leftarrow \mathsf{Eval}(sk_j, x^\star, pp_{\mathsf{DPRF}})$ *for* $j \in S$ *and* $z_i' \leftarrow \mathsf{Eval}(sk_i, x^\star, pp_{\mathsf{DPRF}})$ *for* $i \in S \setminus C$. *Also, let* $z \leftarrow \mathsf{Combine}(\{(j, z_j)\}_{j\in S}, pp_{\mathsf{DPRF}})$ *and* $z^\star \leftarrow \mathsf{Combine}(\{(i, z_i')\}_{i\in S\setminus C} \cup \{(i, z_i^\star)\}_{i\in S\cap C})$. *Output* 1 *if* $z^\star \in \{z, \perp\}$; *else, output* 0.

Definition 4 (DPRF security). *A* DPRF *is* secure *if it satisfies* consistency *(Def. 1) and* pseudorandomness *(Def. 2); it is* strongly-secure *if it also satisfies* correctness *(Def. 3). A* strongly-secure DPRF *ensures that the output of* Combine *cannot be corrupted by adversaries.*

2.3 Threshold Symmetric Encryption (TSE)

A TSE scheme makes use of two distributed protocols to interactively encrypt messages or decrypt ciphertexts with the help of a threshold t number of parties. It can be described as a tuple of three algorithms (Setup, DistEnc, DistDec):

$\mathsf{Setup}(1^k, n, t) \rightarrow ([\![sk]\!]_{[n]}, pp)$: Randomized algorithm that takes a security parameter 1^k, a number of parties n and a threshold t; outputs n secret shares and public parameters. The share sk_i is given to party i for all $i \in [n]$.

$\mathsf{DistEnc}([\![sk]\!]_{[n]}, [j : m, S], pp) \rightarrow [j : c/\perp]$: Distributed protocol where an initiating party j encrypts a message m with the help of t parties in S ($|S| \geq t$) to generate an authenticated ciphertext c. Only the party j knows the message m and the set S. Similary, j is the only party that learns the output $c/\perp$.

$\mathsf{DistDec}([\![sk]\!]_{[n]}, [j : c, S], pp) \rightarrow [j : m/\perp]$: Distributed protocol where an initiating party j decrypts an authenticated ciphertext c with the help of t parties in S ($|S| \geq t$) to recover the message m. Only the party j knows the ciphertext c and the set S. Similary, j is the only party that learns the message m or $\perp$.

In various definitions provided below, (indirect) encryption and (indirect) decryption queries are represented as two separated steps to improve readability. However, the adversary is allowed to alternate between encryption and decryption queries. Intuitively, the correctness definition follows the standard authenticated encryption definition, message privacy can be seen as a distributed variant of IND-CPA security while authenticity satisfies a variant of ciphertext integrity.

Definition 5 (Consistency). TSE *is* consistent *if for any* $n, t \in \mathbb{N}$, $t \leq n$ *and all* $([\![sk]\!]_{[n]}, pp)$ *output by* Setup, *it holds that for any message* m, *any two sets* $S, S' \subseteq [n]$ *of size at least* t *and any two parties* $j \in S$, $j' \in S'$, *can be used to execute a distributed encryption and a distributed decryption with high probability when all parties behave honestly:*

$$\mathsf{Pr}\Big[[j' : m] \leftarrow \mathsf{DistDec}([\![sk]\!]_{[n]}, [j' : c, S'], pp) \;\mid\; [j : c] \leftarrow \mathsf{DistEnc}([\![sk]\!]_{[n]}, [j : m, S], pp)\Big] \geq 1 - \mathbf{negl}(k)$$

Definition 6 (Correctness [2, Def. 6.4]). TSE *is* correct *if all ciphertexts* c *output by* DistEnc, *either result in the original message* m *or in* $\perp$ *when decrypted with* DistDec. *Formally,* TSE *is* correct *if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function such that:*

$$\mathsf{Pr}\big[\mathsf{Correctness}_{\mathsf{TSE}}^{\mathcal{A}}(1^k) = 1\big] \geq 1 - \mathbf{negl}(k)$$

where $\mathsf{Correctness}^{\mathcal{A}}_{\mathsf{TSE}}(1^k)$ *is the following:*

Initialization*. Run* $\mathsf{Setup}(1^k, n, t)$ *to get* $([\![sk]\!]_{[n]}, pp)$*. Give pp to* $\mathcal{A}$*.*

Corruption*. Receive the set of corrupt parties* C *from* $\mathcal{A}$*, where* $|C| < t$*. Give the secret keys* $\{sk_i\}_{i\in C}$ *of these parties to* $\mathcal{A}$*.*

Encryption*. Receive* $(\mathsf{Encrypt}, j, m, S)$ *from* $\mathcal{A}$ *where* $j \in S \setminus C$ *and* $|S| \geq t$*. Initiate* DistEnc *from party* j *with inputs* m*,* S *and pp. If* j *outputs* $c = \perp$ *at the end, output* 1 *and stop. Otherwise, continue.*

Decryption*. Receive* $(\mathsf{Decrypt}, j', S')$ *from* $\mathcal{A}$ *where* $j' \in S' \setminus C$ *and* $|S'| \geq t$*. Initiate* DistDec *from party* j' *with inputs* c*,* S', pp*. Let* $m^\star$ *be the output of* j'*. Note that* j' *(resp.* S'*) may be equal to* j *(resp.* S*).*

Output*. Output* 1 *if* $m^\star \in \{m, \perp\}$*,* 0 *otherwise.*

TSE *is a* strongly correct *scheme if it satisfies* correctness*, but with the following output step:*
Output*. If all the parties contacted during* DistDec *behaved honestly, output* 1 *iff* $m^\star = m$*. Otherwise, if some parties behaved maliciously, output* 1 *if* $m^\star \in \{m, \perp\}$*. Output* 0 *if all conditions are false.*

Definition 7 (Message privacy [2, Def. 6.6]). TSE *is* message private *if an adversary* $\mathcal{A}$ *cannot distinguish between the distributed encryption of two chosen messages. Formally,* TSE *satisfies message privacy if for all* PPT *adversaries* $\mathcal{A}$ *there exists a negligible function such that:*

$$\left|\mathsf{Pr}\left[\mathsf{MsgPriv}^{\mathcal{A}}_{\mathsf{TSE}}(1^k, 0) = 1\right] - \mathsf{Pr}\left[\mathsf{MsgPriv}^{\mathcal{A}}_{\mathsf{TSE}}(1^k, 1) = 1\right]\right| \leq \mathbf{negl}(k)$$

where $\mathsf{MsgPriv}^{\mathcal{A}}_{\mathsf{TSE}}(1^k, b)$ *is the following:*

Initialization*. Run* $\mathsf{Setup}(1^k, n, t)$ *to get* $([\![sk]\!]_{[n]}, pp)$*. Give pp to* $\mathcal{A}$*.*

Corruption*. Receive the set of corrupt parties* C *from* $\mathcal{A}$*, where* $|C| < t$*. Give the secret keys* $\{sk_i\}_{i\in C}$ *of these parties to* $\mathcal{A}$*.*

Pre-challenge encryption queries*. In reply to* $\mathcal{A}$*'s encryption query* $(\mathsf{Encrypt}, j, m, S)$ *where* $j \in S$ *and* $|S| \geq t$*, do the following. If* $j \in C$*, run an instance of* DistEnc *with* $\mathcal{A}$*. Otherwise, if* $j \notin C$*, initiate an instance of* DistEnc *from party* j *with inputs* m*,* S *and pp; the output of* j *is given to* $\mathcal{A}$*. Repeat this step as many times as requested.*

Pre-challenge indirect decryption queries*. In reply to* $\mathcal{A}$*'s decryption query* $(\mathsf{Decrypt}, j, c, S)$ *where* $j \in S \setminus C$ *and* $|S| \geq t$*, party* j *initiates* DistDec *with inputs* c*,* S *and pp. Repeat this step as many times as requested.*

Challenge*.* $\mathcal{A}$ *outputs* $(\mathsf{Challenge}, j^\star, m_0, m_1, S^\star)$ *where* $\mathsf{len}(m_0) = \mathsf{len}(m_1)$*,* $j \in S^\star \setminus C$ *and* $|S^\star| \geq t$*. Initiate* DistEnc *from party* $j^\star$ *with inputs* $m_b, S^\star$ *and pp. Let* $c^\star/\perp$ *be the output of* $j^\star$ *at the end of* DistDec*. Give* $c^\star/\perp$ *to* $\mathcal{A}$*.*

Post-challenge encryption queries*. Same as* Pre-challenge encryption queries*.*

Post-challenge indirect decryption queries*. Same as* Pre-challenge indirect decryption queries*.*

Guess*. Finally,* $\mathcal{A}$ *returns a guess* b'*. Output* b'*.*

Definition 8 (Authenticity [2, Def. 6.8]). *The standard notion of* TSE authenticity *ensures that adversaries cannot forge a ciphertext when all parties behave honestly during the decryption. In the game, the attacker learns* l *ciphertexts via encryption/decryption queries and must output one more ciphertext that decrypts into a message to win the authenticity game. Formally,* TSE *satisfies* authenticity *if for all* PPT *adversaries* $\mathcal{A}$ *there exists a negligible function such that:*

$$\mathsf{Pr}\left[\mathsf{Auth}^{\mathcal{A}}_{\mathsf{TSE}}(1^k) = 1\right] \leq \mathbf{negl}(k)$$

where $\mathsf{Auth}^{\mathcal{A}}_{\mathsf{TSE}}(1^k)$ *is the following:*
Initialization. *Run* $\mathsf{Setup}(1^k, n, t)$ *to get* $([\![sk]\!]_{[n]}, pp)$*. Give* pp *to* $\mathcal{A}$*. Initialize a counter* $\mathsf{ct} := 0$ *and an ordered list* $L_{\mathsf{ctxt}} := \varnothing$*.*
Corruption. *Receive the set of corrupt parties* C *from* $\mathcal{A}$*, where* $|C| < t$*. Give the secret keys* $\{sk_i\}_{i \in C}$ *of these parties to* $\mathcal{A}$*. Define the corruption gap as* $\delta := t - |C|$*.*
Encryption queries. *On receiving* $(\mathsf{Encrypt}, j, m, S)$ *from* $\mathcal{A}$ *where* $j \in S$ *and* $|S| \geq t$*, do the following. If* $j \in C$*, run an instance of* $\mathsf{DistEnc}$ *with* $\mathcal{A}$ *and increment* ct *by* $|S \setminus C|$ *(i.e. the number of honest parties in set* S*). Otherwise, if* $j \notin C$*, initiate an instance of* $\mathsf{DistEnc}$ *from party* j *with inputs* m*,* S*,* pp *and append the ciphertext generated at the end to* L_{ctxt}*.*
Decryption queries. *On receiving* $(\mathsf{Decrypt}, j, c, S)$ *from* $\mathcal{A}$ *where* $j \in S$ *and* $|S| \geq t$*, do the following. If* $j \in C$*, run an instance of* $\mathsf{DistDec}$ *with* $\mathcal{A}$ *and increment* ct *by* $|S \setminus C|$ *(i.e. the number of honest parties in set* S*). Otherwise, if* $j \notin C$*, initiate an instance of* $\mathsf{DistEnc}$ *from party* j *with inputs* c*,* S*,* pp*.*
Targeted decryption queries. *On receiving* $(\mathsf{TargetDecrypt}, j, \ell, S)$ *from* $\mathcal{A}$ *where* $j \in S \setminus C$ *and* $|S| \geq t$*, initiate an instance of* $\mathsf{DistDec}$ *from party* j *with inputs* c*,* S*,* pp*. For this type of query,* c *is the* ℓ*-th element of* L_{ctxt}*.*
Forgery. *Let* $l := \lfloor \mathsf{ct}/\delta \rfloor$*.* $\mathcal{A}$ *outputs* $((j_1, S_1, c_1), \ldots, (j_{l+1}, S_{l+1}, c_{l+1}))$ *where* $j_i \notin C$ *and* $|S_i| \geq t$ *for every* $i \in [l+1]$ *and* $c_u \neq c_v$ *for any* $u \neq v \in [l+1]$ *(i.e. ciphertexts are not repeated). For every* $i \in [l+1]$*, party* j *initiates* $\mathsf{DistDec}$ *with inputs* c_i*,* S_i*,* pp*. In that instance, all parties in* S_i *are required to behave honestly. Output* 0 *if any* j_i *outputs* $\perp$*. Otherwise, output* 1*.*

TSE *satisfies* strong authenticity *if it satisfies* authenticity *with a modified forgery phase; the parties are* not *required to behave honestly during the verification of the forgery. This ensures that adversaries cannot forge ciphertexts when distributed decryption is used.*

Definition 9 (TSE security). *A* TSE *scheme is* (strongly) secure *if it is* consistent *(Def. 5), satisfies* (strong) correctness *(Def. 6),* message privacy *(Def. 7) and* (strong) authenticity *(Def. 8).*

A *strongly secure* TSE scheme is required in two different scenarios; either when distributed decryption is necessary, or when the ciphertexts are not decrypted quickly. The former is to prevent forgeries during the distributed decryption. The latter provides a guarantee that the ciphertext was properly encrypted; otherwise, the generated ciphertext may be invalid due to corruptions during the distributed encryption, and will never decrypt into the original message. Looking forward, a strongly secure DPRF is sufficient to make both DiSE and DiAE strongly secure TSE schemes.

2.4 Encryptment scheme [16]

An encryptment scheme EC is a form of committing encryption where the ciphertext/tag is a commitment to the encryption key (and optionally the authenticated data) in addition to the message. The encryptment scheme described by Dodis et al. [16] is a tuple of algorithms $\mathsf{EC} := (\mathsf{EK_g}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$:

$\mathsf{EK_g} \to K_{\mathsf{EC}}$: Takes no input and generates a key K_{EC} at random.

$\mathsf{EC}(K_{\mathsf{EC}}, A, m) \to (c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$: Committing encryption with the key K_{EC} on the message m and the associated data (AD) A, where c_{EC} is the ciphertext and τ_{EC} the binding tag. We refer to the pair $(c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$ as an encryptment.

$\mathsf{DO}(K_{\mathsf{EC}}, A, c_{\mathsf{EC}}, \tau_{\mathsf{EC}}) \to m/\perp$: Decryption with the key K_{EC} on the encryptment $(c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$ and the AD A.

$\mathsf{EVer}(A, m, K_{\mathsf{EC}}, \tau_{\mathsf{EC}}) \to b$: Verification of a binding tag τ_{EC} for a AD/message pair (A, m) with the key K_{EC}. If $b = 1$, the verification succeeded, otherwise it failed.

Looking forward, a single call to DO is sufficient to *decrypt* and *verify* an encryptment.

Definition 10 (Correctness). EC *is* correct *if all honestly generated encryptments decrypt to the original message and verify with probability 1, i.e.,*

$$\mathsf{Pr}\big[\mathsf{COR}_{\mathsf{EC}}(A, m) = 1\big] = 1$$

where $\mathsf{COR}_{\mathsf{EC}}$ *is described in Fig. 1.*

```
$\mathsf{COR}_{\mathsf{EC}}(A, m)$:
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EK_g}$
$(c_{\mathsf{EC}}, \tau_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, A, m)$
$m' \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, A, c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$
$b \leftarrow \mathsf{EVer}(A, m', K_{\mathsf{EC}}, \tau_{\mathsf{EC}})$
Return $(m = m' \wedge b = 1)$

$\mathsf{S\text{-}COR}_{\mathsf{EC}}(K_{\mathsf{EC}}, A, c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$:
$m \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, A, c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$
$(c'_{\mathsf{EC}}, \tau'_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, A, m)$
$b := (c_{\mathsf{EC}}, \tau_{\mathsf{EC}}) = (c'_{\mathsf{EC}}, \tau'_{\mathsf{EC}})$
Return $b$
```

Fig. 1: COR and S-COR games

Definition 11 (Strong correctness). EC *is* strongly correct *if for all AD, message, key pair there is a unique encryptment* $(c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$ *such that* $m \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, A, c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$*, i.e.,*

$$\mathsf{Pr}\big[\mathsf{S\text{-}COR}_{\mathsf{EC}}(K_{\mathsf{EC}}, A, c_{\mathsf{EC}}, \tau_{\mathsf{EC}}) = 1\big] = 1$$

where $\mathsf{S\text{-}COR}_{\mathsf{EC}}$ *is described in Fig. 1.*

Definition 12 (Confidentiality). EC *is* confidential *if it satisfies one-time real-or-random* (otROR) *security. Given a real encryptment* $(c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$ *under a chosen message/authenticated data pair and an encryptment sampled uniformly at random, they must be indistinguishable to the adversary* $\mathcal{A}$.

Definition 13 (Second-ciphertext unforgeability). EC *satisfies* SCU *if given a single encryptment* $(c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$ *for a chosen* (m, A) *pair under a random key* K_{EC}*,* $\mathcal{A}$ *is unable to find a pair* $(A', c'_{\mathsf{EC}}) \neq (A, c_{\mathsf{EC}})$ *such that* $\mathsf{DO}(K_{\mathsf{EC}}, A', c'_{\mathsf{EC}}, \tau_{\mathsf{EC}}) \neq \bot$.

Definition 14 (Binding security). *Two notions of binding security are required for* EC *schemes;* strong receiver binding $\mathsf{sr\text{-}BIND}$ *and* sender binding $\mathsf{s\text{-}BIND}$. $\mathsf{sr\text{-}BIND}$ *requires that an adversary* $\mathcal{A}$ *cannot find a tag* τ_{EC} *and two distinct triplets* $(A, m, K_{\mathsf{EC}}) \neq (A', m', K'_{\mathsf{EC}})$ *that both pass verification with* τ_{EC}*. The* $\mathsf{sr\text{-}BIND}$ *advantage of* $\mathcal{A}$ *is:*

$$\mathbf{Adv}^{\mathsf{sr\text{-}bind}}_{\mathsf{EC}}(\mathcal{A}) = \mathsf{Pr}\big[\mathsf{sr\text{-}BIND}^{\mathcal{A}}_{\mathsf{EC}} = \mathsf{true}\big]$$

where $\mathsf{sr\text{-}BIND}^{\mathcal{A}}_{\mathsf{EC}}$ *is described in Fig. 2.*
$\mathsf{s\text{-}BIND}$ *ensures that an adversary cannot generate a key, associated data, encryptment tuple* $(K_{\mathsf{EC}}, A, c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$ *that decrypts into a message* m *but fails verification. The advantage of* $\mathcal{A}$ *is:*

$$\mathbf{Adv}^{\mathsf{s\text{-}bind}}_{\mathsf{EC}}(\mathcal{A}) = \mathsf{Pr}\big[\mathsf{s\text{-}BIND}^{\mathcal{A}}_{\mathsf{EC}} = \mathsf{true}\big]$$

where $\mathsf{s\text{-}BIND}^{\mathcal{A}}_{\mathsf{EC}}$ *is described in Fig. 2.*

Definition 15 (Integrity). EC *satisfies* otCTXT *(one-time ciphertext integrity) if an adversary* $\mathcal{A}$ *is unable to forge an encryptment. I.e., given an encryptment* $(c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$ *for chosen* (m, A) *under the key* K_{EC}*,* $\mathcal{A}$ *cannot find a valid tuple* $(A', c'_{\mathsf{EC}}, \tau'_{\mathsf{EC}}) \neq (A, c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$ *that decrypts correctly.*

```
sr-BIND$^{\mathcal{A}}_{\mathsf{EC}}$:
$(V_1, V_2, \tau_{\mathsf{EC}}) \leftarrow\$ \mathcal{A}$
$(A, m, K_{\mathsf{EC}}) := V_1$
$(A', m', K'_{\mathsf{EC}}) := V_2$
$b \leftarrow \mathsf{EVer}(A, m, K_{\mathsf{EC}}, \tau_{\mathsf{EC}})$
$b' \leftarrow \mathsf{EVer}(A', m', K'_{\mathsf{EC}}, \tau_{\mathsf{EC}})$
If $V_1 = V_2$ then
  Return false
Return $b = b' = 1$
```

```
s-BIND$^{\mathcal{A}}_{\mathsf{EC}}$:
$(K_{\mathsf{EC}}, A, c_{\mathsf{EC}}, \tau_{\mathsf{EC}}) \leftarrow\$ \mathcal{A}$
$m' \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, A, c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$
If $m' = \perp$ then
  Return false
$b \leftarrow \mathsf{EVer}(A, m', K_{\mathsf{EC}}, \tau_{\mathsf{EC}})$
If $b = 0$ then
  Return true
Return false
```

Fig. 2: sr-BIND and s-BIND games

3 Flaws in DiSE

The DiSE construction by Agrawal et al. [2] is an instance of TSE (Sect. 2.3) combining a DPRF (Sect. 2.2), a pseudorandom generator PRG and a commitment scheme Com (Def. B.18). We outline the construction here and refer to the original publication for full details.

3.1 DiSE and DPRFs

DiSE.Setup runs DPRF.Setup and $\mathsf{Setup}_{\mathsf{com}}$ and outputs the DPRF key shares $\{sk_i\}_{i=1}^{n}$, as well as the respective public parameters $pp_{\mathsf{DPRF}}, pp_{\mathsf{com}}$ bundled together. The key shares are computed with Shamir's Secret Sharing [38] from a secret s, t shares can be used to recompute the secret s.

The party j initiating DiSE.DistEnc to encrypt m with a set of parties S first computes $\alpha \leftarrow \mathsf{Commit}(m, pp_{\mathsf{com}}; \rho)$ with fresh coins ρ. Then each party $i \in S$ (including j) runs $z_i \leftarrow \mathsf{Eval}(sk_i, j\|\alpha, pp_{\mathsf{DPRF}})$ and sends z_i to j. Finally, j computes $z \leftarrow \mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp_{\mathsf{DPRF}})$ and outputs the ciphertext tuple $(j, \alpha, \mathsf{PRG}(z) \oplus (m\|\rho))$. The underlying PRF of the DPRF is $f_s(x) = \mathcal{H}(x)^s$.

DiSE.DistDec of a ciphertext tuple (j, α, c) proceeds analogously, except that the decrypting party extracts j and α from the ciphertext tuple, uses them as input to a DPRF evaluation. The result z of the latter is then used to decrypt c as $m\|\rho := \mathsf{PRG}(z) \oplus c$ and finally verify that $\alpha = \mathsf{Commit}(m, pp_{\mathsf{Com}}; \rho)$ prior to releasing m.

DiSE can also be used in an "asymmetric" manner where the encryption (and/or the decryption) is executed by a single party, without using of the distributed protocol. This can be achieved by storing the secret s used to compute the DPRF key shares. This secret can then be used to encrypt/decrypt without any interaction.

In their paper, Agrawal et al. [2] put forward multiple DPRFs, with two different security guarantees; *secure* DPRF (i.e. consistent and pseudorandom) and *strongly-secure* DPRF (i.e. consistent, pseudorandom and correct). In this section, we revisit the security of the latter DPRFs. The proposed *strongly-secure* DPRF, called $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$, uses NIZK proofs in each evaluation to prove that each party has computed the z honestly alongside trapdoor commitments. During Combine, each proof is verified; if all the proofs are valid, the output of Combine must be correct as well.

Agrawal et al. give two concrete instantiations of the generic $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$, each with a different NIZK construction and distinct properties: $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pub}}$ and $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri}}$. The former is publicly verifiable, i.e., the verification only takes public parameters. Given the inputs/outputs, a third-party can thus verify whether or not the output was computed honestly. The latter is privately verifiable, i.e., not all NIZK parameters are public. The constructions are shown in Fig. 3. We show that both these instantiations contain mistakes. The first one is present in the generic $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$, and thus also affects the two concrete instantiations, letting an attacker break the correctness of $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$. The second one, presumably a typo, is in the privately verifiable $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri}}$, where the used NIZK proof is broken (all NIZK evaluations fail to verify). $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pub}}$ uses Pedersen commitments [32] and Schnorr-style proofs [37] as NIZK proof. Only the latter is required by $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri}}$.

Generic strongly-secure DPRF ($\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$)

Let $G = \langle g \rangle$ be a multiplicative cyclic group of prime order p in which the DDH assumption holds, $\mathcal{H} : \{0,1\}^* \to G$ and $\mathcal{H}' : \{0,1\}^* \to \mathbb{Z}_p$ be two hash functions modeled as random oracles. Let SSS be Shamir's secret sharing scheme (Def. B.17), TDC $:= (\mathsf{Setup}_{\mathsf{com}}, \mathsf{Commit})$ be a trapdoor commitment scheme (Def. B.19) and NIZK $:= (\mathsf{Prove}^{\mathcal{H}'}, \mathsf{Verify}^{\mathcal{H}'})$ be a simulation-sound NIZK proof system (Def B.20).

- $\mathsf{Setup}(1^k, n, t) \to (\{sk_i\}_{i=1}^n, pp_{\mathsf{DPRF}})$. Sample $s \leftarrow\$ \mathbb{Z}_p$ and get $\{s_i\}_{i=1}^n \leftarrow \mathsf{SSS}(n,t,p,s)$. Run $\mathsf{Setup}_{\mathsf{com}}(1^k)$ to get pp_{com}. Compute a commitment $\gamma_i \leftarrow \mathsf{Commit}(s_i, pp_{\mathsf{com}}; r_i)$ by picking r_i at random. Set $pp_{\mathsf{DPRF}} := (p, g, G, \mathcal{H}, \mathcal{H}', \gamma_1, \ldots, \gamma_n, pp_{\mathsf{com}})$, $sk_i := (s_i, r_i)$ and give sk_i to party i for $i \in [n]$.
- $\mathsf{Eval}(sk_i, x, pp_{\mathsf{DPRF}}) \to z_i$. Compute $w := \mathcal{H}(x)$ and $h_i := w^{s_i}$. Run $\mathsf{Prove}^{\mathcal{H}'}$ with the statement $\mathsf{stmt}_i : \{\exists s, r \text{ s.t. } h_i = w^s \wedge \gamma_i = \mathsf{Commit}(s, pp_{\mathsf{com}}; r)\}$ and witness (s_i, r_i) to obtain a proof π_i. Output $((w, h_i), \pi_i)$.
- $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp_{\mathsf{DPRF}}) \to z/\perp$. If $|S| < t$, output $\perp$. Else, parse z_i as $((w, h_i), \pi_i)$ and check if $\mathsf{Verify}^{\mathcal{H}'}(\mathsf{stmt}_i, \pi_i) = 1$ for all $i \in S$. If check fails for any i, output $\perp$. Else, output $\prod_{i \in S} h_i^{\lambda_{0,i,S}}$.

Publicly verifiable DPRF ($\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pub}}$)

- $\mathsf{Setup}_{\mathsf{com}}(1^k)$: Sample a generator $h \leftarrow\$ G$ and output h.
- $\mathsf{Commit}(s, pp_{\mathsf{com}}; r)$: Output $g^s \cdot h^r$.
- $\mathsf{Prove}^{\mathcal{H}'}(\mathsf{stmt}_i, (s_i, r_i))$: Sample $v_i, v_i' \leftarrow\$ \mathbb{Z}_p$ and set $t_i := w^{v_i}$, $t_i' := g^{v_i} \cdot h^{v_i'}$. Compute a hash $c_i \leftarrow \mathcal{H}'(h_i, w, \gamma_i, g, h, t_i, t_i')$ and the values $u_i := v_i - c_i \cdot s_i$ and $u_i' := v_i' - c_i \cdot r_i$. Output the proof (c_i, u_i, u_i').
- $\mathsf{Verify}^{\mathcal{H}'}(\mathsf{stmt}_i, (c_i, u_i, u_i'))$: Compute $t_i := w^{u_i} \cdot h_i^{c_i}$ and $t_i' := g^{u_i} \cdot h^{u_i'} \cdot \gamma_i^{c_i}$. Output 1 if $c_i = \mathcal{H}'(h_i, w, \gamma_i, g, h, t_i, t_i')$. Output 0 otherwise.

Privately verifiable DPRF ($\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri}}$)

- $\mathsf{Setup}(1^k, n, t) \to (\{sk_i\}_{i=1}^n, pp_{\mathsf{DPRF}})$. Sample $s \leftarrow\$ \mathbb{Z}_p$ and get $\{s_i\}_{i=1}^n \leftarrow \mathsf{SSS}(n,t,p,s)$. Set $pp_{\mathsf{com}} := (p, g, G, \mathcal{H}, \mathcal{H}')$ and $sk_i := (s_i, \{g^{s_j}\}_{j=1}^n)$ for $i \in [n]$.[a]
- $\mathsf{Eval}(sk_i, x, pp_{\mathsf{DPRF}}) \to z_i$. Compute $w \leftarrow \mathcal{H}(x)$ and $h_i := w^{s_i}$. Sample $v_i \leftarrow\$ \mathbb{Z}_p$ and set $t_i := g^{v_i}$. Compute a hash $c_i \leftarrow \mathcal{H}'(h_i, w, g^{s_i}, g, t_i)$ and the value $u_i := v_i - c_i \cdot s_i$. Set (c_i, u_i) as the proof π_i. Output $((w, h_i), \pi_i)$.
- $\mathsf{Verify}^{\mathcal{H}'}(\mathsf{stmt}_i, (c_i, u_i))$: Compute $t_i := w^{u_i} \cdot h_i^{c_i}$. Output 1 if $c_i = \mathcal{H}'(h_i, w, g^{s_i}, g, t_i)$. Output 0 otherwise.

[a] We include the commitments in the key shares similarly to the latest version of the DiSE paper [4]

Fig. 3: DPRF constructions by Agrawal et al. [2]. The fixed generic $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$ and fixed privately verifiable $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri}}$ constructions are obtained by replacing the gray boxes with the content of Figures 4 and 5, respectively.

3.2 Attack on the generic construction $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$

The generic DPRF called $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$ (see Fig. 3) suffers from a flaw in the application of the NIZK proof. This allows a malicious attacker to break the correctness property with a probability of 1. In short, the value w used in the computation of the proof is not validated during the verification. An adversary may output a corrupt value w, such that the NIZK proof verifies but an *incorrect* value is output by Combine nonetheless.

For example, say an adversary outputs the following partial evaluation with a corrupt w: $((w^\star, h_i^\star), \pi_i)$ where $w^\star \neq \mathcal{H}(x)$, $h_i^\star = {w^\star}^{s_i}$. The proof π_i is created by building the statement $\mathsf{stmt}_i : \{\exists\, s, r \text{ s.t. } h_i^\star = {w^\star}^s \wedge \gamma_i = \mathsf{Commit}(s, pp_{\mathsf{com}}; r)\}$ and witnessing (s_i, r_i). One can see that (s, r) are verified in Combine using the public parameter γ_i; thus an attacker must use a valid share. However, the validity of $h_i^\star$ involves an extra parameter, $w^\star$, which is *not* verified. In other words, outputting a corrupt w, but computing h_i with a valid share is enough to pass the verification of the proof and output an invalid value at the end of Combine. This breaks the *correctness* property of $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$.

The same issue is present in $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pub}}$ and (presumably, see below) $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri}}$. A TSE scheme instantiated with a DPRF that does not satisfy correctness fails to satisfy the *strongly secure* definition. As indicated in Sect. 2.3, this may lead to forgeries during the distributed decryption and can corrupt the distributed encryption, rendering ciphertexts "undecryptable".

Definition 16. *Agrawal et al. [2] parses the evaluations as* $((w, h_i), (c_i, u_i))$*, it should be noted that the same* w *is obtained from all the evaluations. This could be seen as suggesting implicitly that the implementations verify the correctness of* w*. However, with no explicit check present, implementations of the protocol faithfully following the original pseudocode would be vulnerable.*

3.3 Flaw in the concrete construction $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri}}$

A closer scrutiny of $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri}}$ (see Fig. 3) reveals another issue: the generation and the verification of the NIZK proof are incompatible. The proof, as defined in Eval, computes $\pi_i = (c_i, u_i) = (\mathcal{H}'(h_i, w, g^{s_i}, g, t_i),\ v_i - c_i \cdot s_i)$ where $v_i \leftarrow\$\ \mathbb{Z}_p$ and $t_i := g^{v_i}$. Later in Combine, the proof is verified by computing $t_i := w^{u_i} \cdot h_i^{c_i} = w^{v_i - c_i \cdot s_i} \cdot w^{c_i \cdot s_i} = w^{v_i}$. Seeing that the variable $t_i = w^{v_i}$ computed here is statistically independent of $t_i = g^{v_i}$ computed in Eval, the proof is considered invalid with overwhelming probability and virtually every honest DPRF evaluation will be considered invalid and fail.

We believe that this was likely the result of a typo.[3] As the mistake was not spotted, it is safe to assume it has been an inconspicuous one. A likely candidate is an accidental replacement of w by g in Eval, yielding $t_i := g^{v_i}$ instead of $t_i := w^{v_i}$. Indeed, substituting the former term by the latter in Eval lets us validate the proofs in Combine. However, upon analysis of this construction, we show this construction is also broken *even assuming a typo*; see next section.

3.4 A non-fix of $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri}}$

We turn our attention to a variant of $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri}}$ that we believe to be the likeliest intention of Agrawal et al., named $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri^\star}}$. As introduced beforehand, this is the same as $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP\text{-}Pri}}$ except for one small change in Eval highlighted below:

- $\mathsf{Eval}(sk_i, x, pp_{\mathsf{DPRF}}) \to z_i$. Compute $w \leftarrow \mathcal{H}(x)$ and $h_i := w^{s_i}$. Sample $v_i \leftarrow\$\ \mathbb{Z}_p$ and set $t_i := w^{v_i}$. Compute a hash $c_i \leftarrow \mathcal{H}'(h_i, w, g^{s_i}, g, t_i)$ and the value $u_i := v_i - c_i \cdot s_i$. Set (c_i, u_i) as the proof π_i. Output $((w, h_i), \pi_i)$.

While this modification allows to verify the proof and compute the output of the DPRF, the resulting construction fails to satisfy the DPRF correctness property and is not *strongly-secure* as a consequence.

A proof π_i now consists of $(c_i = \mathcal{H}'(h_i, w, g^{s_i}, g, t_i), u_i = v_i - c_i \cdot s_i)$ where $v_i \leftarrow\$\ \mathbb{Z}_p$ and $t_i := w^{v_i}$. The proof is verified by recovering $t_i = w^{u_i} \cdot h_i^{c_i} = w^{v_i - c_i \cdot s_i} \cdot w^{c_i \cdot s_i} = w^{v_i}$ and checking that c_i is

[3] We were not able to find an updated version where this mistake is corrected, it appears to be present in all 3 versions [2–4] published so far.

equal to the hash. Unlike before, the same t_i is recovered, thus the proof verifies. However, one can see that this proof does not enforce the usage of the correct secret s_i as no public parameters are involved during the verification. In fact, the proof only enforces that the exponent used in computing $h_i = w^{s_i}$ is equal to the one used to compute $u_i = v_i - c_i \cdot s_i$. An attacker can therefore break the correctness of this DPRF by using any value instead of the true s_i.

3.5 Other vulnerable constructions

Due to the problem in the generic DPRF described above, some recent papers [5,11,27] suffer from a similar issue in their modified distributed pseudorandom functions. For instance, in the Amortized Threshold Symmetric-key Encryption scheme by Christodorescu et al. [11], the issue is present in their strongly-secure FTKD construction. The notion of correctness can be attacked using a similar method as the one described in Sect. 3.2.

4 DiAE: DiSE reloaded

In this section we first fix the DPRF constructions and then propose our new DiAE construction.

4.1 Fixing the DPRFs

We start by fixing the flaws presented in Section 3.

4.1.1 Generic construction. To fix the problems in the generic $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$ construction, the variable w must be validated across all output shares. This can be ensured either by making the "correct" w an input argument to Combine, or by checking inside Combine that the w in every output share is the same prior to releasing the output. We chose the latter for the sake of backwards compatibility, as this allows a black-box reuse of existing implementations of Combine. As the security model of DPRF requires at least one party to behave honestly, at least one w is valid and DPRF evaluations with malicious $w^\star \neq w$ will be detected. However, we cannot detect which parties output a corrupted value. If a corruption is detected, the protocol has to be executed again with a different set of parties. The changes in Combine from $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$ are defined in Fig. 4, we name this construction $\Pi_{\mathsf{DDH\text{-}Gen}}$.

– $\mathsf{Combine}(\{(i, z_i)\}_{i\in S}, pp_{\mathsf{DPRF}}) \to z/\perp$. If $|S| < t$, output $\perp$. Else, parse z_i as $((w_i, h_i), \pi_i)$. Check if $w_i = w_j$ for all $i, j \in S,\ i \neq j$. If check fails for any i, j, output $\perp$. Check if $\mathsf{Verify}^{\mathcal{H}'}(\mathsf{stmt}_i, \pi_i) = 1$ for all $i \in S$. If check fails for any i, output $\perp$. Else, output $\prod_{i\in S} h_i^{\lambda_{0,i,S}}$.

Fig. 4: Updated Combine algorithm of $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$

Theorem 1. $\Pi_{\mathsf{DDH\text{-}Gen}}$ *is a* strongly-secure DPRF *under the DDH assumption in the programmable random oracle model.*

Proof. The proof consists of proving that $\Pi_{\mathsf{DDH\text{-}Gen}}$ is consistent (Lemma 1), pseudorandom (Lemma 2) and correct (Lemma 3). □

Lemma 1 (Consistency). $\Pi_{\mathsf{DDH\text{-}Gen}}$ *satisfies the consistency property.*

Proof. Consistency is straightforward as it is a property of Shamir's Secret Sharing. □

Lemma 2 (Pseudorandomness). $\Pi_{\mathsf{DDH\text{-}Gen}}$ *satisfies the pseudorandomness property.*

Proof. We only provide a short explanation on the additions to DiSE's proof [3, Appendix C.5] to reflect the changes in the construction. Essentially, the proof is the same, except for the addition of an extra step in the challenge queries. Before the verification of the proofs, we add a new step that outputs 0 if any $w_i \neq \mathcal{H}(x^\star)$ for $i \in [u]$ where w_i is provided by the adversary. While it differs from the actual construction, this is allowed because $x^\star$ is a challenge query parameter that we use to compute the honest evaluations; thus the two methods are equivalent. □

Lemma 3 (Correctness). $\Pi_{\mathsf{DDH\text{-}Gen}}$ *satisfies the correctness property.*

Proof. The proof is adapted from DiSE's [3, Appendix C.5]. To prove correctness of $\Pi_{\mathsf{DDH\text{-}Gen}}$, we will exploit the extractability property of proofs and the binding property of commitments. We describe the last step of the $\mathsf{Correctness}^{\mathcal{A}}_{\mathsf{DPRF}}(1^k)$ game in detail (the substitutions of generic pseudocode by computations from the construction are highlighted in blue), the first three steps are the same as $\mathsf{PseudoRand}'_{\mathcal{A}}(b)$ (Lemma 2):

4. On the challenge query $(\mathsf{Challenge}, x^\star, S, ((w_1, h^\star_1), \pi_1), \ldots, ((w_u, h^\star_u), \pi_u))$ for $u \leq \ell$ (without loss of generality assume that $S \cap C = [u]$):
   (a) If $|S| < t$ or any $\mathcal{H}(x^\star) \neq w_i,\ i \in [u]$ or any $\pi_1, \ldots, \pi_u$ do not verify, output 1.
   (b) Else, compute $h_j := \mathcal{H}(x^\star)^{s_j}$ for $j \in S$, $h^\star_i := \mathcal{H}(x^\star)^{s_i}$ for $i \in S \setminus C$, $z := \prod_{i \in S} h_i^{\lambda_{0,i,S}}$ and $z^\star := \prod_{i \in S} h_i^{\star\lambda_{0,i,S}}$. If $z^\star = z$, output 1.
   (c) Else, output 0.

Suppose there exists an adversary $\mathcal{A}$ s.t. the correctness game reaches the very last step (4c), leading the challenger to output 0, with non-negligible probability. We will show that this leads to a contradiction. Towards this, we define a few intermediate hybrid games. In the first hybrid game, the hash function $\mathcal{H}'$ is replaced with the simulator $\mathcal{S}_1$ guaranteed by the zero-knowledge property of NIZK.
In the second hybrid, instead of producing a zero in the very last step (4c), the challenger:

– finds an $i^\star \in [u]$ s.t. $h^\star_{i^\star} \neq h_{i^\star}$ (such an $i^\star$ exists because $z^\star \neq z$);
– invokes the extractor $\mathcal{E}$ guaranteed by the argument of knowledge property on the adversary with inputs $(\mathsf{stmt}_{i^\star}, \pi_{i^\star}, Q)$ (Q is the list of queries made to $\mathcal{S}_1$ and their responses); and,
– outputs whatever the extractor does.

If the first hybrid game reaches the very last step, it means that all $w_i,\ i \in [u]$ and the proofs provided by $\mathcal{A}$ were valid. Thus, if the game outputs 0 with non-negligible probability, the challenger will output a witness $(s'_{i^\star}, r'_{i^\star})$ for $(\mathsf{stmt}_{\mathsf{i}^\star}$ with non-negligible probability.

In the last hybrid, the challenger outputs $(s_{i^\star}, r_{i^\star})$ along with the extracted witness. Since $h^\star_{i^\star} \neq h_{i^\star}$ and $w_{i^\star} = \mathcal{H}(x^\star)$, $s'_{i^\star} \neq s_{i^\star}$. Therefore, the challenger finds two distinct pairs $(s_{i^\star}, r_{i^\star})$ and $(s'_{i^\star}, r'_{i^\star})$ that produce the same commitment (with non-negligible probability). This breaks the binding property of TDC. □

4.1.2 Privately verifiable DPRF. For the privately verifiable DPRF, we start from the generic strongly-secure $\Pi_{\mathsf{DDH\text{-}Gen}}$ (Fig. 4), use a trapdoor-less commitment and apply Schnorr style proofs [37] to prove the equality of discrete logarithms. This leads to a proof system similar to those used and proven secure in multiple papers [10, 18, 23]. We call this construction $\Pi_{\mathsf{DDH\text{-}Pri}}$.

The modifications needed to obtain the updated privately verifiable DPRF from $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$ are shown in Fig. 5.

– $\mathsf{Setup}(1^k, n, t) \to (\{sk_i\}_{i=1}^n, pp_{\mathsf{DPRF}})$. Sample $s \leftarrow\$ \mathbb{Z}_p$ and get $\{s_i\}_{i=1}^n \leftarrow \mathsf{SSS}(n, t, p, s)$. Set $pp_{\mathsf{com}} := (p, g, G, \mathcal{H}, \mathcal{H}'), sk_i := (s_i, \{pk_j\}_{j=1}^n)$ where $pk_j = g^{s_j}$.
– $\mathsf{Eval}(sk_i, x, pp_{\mathsf{DPRF}}) \to z_i$. Compute $w_i \leftarrow \mathcal{H}(x)$ and $h_i := w_i^{s_i}$. Sample $v_i \leftarrow\$ \mathbb{Z}_p$, set $t_i := g^{v_i}$ and $t'_i := w_i^{v_i}$. Compute a hash $c_i \leftarrow \mathcal{H}'(h_i, w_i, pk_i, g, t_i, t'_i)$ and the value $u_i := v_i - c_i \cdot s_i$. Set (c_i, u_i) as the proof π_i. Output $((w_i, h_i), \pi_i)$.
– $\mathsf{Verify}^{\mathcal{H}'}(\mathsf{stmt}_i, (c_i, u_i))$: Compute $t_i := g^{u_i} \cdot pk_i^{c_i}$ and $t'_i := w_i^{u_i} \cdot h_i^{c_i}$. Output 1 if $c_i = \mathcal{H}'(h_i, w_i, pk_i, g, t_i, t'_i)$. Output 0 otherwise.

Fig. 5: Updated privately verifiable DPRF ($\Pi_{\mathsf{DDH\text{-}Pri}}$)

Theorem 2. $\Pi_{\mathsf{DDH\text{-}Pri}}$ *is a* strongly-secure DPRF *under the DDH assumption in the programmable random oracle model.*

Proof. The proof is obtained by combining Lemmas 4 to 6. □

Lemma 4 (Consistency). $\Pi_{\mathsf{DDH\text{-}Pri}}$ *satisfies the consistency property.*

Proof. Consistency is straightforward as it is a property of Shamir's Secret Sharing. □

Lemma 5 (Pseudorandomness). $\Pi_{\mathsf{DDH\text{-}Pri}}$ *satisfies the pseudorandomness property.*

Proof. To prove pseudorandomness of $\Pi_{\mathsf{DDH\text{-}Pri}}$, we will exploit the extractability property of proofs and the binding property of commitments. We first describe the pseudorandomness game in detail.

Let $\mathsf{PseudoRand}''_{\mathcal{A}}(b)$ (shorthand for $\mathsf{PseudoRand}^{\mathcal{A}}_{\Pi_{\mathsf{DDH\text{-}Pri}}}(b)$, Def. 2) be the following:

1. Let G be a cyclic group of order p and g a generator of G. Sample $s \leftarrow\$ \mathbb{Z}_p$ and get $\{s_i\}_{i=1}^n \leftarrow \mathsf{SSS}(n, t, p, s)$. Send public parameters $pp_{\mathsf{DPRF}} := (p, g, G, \mathcal{H}, \mathcal{H}')$ to $\mathcal{A}$.
2. Get the set of corrupt parties C from $\mathcal{A}$. WLOG assume that $C = \{1, \ldots, \ell\}$. Set the corruption gap as $\delta := t - |C|$. Send the corresponding secret keys $\{(s_i, \{pk_j\}_{j=1}^n)\}_{i \in C}$ to $\mathcal{A}$ where $pk_j := g^{s_j}$ is a commitment to the secret j.
3. On an evaluation query (Eval, x, i) for an honest i, compute $w_i \leftarrow \mathcal{H}(x)$ and $h_i := w_i^{s_i}$. Run $\mathsf{Prove}^{\mathcal{H}'}$ with the statement $\mathsf{stmt_i} : \{\exists s \text{ s.t. } h_i = w_i^s \wedge pk_i = g^s\}$ and witness s_i to obtain the proof π_i. Return $((w_i, h_i), \pi_i)$ to $\mathcal{A}$.
4. On the challenge query $(\mathsf{Challenge}, x^\star, S, ((w_1, h_1^\star), \pi_1), \ldots, ((w_u, h_u^\star), \pi_u))$ for $u \leq \ell$ (without loss of generality assume that $S \cap C = [u]$):
   (a) If $\mathcal{A}$ has already made at least δ queries of the form $(\mathsf{Eval}, x^\star, *)$, then output 0 and stop.
   (b) Otherwise do as follows:
      i. If $w_i \neq \mathcal{H}(x^\star)$ for any $i \in [u]$ or $\mathsf{Verify}^{\mathcal{H}'}(\mathsf{stmt}_i, \pi_i) \neq 1$ for any $i \in [u]$, output 0 and stop.
      ii. Set $h_i^\star := \mathcal{H}(x^\star)^{s_i}$ for $i \in S \setminus C$.
      iii. If $b = 0$, compute $z := \prod_{i \in S} {h_i}^{\star\lambda_{0,i,S}}$. Otherwise, choose a random $z \leftarrow\$ G$.
   (c) Send z to $\mathcal{A}$.
5. Continue answering evaluation queries as before, but if $\mathcal{A}$ makes a query of the form $(\mathsf{Eval}, x^\star, i)$ for some $i \in [n] \setminus C$ and i is the δ-th party it contacted, then output 0 and stop.
6. Receive a guess b' from $\mathcal{A}$; output b'.

First hybrid. Define a hybrid $\mathsf{Hyb}^{(\mathsf{zk})}_{\mathcal{A}}(b)$ which is similar to $\mathsf{PseudoRand}''_{\mathcal{A}}(b)$ except that the real proofs π_i in Step 3 and 5 are replaced with simulated proofs; thus the witness s_i is not needed. $\mathsf{PseudoRand}''_{\mathcal{A}}(b)$ is indistinguishable from $\mathsf{Hyb}^{(\mathsf{zk})}_{\mathcal{A}}(b)$ for any PPT adversary $\mathcal{A}$ where $b \in \{0, 1\}$ due to the zero-knowledge property of NIZK (Def. B.20).

Second hybrid. Define a hybrid $\mathsf{Hyb}^{(\mathsf{com})}_{\mathcal{A}}(b)$ which is similar to the first hybrid, except in Step 2:

- Set $pk_j := g^{s_j}$ for $j \in C$ and $pk_{j'} := g^{r_{j'}}$ for $j' \in [n] \setminus C$ where $r_{j'} \leftarrow\$ \mathbb{Z}_p$. I.e., output a real commitment for the corrupt parties and a commitment to a random value for the honest parties.

One can see that $\mathsf{Hyb}_{\mathcal{A}}^{(\mathsf{zk})}(b)$ is indistinguishable from $\mathsf{Hyb}_{\mathcal{A}}^{(\mathsf{com})}(b)$ for any PPT adversary $\mathcal{A}$ where $b \in \{0,1\}$ under the DLOG problem and the information-theoretic security of Shamir's Secret Sharing.

Reduction. We now show that if a PPT adversary $\mathcal{A}$ can distinguish between the $b = 0$ and $b = 1$ cases in the hybrid game above with a non-negligible probability, then one can build a PPT adversary $\mathcal{A}'$ to break the pseudorandomness property of the scheme $\Pi_{\mathsf{DDH\text{-}DP}}$ (DiSE [3, Section 8.1]), which would be in contradiction to DiSE [3, Theorem 8.1]. Let Chal denote the challenger in $\mathsf{PseudoRand}_{\mathcal{A}'}(b)$ shown in DiSE [3, Appendix C.4]. We construct $\mathcal{A}'$ as follows:

- Get group parameters $(p, g, G, \mathcal{H})$ from Chal. Send $(p, g, G, \mathcal{H}, \mathcal{H}')$ to $\mathcal{A}$.
- Get the set of corrupt parties $C = \{1, \ldots, \ell\}$ from $\mathcal{A}$. Pass it along to Chal and get shares $\{s_i\}_{i=1}^{\ell}$. Set the corruption gap as $\delta := t - |C|$. Set $pk_j := g^{s_j}$ for $j \in C$ and $pk_{j'} := g^{r_{j'}}$ for $j' \in [n] \setminus C$ where $r_{j'} \leftarrow\$ \mathbb{Z}_p$. Send $\{(s_1, \{pk_j\}_{j=1}^n), \ldots, (s_\ell, \{pk_j\}_{j=1}^n)\}$ to $\mathcal{A}$.
- On an evaluation query (Eval, x, i) for an honest i from $\mathcal{A}$, send the same query to Chal and get back h_i. Compute $w_i \leftarrow \mathcal{H}(x)$ and a simulated proof π_i'. Return $((w_i, h_i), \pi_i')$ to $\mathcal{A}$.
- On the challenge query $(\mathsf{Challenge}, x^\star, S, ((w_1, h_1^\star), \pi_1), \ldots, ((w_u, h_u^\star), \pi_u))$ from $\mathcal{A}$, first verify if $\mathcal{A}$ has already made at least δ queries of the form $(\mathsf{Eval}, x^\star, *)$ or $w_i \neq \mathcal{H}(x^\star)$ for any $i \in [u]$ or one of the proofs does not verify, then output 0 and stop. Query Chal with $(\mathsf{Challenge}, x^\star, S, h_1^\star, \ldots, h_u^\star)$ and get back z. Return z to $\mathcal{A}$.
- Continue answering evaluation queries as before, but if $\mathcal{A}$ makes a query of the form $(\mathsf{Eval}, x^\star, i)$ for some $i \in [n] \setminus C$ and i is the δ-th party it contacted, then output 0 and stop.
- Receive a guess b' from $\mathcal{A}$; output b'.

Observe that for $b \in \{0,1\}$, when $\mathcal{A}'$ is in the game $\mathsf{PseudoRand}_{\mathcal{A}'}(b)$, the view of $\mathcal{A}$ in the reduction is exactly the same as in $\mathsf{Hyb}_{\mathcal{A}}^{(\mathsf{com})}(b)$. Thus, if the output of $\mathsf{Hyb}_{\mathcal{A}}^{(\mathsf{com})}(0)$ is computationally distinguishable from $\mathsf{Hyb}_{\mathcal{A}}^{(\mathsf{com})}(1)$, the output of $\mathsf{PseudoRand}_{\mathcal{A}'}(0)$ would also be from $\mathsf{PseudoRand}_{\mathcal{A}'}(1)$. □

Lemma 6 (Correctness). $\Pi_{\mathsf{DDH\text{-}Pri}}$ *satisfies the correctness property.*

Proof. The proof below is based on the generic DPRF's proof, but adapted to our privately verifiable DPRF.

To prove correctness of $\Pi_{\mathsf{DDH\text{-}Pri}}$, we will exploit the extractability property of proofs. We describe the last step of the $\mathsf{Correctness}_{\mathsf{DPRF}}^{\mathcal{A}}(1^k)$ game in detail, the first three steps are the same as $\mathsf{PseudoRand}''_{\mathcal{A}}(b)$ (Lemma 5):

4. On the challenge query $(\mathsf{Challenge}, x^\star, S, ((w_1, h_1^\star), \pi_1), \ldots, ((w_u, h_u^\star), \pi_u))$ for $u \leq \ell$ (without loss of generality assume that $S \cap C = [u]$):
   (a) If $|S| < t$ or any $\mathcal{H}(x^\star) \neq w_i$, $i \in [u]$ or any $\pi_1, \ldots, \pi_u$ do not verify, output 1.
   (b) Else, compute $h_j := \mathcal{H}(x^\star)^{s_j}$ for $j \in S$, $h_i^\star := \mathcal{H}(x^\star)^{s_i}$ for $i \in S \setminus C$, $z := \prod_{i \in S} h_i^{\lambda_{0,i,S}}$ and $z^\star := \prod_{i \in S} h_i^{\star\lambda_{0,i,S}}$. If $z^\star = z$, output 1.
   (c) Else, output 0.

Suppose there exists an adversary $\mathcal{A}$ s.t. the correctness game reaches the very last step (4c), leading the challenger to output 0, with non-negligible probability. We will show that this leads to a contradiction. Towards this, we define a few intermediate hybrid games. In the first hybrid game, the hash function $\mathcal{H}'$ is replaced with the simulator $\mathcal{S}_1$ guaranteed by the zero-knowledge property of NIZK.
In the second hybrid, instead of producing a zero in the very last step (4c), the challenger:

– finds an $i^\star \in [u]$ s.t. $h^\star_{i^\star} \neq h_{i^\star}$ (such an $i^\star$ exists because $z^\star \neq z$);
– invokes the extractor $\mathcal{E}$ guaranteed by the argument of knowledge property on the adversary with inputs $(\mathsf{stmt}_{i^\star}, \pi_{i^\star}, Q)$ (Q is the list of queries made to $\mathcal{S}_1$ and their responses); and,
– outputs whatever the extractor does.

If the first hybrid game reaches the very last step, it means that all $w_i = \mathcal{H}(x^\star)$ and proofs π_i were valid for $i \in [u]$. Thus, if the game outputs 0 with non-negligible probability, the challenger will output a witness $s'_{i^\star}$ for $\mathsf{stmt}_{\mathsf{i}^\star}$ with non-negligible probability.

In the last hybrid, the challenger outputs $s_{i^\star}$ along with the extracted witness. Since $h^\star_{i^\star} \neq h_{i^\star}$ and $w_{i^\star} = \mathcal{H}(x^\star)$, $s'_{i^\star} \neq s_{i^\star}$. Therefore, the challenger finds two distinct values $s_{i^\star}$ and $s'_{i^\star}$ that produce the same commitment. This leads to a contradiction as the commitment is perfectly binding ($g^{s_{i^\star}} = g^{s'_{i^\star}} \iff s_{i^\star} = s'_{i^\star}$). □

Remark 1. It is crucial to instantiate $\mathcal{H}$ with a proper hash function (e.g., [17] for elliptic curves) and to verify that all w_i are equal. For instance, a bad instantiation on elliptic curves could implement $\mathcal{H}(x)$ as $\mathsf{PRG}(x) \cdot G$ and not verify the correctness of w_i. Those two mistakes, could lead to a powerful attack that breaks the pseudorandomness of DPRF; and thus the security of DiSE (and DiAE).

A malicious adversary can forge an evaluation to output an arbitrary value at the end of combine with probability $1/\binom{n-2}{t-2}$ (guessing which parties are contacted). This is done by leveraging the malleability of the commitments ($PK_i = s_i \cdot G$) and the discrete log relation of the hash to G. WLOG we assume below that $S = \{1, 2, \ldots, t\}$ (i.e. the first t parties are contacted). The attacker $i = 1$ computes $W^\star_1$, the partial eval $H_1 \coloneqq s_1 \cdot W^\star_1$ and the proof π_1. Both H_1 and π_1 are computed honestly, while $W^\star_1$ is computed maliciously as $W^\star_1 \coloneqq \lambda^{-1}_{0,1,S} \cdot s^{-1}_1 \cdot a \cdot G - \sum_{i=2}^t \lambda^{-1}_{0,1,S} \cdot s^{-1}_1 \cdot \lambda_{0,i,S} \cdot \mathsf{PRG}(x) \cdot PK_i$.

Then, in Combine, the proof verification passes as it was computed honestly and the final output is obtained by the honest initiating party as $\sum_{i=1}^t \lambda_{0,i,S} \cdot H_i = \lambda_{0,1,S} \cdot s_1 \cdot W^\star_1 + \sum_{i=2}^t \lambda_{0,i,S} \cdot s_i \cdot W_i = a \cdot G - \sum_{i=2}^t \lambda_{0,i,S} \cdot \mathsf{PRG}(x) \cdot s_i \cdot G + \sum_{i=2}^t \lambda_{0,i,S} \cdot s_i \cdot \mathsf{PRG}(x) \cdot G = a \cdot G$, which corresponds to the arbitrary output chosen by the adversary.

4.2 From commitment to encryptment

We propose a new construction that we call Distributed Authenticated Encryption (DiAE), see Fig. 6. This construction replaces the combination of commitment and (stream cipher-like) encryption in DiSE by a more specialized primitive called encryptment (Sect. 2.4). The encryptment allows us to replace both the commitment and the encryption in DiSE by a single primitive that provides similar security guarantees. This yields interesting results as it enables our construction to execute the encryption of the message prior to the distributed part of the protocol. As such, implementations can start transmitting ciphertexts immediately after the message encryption. This can also be done in a streaming fashion by encrypting and transmitting the messages block-by-block; thus only a constant-size block must be stored during the encryption. With our construction, once the message has been encrypted and (optionally) transmitted, only a constant-size state, a binding tag, must be stored to execute the end of the distributed encryption.

In short, our construction works as follows. For the distributed encryption, we first sample an encryption key K_{EC} uniformly at random which is used to encrypt the message, resulting in a ciphertext and a binding tag. This tag, is then sent to $t-1$ other parties to obtain the output z of the DPRF based on the tag (and encryption initiator). Finally, z is used with a KDF to encrypt K_{EC}. The distributed decryption is executed by first recovering K_{EC} using the binding tag as the DPRF input. The ciphertext is then decrypted into the message with K_{EC}.

Theorem 3. DiAE *is a (strongly) secure* TSE *if* DPRF *is (strongly) secure.*

Ingredients:
A (strongly) secure $\mathsf{DPRF} := (\mathsf{DPRF.Setup}, \mathsf{DPRF.Eval}, \mathsf{DPRF.Combine})$ (Sect. 2.2).
A KDF (Sect. B.21).
A strongly correct EC scheme $\mathsf{EC} := (\mathsf{EK_g}, \mathsf{EC}, \mathsf{DO})$ (Sect. 2.4).

$\underline{\mathsf{Setup}(1^k, n, t) \to (\{sk_i\}_{i=1}^n, pp)}$:

- Execute $\mathsf{DPRF.Setup}(1^k, n, t)$ to get $(\{sk_i\}_{i=1}^n, pp_{\mathsf{DPRF}})$.
- Set $pp := pp_{\mathsf{DPRF}}$ and output $(\{sk_i\}_{i=1}^n, pp)$.

$\underline{\mathsf{DistEnc}(j, m, S, pp) \to c/\perp}$:

- The encryption initiator j samples an encryption key $K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EK_g}$ and computes the encryptment $(c_{\mathsf{EC}}, \tau_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, j, m)$.
- Sends $x := j \mid\mid \tau_{\mathsf{EC}}$ to all parties $i \in S$. Each party i first verifies that x is of the form $j^\star \mid\mid \tau^\star_{\mathsf{EC}}$ where $j^\star$ matches the identity of the sender, then executes $\mathsf{DPRF.Eval}(sk_i, x, pp_{\mathsf{DPRF}})$ to get z_i and sends it to the encryption initiator.
- Once t evals have been gathered, executes $\mathsf{DPRF.Combine}(\{i, z_i\}_{i\in S}, pp_{\mathsf{DPRF}})$ to get z or $\perp$. Finally, j computes $e := \mathsf{KDF}(z) \oplus K_{\mathsf{EC}}$ and outputs the ciphertext $c := (j, c_{\mathsf{EC}}, \tau_{\mathsf{EC}}, e)$.

$\underline{\mathsf{DistDec}(c, S, pp) \to m/\perp}$:

- The decryption initiator j' first parses the ciphertext c into $(j, c_{\mathsf{EC}}, \tau_{\mathsf{EC}}, e)$.
- Sends $x := j \mid\mid \tau_{\mathsf{EC}}$ to all parties $i \in S$. Each party i first verifies that x is of the form $j^\star \mid\mid \tau^\star_{\mathsf{EC}}$ where $j^\star \in [n]$, then executes $\mathsf{DPRF.Eval}(sk_i, x, pp_{\mathsf{DPRF}})$ to get z_i and sends it to the decryption initiator.
- Once t evals have been gathered, j' executes $\mathsf{DPRF.Combine}(\{i, z_i\}_{i\in S}, pp_{\mathsf{DPRF}})$ to get z or $\perp$. Then, j' computes $K_{\mathsf{EC}} := \mathsf{KDF}(z) \oplus e$ and decrypts the ciphertext to get $m/\perp \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, j, c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$ and output the result.

Fig. 6: DiAE protocol

Proof. The proof is obtained by combining the Lemmas 7 to 12. □

Lemma 7 (Consistency). *DiAE is a consistent TSE scheme.*

Proof. As consistency of TSE requires parties to behave honestly, this property follows easily from the consistency of DPRF. □

Lemma 8 (Correctness). *DiAE is a correct TSE scheme.*

Proof. DiAE is a correct TSE scheme if for all ciphertexts c generated by an honest party j in DistEnc, any honest party j' recovers the original message m or $\perp$ at the end of DistDec. On the contrary, suppose that party j' outputs the message $m^\star \notin \{m, \perp\}$ where $m^\star \leftarrow \mathsf{DO}(K^\star_{\mathsf{EC}}, j', c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$. Note that $K^\star_{\mathsf{EC}}$ used in DistDec may be different than K_{EC} in DistEnc due to corruptions. In other words, there are two distinct tuples $(j', m, K_{\mathsf{EC}}), (j', m^\star, K^\star_{\mathsf{EC}})$ that decrypt under the same tag τ_{EC}. This either breaks the *sender binding*[4] or the *strong receiver binding*[5] security of EC (Definition 14). □

Lemma 9 (Strong correctness). *If DPRF is a strongly-secure DPRF, DiAE is a strongly correct TSE scheme.*

[4] Cannot find an encryptment that decrypts correctly, but for which verification fails.
[5] Cannot find two tuples $(A, m, K_{\mathsf{EC}}), (A', m', K'_{\mathsf{EC}})$ that *verify* successfully under the same tag τ_{EC}.

Proof. DiAE is a correct TSE scheme if for all ciphertexts c generated by an honest party j in $\mathsf{DistEnc}$, any honest party j' recovers the original message m at the end of $\mathsf{DistDec}$ with high probability. On the contrary, suppose that j' recovers $\perp$. If all parties behave honestly, party j' obtains the value z during $\mathsf{DistDec}$ and recovers the key $K^\star_{\mathsf{EC}} := \mathsf{KDF}(z) \oplus e$. As $j, c_{\mathsf{EC}}, \tau_{\mathsf{EC}}$ are the same parameters used during the encryption, the decryption fails iff $K^\star_{\mathsf{EC}} \neq K_{\mathsf{EC}} \iff \mathsf{KDF}(z^\star) \neq \mathsf{KDF}(z) \iff z^\star \neq z$ due to the *correctness* of EC (Def. 10). This leads to a contradiction as it breaks the *correctness* property of DPRF (Def. 3). The case where the parties behave maliciously during $\mathsf{DistDec}$ is covered by Lemma 8. ◻

Lemma 10 (Message private). *DiAE is a message private TSE scheme.*

Proof. We defer the complete proof to Appendix A.1 and provide only a sketch of the proof here. In the message privacy game (Def. 7), the adversary receives a ciphertext $c^\star := (j^\star, c^\star_{\mathsf{EC}}, \tau^\star_{\mathsf{EC}}, e^\star)$ where $j^\star$ is an honest party, $(c^\star_{\mathsf{EC}}, \tau^\star_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K^\star_{\mathsf{EC}}, j^\star, m_b)$ and $e^\star := \mathsf{KDF}(z^\star) \oplus K^\star_{\mathsf{EC}}$; the adversary wins by distinguishing the bit b. The proof consists of proving that the output of the encryptment $(c^\star_{\mathsf{EC}}, \tau^\star_{\mathsf{EC}})$ and the output of the DPRF $(z^\star)$ appear pseudorandom to the adversary. The former is straightforward due to the properties of encryptment. For the latter, we are left to prove that the adversary never learns the output of the DPRF on the input $j^\star || \tau^\star_{\mathsf{EC}}$ as otherwise the output appears pseudorandom to PPT adversaries. The adversary can initiate an encryption on the input $j || \tau^\star_{\mathsf{EC}}$ for any corrupt $j \neq j^\star$ since $j^\star$ is honest. The adversary can also request the honest party $j^\star$ to initiate the encryption on $j^\star || \tau_{\mathsf{EC}}$, but due to the binding property of EC since $j^\star$ is honest, $\tau_{\mathsf{EC}} \neq \tau^\star_{\mathsf{EC}}$. Thus, as the adversary cannot learn the DPRF output, the security relies on the pseudorandomness of DPRF. ◻

Lemma 11 (Authenticity). *If DPRF is a* secure DPRF, *DiAE satisfies authenticity.*

Proof. We only provide a proof sketch and defer the complete proof to Appendix A.2. A successful forgery output by the adversary consists of a $r+1$ tuple $((j_1, S_1, c_1), \ldots, (j_{r+1}, S_{r+1}, c_{r+1}))$ where $c_\ell = (j_\ell, c_{\ell_{\mathsf{EC}}}, \tau_{\ell_{\mathsf{EC}}}, e_\ell)$. Suppose that there exists a $u \neq v$ such that $j_u = j_v$, $\tau_{u_{\mathsf{EC}}} = \tau_{v_{\mathsf{EC}}}$ and $c_u \neq c_v$. We can prove the contrary with the binding property (Def. 14) and the strong correctness (Def. 10) of EC. As such, the ciphertexts output by the adversary all have different (j, τ_{EC}). As the DPRF inputs are unique and the number of DPRF outputs learned by the adversary is counted (the value r), the adversary wins the game by computing a DPRF output without contacting at least one honest party. This breaks the pseudorandomness of DPRF. ◻

Lemma 12 (Strong authenticity). *If DPRF is* strongly-secure, *DiAE satisfies strong authenticity.*

Proof. We only provide a proof sketch here as proving strong authenticity relies heavily on the correctness of the DPRF and only slightly on the TSE protocol itself; the proof is therefore very similar to DiSE's [3, Appendix C.3] proof. For authenticity, we relies on the consistency property of the DPRF. This was possible because all the parties are required to behave honestly during the verification of the forgery. However, for strong authenticity, this requirement is removed and the adversary can behave in a malicious way; thus consistency is not sufficient. DiSE's proof relies on both consistency and *correctness*. Thanks to correctness, we can assume with high probability that we either recover the same z_u and z_v (as $u \neq v$, $j_u = j_v$ and $\tau_{u_{\mathsf{EC}}} = \tau_{v_{\mathsf{EC}}}$, the same input is given to the DPRF) or $\perp$. In the former case, the proof is similar to the one described above, while in the latter case, $\mathsf{Auth}^{\mathcal{A}}_{\mathsf{DiAE}}$ outputs 0. ◻

5 Benchmarks

This section contains performance benchmarks on our implementation of DiAE (and DiSE). The implementation was done in C99 using standard libraries, as well as the mbedtls library for the various cryptographic primitives and the arithmetic.

We implemented and benchmarked both the DiSE and the DiAE protocols, with the basic DPRF ($\Pi_{\mathsf{DDH\text{-}DP}}$) [2, Sect. 8.1] and the updated privately verifiable DPRF ($\Pi_{\mathsf{DDH\text{-}Pri}}$). The implementation relies on elliptic curve cryptography over the curve secp256r1. The hash functions $\mathcal{H}$, $\mathcal{H}'$ were implemented based on the IRTF's Hash to Curve draft [17] (which produces a distribution on the curve that is indistinguishable from uniform), the KDF based on HKDF. For DiAE, the encryptment scheme EC was implemented based on the HFC (Hash Function Chaining) construction described in [16] instantiated with the SHA-256 compression function. For DiSE, we used a hash commitment scheme instantiated with SHA-256; the PRG was implemented with AES-CTR.

In Table 1, we summarize the number of operations required by $\Pi_{\mathsf{DDH\text{-}DP}}$ compared to $\Pi_{\mathsf{DDH\text{-}Pri}}$. We can see that Eval with the privately verifiable DPRF ($\Pi_{\mathsf{DDH\text{-}Pri}}$) has a complexity of about 3 times the basic DPRF ($\Pi_{\mathsf{DDH\text{-}DP}}$). Combine, requires about 4 times more operations.

Table 1: Complexity of $\Pi_{\mathsf{DDH\text{-}DP}}$ and $\Pi_{\mathsf{DDH\text{-}Pri}}$. The operations inside the hash to elliptic curve are not included in the summary.

	Eval		Combine	
Operation	$\Pi_{\mathsf{DDH\text{-}DP}}$	$\Pi_{\mathsf{DDH\text{-}Pri}}$	$\Pi_{\mathsf{DDH\text{-}DP}}$	$\Pi_{\mathsf{DDH\text{-}Pri}}$
# of multiplications by G	0	1	0	$t-1$
# of multiplications by $P \neq G$	1	2	t	$3 \cdot (t-1) + t$
# of point additions	0	0	$t-1$	$3 \cdot (t-1)$
# of hash to elliptic curve	1	1	0	0

The benchmarks in Table 2 were performed on a machine with an AMD Ryzen 5900x 12 cores CPU at 3.70GHz. The time is measured in milliseconds and represent the average of one complete execution of the distributed encryption protocol. For a threshold $t \leq 12$, each partial evalutation was computed on a different core to better represent the distributed part of the protocol. The experimental evaluation, shows that using DiAE versus DiSE does not impact the performance; the execution time is about the same with either protocol. This is expected as the main bottleneck of the two protocols is the execution of the underlying DPRF.

While the performance of both constructions is similar, the memory requirements of DiSE can be substantially higher when working with big messages. This is because DiSE stores a commitment to the message at the start and encrypts the message at the end; thus the entire message must be stored until the end of the distributed protocol. DiAE, on the other hand, only needs to store one binding tag until the end of the protocol as the ciphertext can be transmitted early. In other words, this means that the minimal memory usage of DiSE is linear in the length of the message, while DiAE can have a memory usage that is independent of the message (with a streaming implementation). We summarize the memory requirements of DiSE and DiAE in Table 3. The minimal memory is computed assuming that the variables are discarded immediately when they are not needed.

6 Conclusion

As an enabler for HW-independent key protection, as well as a cryptographic representation of a group of devices, threshold cryptography has been gaining new momentum [31]. Practically tractable

Table 2: Average performance of 1000 distributed encryptions for randomly generated 32 bytes messages in milliseconds.

		$\Pi_{\mathsf{DDH\text{-}DP}}$		$\Pi_{\mathsf{DDH\text{-}Pri}}$	
t	n	DiSE	DiAE	DiSE	DiAE
$n/4$	8	3.4	3.4	10.0	10.5
	12	5.3	5.4	15.5	15.8
	24	11.4	11.3	31.9	32.5
	40	21.0	20.7	55.9	55.5
$n/3$	9	5.3	5.5	15.6	15.4
	12	7.3	7.5	20.7	21.2
	21	13.7	13.6	37.3	37.5
	33	24.0	23.1	60.8	60.7
$2n/3$	9	11.4	11.7	32.4	32.4
	12	16.2	15.8	44.0	44.2
	21	33.4	33.0	86.7	85.6
	33	62.2	60.8	150.3	150.7
n	8	15.4	15.5	43.8	43.9
	12	25.7	25.7	72.4	70.6
	16	38.4	38.6	102.1	100.9
	24	66.1	66.4	167.8	167.2
	32	89.4	89.7	222.3	225.5
	40	118.1	118.1	282.8	284.5

Table 3: Memory requirements of DiSE and DiAE using $\Pi_{\mathsf{DDH\text{-}DP}}$ in bytes. DiAE uses streaming to obtain a message / ciphertext of one block each. Values in blue indicate the usage of $\Pi_{\mathsf{DDH\text{-}Pri}}$. $\mathsf{len}(m)$ denotes the length of m in bytes.

Algorithm	Variable	DiSE	DiAE
Setup	pp	192	192
	sk_i	$32 + 64n$	$32 + 64n$
DistEnc	m	$\mathsf{len}(m)$	32
	ρ/K_{EC}	32	64
	$\alpha/\tau_{\mathsf{EC}}$	32	32
	c_{EC}	-	32
	$\{z_i\}_{i\in S}$	$64t + 128t$	$64t + 128t$
	z	64	64
	e	$\mathsf{len}(m) + 32$	64
	Minimal memory	$\mathsf{len}(m) + 64 + 64t + 128t$	$128 + 64t + 128t$

constructions such as DiSE are thus bound to see real-world deployment, e.g., in IoT. Asserting the security of such constructions is of utmost importance. Our contribution in this vein is the identification of security flaws in DiSE's underlying DPRF. Complementing this, our constructive contribution are simple fixes for these flaws, complete with security proofs, as well as a new TSE construction dubbed DiAE. Unlike DiSE, it allows for constant-memory implementations. Our implementation benchmarks show that this new feature comes at no expense at performance.

References

1. Agrawal, S., Miao, P., Mohassel, P., Mukherjee, P.: PASTA: password-based threshold authentication. In: CCS. pp. 2042–2059. ACM (2018)
2. Agrawal, S., Mohassel, P., Mukherjee, P., Rindal, P.: Dise: distributed symmetric-key encryption. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1993–2010 (2018)
3. Agrawal, S., Mohassel, P., Mukherjee, P., Rindal, P.: Dise: Distributed symmetric-key encryption. Cryptology ePrint Archive, Report 2018/727 (2018), https://ia.cr/2018/727
4. Agrawal, S., Mohassel, P., Mukherjee, P., Rindal, P.: Dise: Distributed symmetric-key encryption. NIST Threshold Cryptography Workshop 2019 (NTCW19) (2019), https://csrc.nist.gov/CSRC/media/Events/NTCW19/papers/paper-AMMR.pdf
5. Baird, L., Mukherjee, P., Sinha, R.: Temp: Time-locked encryption made practical. Cryptology ePrint Archive, Report 2021/800 (2021), https://ia.cr/2021/800
6. Bendlin, R., Damgård, I.: Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In: TCC. Lecture Notes in Computer Science, vol. 5978, pp. 201–218. Springer (2010)
7. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: Desmedt, Y.G. (ed.) Public Key Cryptography — PKC 2003. pp. 31–46. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
8. Boneh, D., Boyen, X., Halevi, S.: Chosen ciphertext secure public key threshold encryption without random oracles. In: CT-RSA. Lecture Notes in Computer Science, vol. 3860, pp. 226–243. Springer (2006)
9. Canetti, R., Goldwasser, S.: An efficient *Threshold* public key cryptosystem secure against adaptive chosen ciphertext attack. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 1592, pp. 90–106. Springer (1999)
10. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) Advances in Cryptology — CRYPTO' 92. pp. 89–105. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)
11. Christodorescu, M., Gaddam, S., Mukherjee, P., Sinha, R.: Amortized threshold symmetric-key encryption. In: CCS. pp. 2758–2779. ACM (2021)
12. Corporation., A.: 8-bit Atmel Microcontroller with 128KBytes In-SystemProgrammable Flash (2011), https://ww1.microchip.com/downloads/en/DeviceDoc/doc2467.pdf
13. Damgård, I., Koprowski, M.: Practical threshold rsa signatures without a trusted dealer. In: Pfitzmann, B. (ed.) Advances in Cryptology — EUROCRYPT 2001. pp. 152–165. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
14. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 307–315. Springer New York, New York, NY (1990)
15. Dodis, Y.: Efficient construction of (distributed) verifiable random functions. In: Desmedt, Y.G. (ed.) Public Key Cryptography — PKC 2003. pp. 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
16. Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast message franking: From invisible salamanders to encryptment. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018. pp. 155–186. Springer International Publishing, Cham (2018)
17. Faz-Hernández, A., Scott, S., Sullivan, N., Wahby, R.S., Wood, C.A.: Hashing to Elliptic Curves. Internet-Draft draft-irtf-cfrg-hash-to-curve-13, Internet Engineering Task Force (Nov 2021), https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-13, work in Progress
18. Franklin, M., Zhang, H.: Unique ring signatures: A practical construction. In: Sadeghi, A.R. (ed.) Financial Cryptography and Data Security. pp. 162–170. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
19. Gennaro, R., Halevi, S., Krawczyk, H., Rabin, T.: Threshold rsa for dynamic and ad-hoc groups. In: Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology. p. 88–107. EUROCRYPT'08, Springer-Verlag, Berlin, Heidelberg (2008)
20. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold dss signatures. In: Maurer, U. (ed.) Advances in Cryptology — EUROCRYPT '96. pp. 354–371. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)

21. HashiCorp: Vault, https://www.vaultproject.io/
22. Hua, S., Li, G., Aimin, W.: A forward secure threshold signature scheme based on bilinear pairing. In: 2010 IEEE International Conference on Intelligent Computing and Intelligent Systems. vol. 3, pp. 403–406 (2010)
23. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and t-pake in the password-only model. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014. pp. 233–253. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
24. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: CRYPTO. Lecture Notes in Computer Science, vol. 6223, pp. 631–648. Springer (2010)
25. Libert, B., Stehlé, D., Titiu, R.: Adaptively secure distributed prfs from lwe. In: Beimel, A., Dziembowski, S. (eds.) Theory of Cryptography. pp. 391–421. Springer International Publishing, Cham (2018)
26. Micali, S., Sidney, R.: A simple method for generating and sharing pseudo-random functions, with applications to clipper-like key escrow systems. In: Coppersmith, D. (ed.) Advances in Cryptology — CRYPT0' 95. pp. 185–196. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
27. Mukherjee, P.: Adaptively secure threshold symmetric-key encryption. In: International Conference on Cryptology in India. pp. 465–487. Springer (2020)
28. Naor, M., Pinkas, B., Reingold, O.: Distributed pseudo-random functions and kdcs. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 327–346. Springer (1999)
29. Neuman, D.C., Hartman, S., Raeburn, K., Yu, T.: The Kerberos Network Authentication Service (V5). RFC 4120 (Jul 2005), https://rfc-editor.org/rfc/rfc4120.txt
30. Nielsen, J.B.: A threshold pseudorandom function construction and its applications. In: Yung, M. (ed.) Advances in Cryptology — CRYPTO 2002. pp. 401–416. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
31. NIST: Multi-party threshold cryptography: Csrc (2018), https://csrc.nist.gov/Projects/threshold-cryptography
32. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140. Springer (1991)
33. Pister, K., Kahn, J., Boser, B.: Smart dust. keynote address, IPSN **3** (2001)
34. Ronen, E., Shamir, A., Weingarten, A., O'Flynn, C.: IoT Goes Nuclear: Creating a ZigBee Chain Reaction. In: IEEE Symposium on Security and Privacy. pp. 195–212. IEEE Computer Society (2017)
35. Samsung: Knox, https://www.samsungknox.com/
36. Santis, A.D., Desmedt, Y., Frankel, Y., Yung, M.: How to share a function securely. In: STOC. pp. 522–533. ACM (1994)
37. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 239–252. Springer New York, New York, NY (1990)
38. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
39. Shoup, V., Gennaro, R.: Securing threshold cryptosystems against chosen ciphertext attack. In: Nyberg, K. (ed.) Advances in Cryptology — EUROCRYPT'98. pp. 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
40. Square: Keywhiz, https://square.github.io/keywhiz/

A Deferred proofs

A.1 Proof of Lemma 4.4 (DiAE Message Privacy)

Proof. First we write the challenge phase of $\mathsf{MsgPriv}^{\mathcal{A}}_{\mathsf{DiAE}}(1^k, b)$ (Def. 7) in detail:

1. $\mathcal{A}$ outputs $(\mathsf{Challenge}, j^\star, m_0, m_1, S^\star)$ where $\mathsf{len}(m_0) = \mathsf{len}(m_1)$, $j^\star \in S^\star \setminus C$ and $|S^\star| \geq t$.
2. Generate the key $K^\star_{\mathsf{EC}} \leftarrow\$ \mathsf{EK_g}$ and compute the encryptment $(c^\star_{\mathsf{EC}}, \tau^\star_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K^\star_{\mathsf{EC}}, j^\star, m_b)$. Compute $z^\star_i \leftarrow \mathsf{DPRF.Eval}(sk_i, j^\star \mid\mid \tau^\star_{\mathsf{EC}}, pp_{\mathsf{DPRF}})$ for $i \in S^\star \setminus C$.
3. Send the binding tag $\tau^\star_{\mathsf{EC}}$ to $\mathcal{A}$ and get back $\hat{z}_i$ for $i \in S^\star \cap C$.
4. Compute $z^\star \leftarrow \mathsf{Combine}(\{(i, z^\star_i)\}_{i \in S^\star \setminus C} \cup \{(i, \hat{z}_i)\}_{i \in S^\star \cap C}, pp_{\mathsf{DPRF}})$. If $z^\star = \perp$, output $\perp$ to $\mathcal{A}$. Else, compute $e^\star := \mathsf{KDF}(z^\star) \oplus K^\star_{\mathsf{EC}}$. Send $c^\star := (j^\star, c^\star_{\mathsf{EC}}, \tau^\star_{\mathsf{EC}}, e^\star)$ to $\mathcal{A}$.

We use a sequence of hybrid to prove TSE satisfies message privacy. In the hybrids below, only the changes from the previous hybrid are shown.

– Hyb_1: If a non-unique binding tag $\tau^\star_{\mathsf{EC}}$ is output by EC, notify the adversary and end the challenge.
– Hyb_2: Replace the input of KDF by a random string of same length.
– Hyb_3: Replace the output of KDF by a random string of same length.
– Hyb_4: Replace $e^\star$ by a random string of length $\mathsf{len}(K^\star_{\mathsf{EC}})$.
– Hyb_5: Replace $(c^\star_{\mathsf{EC}}, \tau^\star_{\mathsf{EC}})$ by an encryptment to a random message.

In $\mathsf{DistEnc}$, the key K_{EC} is sampled uniformly at random and the encryptment scheme is binding which ensures that an adversary cannot find two triplets $(j, m, K_{\mathsf{EC}}) \neq (j', m', K'_{\mathsf{EC}})$ that output the same tag τ_{EC}. Therefore, as $K^\star_{\mathsf{EC}}$ is sampled uniformly at random, $\mathsf{MsgPriv}$ is indistinguishable from Hyb_1. Hyb_3 is indistinguishable from Hyb_2 as the output of KDF appears pseudorandom to adversaries. Similarly, Hyb_4 is indistinguishable from Hyb_3. As EC is otROR secure, Hyb_5 is indistinguishable from Hyb_4.

We are left to prove indistinguishability of Hyb_2 from Hyb_1. On the contrary, suppose that there exists PPT adversary $\mathcal{A}$ that can distinguish between those two hybrids with non-negligible probability. We use $\mathcal{A}$ to build another PPT adversary $\mathcal{B}$ who succeeds in the $\mathsf{PseudoRand}$ game of DPRF (Def. 2) with the same probability:

– *Initialization.* Get pp_{DPRF} from Chal (Chal is used as the challenger in $\mathsf{PseudoRand}$) and give it to $\mathcal{A}$.
– *Corruption.* Receive the set of corrupt parties C from $\mathcal{A}$, where $|C| < t$. Forward it to *Chal* to get $\{sk_i\}_{i \in C}$ and send it to $\mathcal{A}$.
– *Pre-challenge encryption queries.* Suppose $\mathcal{A}$ outputs an encryption query $(\mathsf{Encrypt}, j, m, S)$ where $j \in S$ and $|S| \geq t$. There are two scenarios based on the value of j:
  - If j is honest (i.e. $j \in S \setminus C$), a distributed encryption is executed. First, compute the encryptment $(c_{\mathsf{EC}}, \tau_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, j, m)$ where $K_{\mathsf{EC}} \leftarrow\$ \mathsf{EK_g}$. If the binding tag τ_{EC} was already generated in a previous query, notify $\mathcal{A}$ and stop. Otherwise, request the values $z_i,\ i \in S \cap C$ from $\mathcal{A}$. Send $(\mathsf{Eval}, j \mid\mid \tau_{\mathsf{EC}}, i)$ to Chal to get the honest evals $z_i,\ i \in S \setminus C$. Run $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp_{\mathsf{DPRF}})$ to get $z/\bot$. If $\bot$ is output, send it to $\mathcal{A}$ and stop. Otherwise, compute $e := \mathsf{KDF}(z) \oplus K_{\mathsf{EC}}$ and send $c := (j, c_{\mathsf{EC}}, \tau_{\mathsf{EC}}, e)$ to $\mathcal{A}$.
  - if j is corrupt (i.e. $j \in S \cap C$), $\mathcal{A}$ expects the honest parties in S to help compute a $\mathsf{DPRF.Eval}$. Upon reception of a message x sent to an honest party j' from $\mathcal{A}$, verify that x is of the form $j^\star \mid\mid \tau^\star_{\mathsf{EC}}$ where $j^\star = j$. If not, return $\bot$, otherwise send (Eval, x, j') to Chal and forward the response to $\mathcal{A}$.
– *Pre-challenge indirect decryption queries.* Suppose $\mathcal{A}$ outputs a decryption query $(\mathsf{Decrypt}, j, c, S)$ where $j \in S \setminus C$ and $|S| \geq t$. Parse c as $(j', c_{\mathsf{EC}}, \tau_{\mathsf{EC}}, e)$ and send $j' \mid\mid \tau_{\mathsf{EC}}$ to $\mathcal{A}$.
– *Challenge.* $\mathcal{A}$ outputs $(\mathsf{Challenge}, j^\star, m_0, m_1, S^\star)$ where $\mathsf{len}(m_0) = \mathsf{len}(m_1)$, $j^\star \in S^\star \setminus C$ and $|S^\star| \geq t$. Generate the key $K^\star_{\mathsf{EC}} \leftarrow\$ \mathsf{EK_g}$ and compute the encryptment $(c^\star_{\mathsf{EC}}, \tau^\star_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K^\star_{\mathsf{EC}}, j^\star, m_b)$. If the binding tag τ_{EC} had been already generated in a previous query, notify $\mathcal{A}$ and stop. Otherwise, request the values $z^\star_i,\ i \in S^\star \cap C$ from $\mathcal{A}$. Send $(\mathsf{Challenge}, j^\star \mid\mid \tau^\star_{\mathsf{EC}}, S^\star, \{(i, z^\star_i)\}_{i \in S^\star \cap C})$ to Chal. If Chal outputs $z^\star = \bot$, send it to $\mathcal{A}$. Otherwise, compute $e^\star := \mathsf{KDF}(z^\star) \oplus K_{\mathsf{EC}^\star}$ and send $c^\star := (j^\star, c^\star_{\mathsf{EC}}, \tau^\star_{\mathsf{EC}}, e^\star)$ to $\mathcal{A}$.
– *Post-challenge encryption queries.* Same as *Pre-challenge encryption queries.*
– *Post-challenge indirect decryption queries.* Same as *Pre-challenge indirect decryption queries.*
– *Guess.* Finally, $\mathcal{A}$ returns a guess b'. Output b'.

$\mathcal{B}$ requests a $\mathsf{DPRF.Eval}$ only in the $\mathsf{pre/post\text{-}challenge\ encryption\ query}$. In the first scenario, as j is honest, the protocol aborts if τ_{EC} is not unique. The second scenario, when j is corrupt, $j^\star \mid\mid \tau^\star_{\mathsf{EC}}$

can never be requested because $j^\star$ is honest; thus $\mathcal{B}$ never requests an Eval on $j^\star \mid\mid \tau^\star_{\mathsf{EC}}$. Because of that, the challenge sent to Chal is valid. Therefore, it perfectly simulates Hyb_1 when $\mathsf{Combine}$ is computed normally and it perfectly simulates Hyb_2 when the result of $\mathsf{Combine}$ is replaced by a random string. $\square$

A.2 Proof of Lemma 4.5 (DiAE Authenticity)

Proof. The proof below is based on DiSE's proof but adapted to DiAE.

We describe the forgery step of the authentication game ($\mathsf{Auth}^{\mathcal{A}}_{\mathsf{DiAE}}$) in detail. ct denotes the number of honest parties contacted by the adversary, δ is the minimum number of honest parties that must be contacted to generate a valid ciphertext. $r := \lfloor \mathsf{ct}/\delta \rfloor$ is the number of ciphertexts learned by the adversary.

1. $\mathcal{A}$ outputs $((j_1, S_1, c_1), \ldots, (j_{r+1}, S_{r+1}, c_{r+1}))$ where $j_\ell \in S_\ell \setminus C$, $|S_\ell| \geq t$ for all $\ell \in [r+1]$ and $c_u \neq c_v$ for any $u \neq v \in [r+1]$. Let $c_\ell = (j_\ell, c_{\ell_{\mathsf{EC}}}, \tau_{\ell_{\mathsf{EC}}}, e_\ell)$.
2. For $\ell \in [r+1]$, compute $z_{\ell,i} \leftarrow \mathsf{Eval}(sk_i, j_\ell \mid\mid \tau_{\ell_{\mathsf{EC}}}, pp_{\mathsf{DPRF}})$ for $i \in S_\ell$.
3. Compute $z_\ell \leftarrow \mathsf{Combine}(\{i, z_{\ell,i}\}_{i \in S_\ell}, pp_{\mathsf{DPRF}})$ for $\ell \in [r+1]$. If any $z_\ell = \perp$ for $\ell \in [r+1]$, output 0. Otherwise, compute $K_{\ell_{\mathsf{EC}}} := \mathsf{KDF}(z_\ell) \oplus e_\ell$ and decrypt the message by computing $m_\ell \leftarrow \mathsf{DO}(K_{\ell_{\mathsf{EC}}}, j_\ell, c_{\ell_{\mathsf{EC}}}, \tau_{\ell_{\mathsf{EC}}})$. Output 0 if $m_\ell = \perp$ for any ℓ. Output 1 otherwise.

First, we define a new hybrid, $\mathsf{Auth\text{-}U}^{\mathcal{A}}_{\mathsf{DiAE}}$, that ensures the input of Eval is unique. The delta from the previous game is highlighted in blue:

1. $\mathcal{A}$ outputs $((j_1, S_1, c_1), \ldots, (j_{r+1}, S_{r+1}, c_{r+1}))$ where $j_\ell \in S_\ell \setminus C$, $|S_\ell| \geq t$ for all $\ell \in [r+1]$ and $c_u \neq c_v$ for any $u \neq v,\ u, v \in [r+1]$. Let $c_\ell = (j_\ell, c_{\ell_{\mathsf{EC}}}, \tau_{\ell_{\mathsf{EC}}}, e_\ell)$. Output 0 if for any $u \neq v,\ j_u = j_v$ and $\tau_{u_{\mathsf{EC}}} = \tau_{v_{\mathsf{EC}}}$

To prove indistinguishability of $\mathsf{Auth\text{-}U}^{\mathcal{A}}_{\mathsf{DiAE}}$ from $\mathsf{Auth}^{\mathcal{A}}_{\mathsf{DiAE}}$ we need to show that $\mathsf{Auth}^{\mathcal{A}}_{\mathsf{DiAE}}$ also outputs 0 when the added condition is satisfied. Suppose $\mathcal{A}$ outputs $r+1$ distinct ciphertexts where two ciphertexts c_u and c_v satisfy $j_u = j_v$ and $\tau_{u_{\mathsf{EC}}} = \tau_{v_{\mathsf{EC}}}$. As the parties have to behave honestly and due to the consistency property of DPRF, we know that $z_u = z_v$; thus, $(c_{u_{\mathsf{EC}}}, K_{u_{\mathsf{EC}}}) \neq (c_{v_{\mathsf{EC}}}, K_{v_{\mathsf{EC}}})$. After the decryption, we obtain the pairs $(m_u, K_{u_{\mathsf{EC}}})$, $(m_v, K_{v_{\mathsf{EC}}})$. If the pairs are different, this leads to a contradiction as there cannot be two distinct header/message/key pair that validate under the same tag τ_{EC} due to the binding security of EC (Def. 14). Thus, either $m_u = \perp$ and/or $m_v = \perp$. If the pairs are equal and $m_u = m_v \neq \perp$, $K_{u_{\mathsf{EC}}} = K_{v_{\mathsf{EC}}} \implies e_u = e_v \implies c_{u_{\mathsf{EC}}} \neq c_{v_{\mathsf{EC}}}$, which breaks strong correctness[6] of EC (Def. 11).

We showed that $\mathsf{Auth\text{-}U}^{\mathcal{A}}_{\mathsf{DiAE}}$ is indistinguishable from $\mathsf{Auth}^{\mathcal{A}}_{\mathsf{DiAE}}$. Now, we show that if there exists a PPT adversary $\mathcal{A}$ where $\mathsf{Auth\text{-}U}^{\mathcal{A}}_{\mathsf{DiAE}}$ outputs 1 with probability at least ε, we use $\mathcal{A}$ to build another adversary $\mathcal{B}$ that has an advantage of a least $\varepsilon - \mathbf{negl}(k)$ in the game $\mathsf{PseudoRand}^{\mathcal{A}}_{\mathsf{DPRF}}$ (Def. 2). Chal is the challenger in the aforementioned game, $\mathcal{B}$ acts as follow:

- *Initialization.* Get pp_{DPRF} from Chal and give it to $\mathcal{A}$. Initialize an ordered list $L_{\mathsf{p\text{-}ctxt}} := \varnothing$ and a counter $\mathsf{ct}_{j,\tau_{\mathsf{EC}}} := 0$ for every (j, τ_{EC}).
- *Corruption.* Receive the set of corrupt parties C from $\mathcal{A}$, where $|C| < t$. Forward it to *Chal* to get $\{sk_i\}_{i \in C}$ and send it to $\mathcal{A}$. Set $g := t - |C|$.
- *Encryption queries.* Suppose $\mathcal{A}$ outputs an encryption query $(\mathsf{Encrypt}, j, m, S)$ where $j \in S$ and $|S| \geq t$. There are two scenarios based on the value of j:
  - if j is honest (i.e. $j \in S \setminus C$), first compute the encryptment $(c_{\mathsf{EC}}, \tau_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, j, m)$ where $K_{\mathsf{EC}} \leftarrow\$ \; \mathsf{EK_g}$. Send $j \mid\mid \tau_{\mathsf{EC}}$ to $\mathcal{A}$. Append (j, τ_{EC}) to $L_{\mathsf{p\text{-}ctxt}}$.

[6] There is a unique encryptment $(c_{\mathsf{EC}}, \tau_{\mathsf{EC}})$ for all (j, m, K_{EC}) pairs.

  - if j is corrupt (i.e. $j \in S \cap C$), $\mathcal{A}$ expects the honest parties in S to help compute a $\mathsf{DPRF.Eval}$. Upon reception of a message x sent to an honest party j' from $\mathcal{A}$, verify that x is of the form $j^\star \mid\mid \tau^\star_{\mathsf{EC}}$ where $j^\star = j$. If not, return $\perp$, otherwise send (Eval, x, j') to Chal and forward the response to $\mathcal{A}$. Increment $\mathsf{ct}_{j^\star,\tau^\star_{\mathsf{EC}}}$ by 1.
- *Decryption queries.* Suppose $\mathcal{A}$ outputs a decryption query $(\mathsf{Decrypt}, j', c, S)$ where $j' \in S$ and $|S| \geq t$. Parse c into $(j, \tau_{\mathsf{EC}}, c_{\mathsf{EC}}, e)$. There are two scenarios based on the value of j':
  - if j' is honest (i.e. $j' \in S \setminus C$), send $j \mid\mid \tau_{\mathsf{EC}}$.
  - if j' is corrupt (i.e. $j' \in S \cap C$), $\mathcal{A}$ expects the honest parties in S to help compute a $\mathsf{DPRF.Eval}$. Upon reception of a message x sent to an honest party j'' from $\mathcal{A}$, verify that x is of the form $j^\star \mid\mid \tau^\star_{\mathsf{EC}}$ where $j^\star \in [n]$. If not, return $\perp$, otherwise send (Eval, x, j'') to Chal and forward the response to $\mathcal{A}$. Increment $\mathsf{ct}_{j^\star,\tau^\star_{\mathsf{EC}}}$ by 1.
- *Targeted decryption queries.* Suppose $\mathcal{A}$ outputs a targeted decryption query $(\mathsf{TargetDecrypt}, j', \ell, S)$ where $j' \in S \setminus C$ and $|S| \geq t$. Let (j, τ_{EC}) be the ℓ-th entry of $L_{\mathsf{p\text{-}ctxt}}$. Send $j \mid\mid \tau_{\mathsf{EC}}$ to $\mathcal{A}$.
- *Forgery.* $\mathcal{A}$ outputs $((j_1, S_1, c_1), \ldots, (j_\omega, S_\omega, c_\omega))$ where $j_\ell \in S \setminus C$, $|S_\ell| \geq t$ for all $\ell \in [\omega]$ and $c_u \neq c_v$ for any $u \neq v$, $u, v \in [\omega]$. Let $c_\ell = (j_\ell, c_{\ell_{\mathsf{EC}}}, \tau_{\ell_{\mathsf{EC}}}, e_\ell)$ for $\ell \in [\omega]$.
  - if for any $u \neq v$, $j_u = j_v$ and $\tau_{u_{\mathsf{EC}}} = \tau_{v_{\mathsf{EC}}}$, output 1 as the guess to Chal and stop.
  - Pick a ciphertext $c_{\ell^\star}$ such that $\mathsf{ct}_{j_{\ell^\star},\tau_{\ell^\star_{\mathsf{EC}}}} < g$. Send $(\mathsf{Challenge}, j_{\ell^\star} \mid\mid \tau_{\ell^\star_{\mathsf{EC}}}, S_{\ell^\star}, \varnothing)$ to Chal. Let $z^\star$ be the response of Chal, if $z^\star = \perp$, output 1; otherwise compute $K_{\ell^\star_{\mathsf{EC}}} := \mathsf{KDF}(z^\star) \oplus e_{\ell^\star}$ and compute the message $m_{\ell^\star} \leftarrow \mathsf{DO}(K_{\ell^\star_{\mathsf{EC}}}, j_{\ell^\star}, c_{\ell^\star_{\mathsf{EC}}}, \tau_{\ell^\star_{\mathsf{EC}}})$. If $m_{\ell^\star} = \perp$ output 1, otherwise output 0.

The sum of $\mathsf{ct}_{j,\tau_{\mathsf{EC}}}$ for all (j, τ_{EC}) is at most ct, the variable in $\mathsf{Auth\text{-}U}^{\mathcal{A}}_{\mathsf{DiAE}}$ that counts the number of contacted honest parties. In turn ct must be less than $\omega \cdot g$ (as $r := \lfloor \mathsf{ct}/g \rfloor$); thus there is a ciphertext $c_{\ell^\star}$ such that $\mathsf{ct}_{j_{\ell^\star},\tau_{\ell^\star}} < g$. This ensures that $j_{\ell^\star} \mid\mid \tau_{\ell^\star_{\mathsf{EC}}}$ is not in the L list maintained by Chal (otherwise $\perp$ is output as a response to $\mathcal{B}$'s challenge query).
From $\mathcal{A}$'s perspective, $\mathcal{B}$ perfectly simulates $\mathsf{Auth\text{-}U}^{\mathcal{A}}_{\mathsf{DiAE}}$. Let b' denote the bit output by $\mathcal{B}$ at the end. First, suppose the bit b in $\mathsf{PseudoRand}^{\mathcal{A}}_{\mathsf{DPRF}}$ is 0. Looking at $\mathsf{Auth\text{-}U}^{\mathcal{A}}_{\mathsf{DiAE}}$, one can see that 1 is output only if $z_\ell \neq \perp$ and $m_\ell \neq \perp$ for all ℓ. As we assumed that $\mathsf{Auth\text{-}U}^{\mathcal{A}}_{\mathsf{DiAE}}$ outputs 1 with probability at least ε, $\mathsf{Pr}[b' = 1 \mid b = 0] \leq 1 - \varepsilon$. Now, suppose the bit b in $\mathsf{PseudoRand}^{\mathcal{A}}_{\mathsf{DPRF}}$ is 1. In this case, $z^\star$ is picked randomly so in turn $K_{\ell^\star_{\mathsf{EC}}}$ is pseudorandom. The probability that $\mathsf{DO}(K_{\ell^\star_{\mathsf{EC}}}, j_{\ell^\star}, c_{\ell^\star_{\mathsf{EC}}}, \tau_{\ell^\star_{\mathsf{EC}}}) \neq \perp$ is negligible ($\mathsf{sr\text{-}BIND}$ security of EC). Thus, $\mathsf{Pr}[b' = 1 \mid b = 0] \geq 1 - \mathbf{negl}(k)$. Putting both probabilities together:

$$\Big|\mathsf{Pr}\big[b' = 1 \mid b = 0\big] - \mathsf{Pr}\big[b' = 1 \mid b = 1\big]\Big| \geq \varepsilon - \mathbf{negl}(k)$$

□

B Missing definitions

Definition B.17 (Shamir's Secret Sharing). *Shamir's Secret Sharing [38] is an algorithm* SSS *that generates shares based on the inputs n, t, p, s. n is the number of shares to be generated, t is the threshold ("the number of shares required to reconstruct s"), p is a prime number and s is the secret.*

SSS *is information theoretic secure; no information is learned on s until at least t shares are learned.*

Definition B.18 (Commitment scheme). *A commitment scheme* Com *is a pair of algorithms* $(\mathsf{Setup_{com}}, \mathsf{Commit})$.

$\mathsf{Setup_{com}}(1^k) \to pp_{\mathsf{com}}$*: Takes the security parameter and outputs public parameters.*

$\mathsf{Commit}(m, pp_{\mathsf{com}}; \rho) \to \alpha$*: Outputs a commitment α based on a message, public parameters and some randomness.*

Binding. *The commitment scheme is (t, ε)-binding if for any t-time adversary $\mathcal{A}$, the following is true:*

$$\mathsf{Pr}\big[\mathsf{Commit}(m, pp_{\mathsf{com}}; \rho) = \mathsf{Commit}(m', pp_{\mathsf{com}}; \rho') \mid (m, m', \rho, \rho') \leftarrow\!\!\$\, \mathcal{A},\ m \neq m'\big] \leq \varepsilon$$

Hiding. *The commitment scheme is (t, ε)-hiding if for any t-time adversary $\mathcal{A}$, the following is true:*

$$\Big|\mathsf{Pr}\big[\mathcal{A}(pp_{\mathsf{com}}, \mathsf{Commit}(m, pp_{\mathsf{com}}; \rho)) = 1\big] - \mathsf{Pr}\big[\mathcal{A}(pp_{\mathsf{com}}, \mathsf{Commit}(m', pp_{\mathsf{com}}; \rho')) = 1\big]\Big| \leq \varepsilon$$

where $pp_{\mathsf{com}} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^k),\ m \neq m'$.

Definition B.19 (Trapdoor Commitment scheme (TDC)). *A trapdoor commitment scheme* TDC *is composed of a commitment scheme* Com $:= (\mathsf{Setup}_{\mathsf{com}}, \mathsf{Commit})$ *and two additional algorithms* (SimSetup, SimOpen).

$\mathsf{SimSetup}(1^k) \to (pp_{\mathsf{com}}, \tau_{\mathsf{TDC}})$*: Takes the security parameter and outputs public parameters and a trapdoor.*

$\mathsf{SimOpen}(pp_{\mathsf{com}}, \tau_{\mathsf{TDC}}, m', (m, \rho)) \to \rho'$*: Takes the parameters generated by* SimSetup*, a message m' and a message-randomness pair. Outputs a randomness ρ'.*

For all (m, ρ) and m', there exists a negligible function such that:

pp_{com} *and* pp'_{com} *are statistically close, and,*

$$\mathsf{Pr}\big[\mathsf{Commit}(m, pp'_{\mathsf{com}}; \rho) = \mathsf{Commit}(m', pp'_{\mathsf{com}}; \rho') \\ \mid \rho' \leftarrow \mathsf{SimOpen}(pp'_{\mathsf{com}}, \tau_{\mathsf{TDC}}, m', (m, \rho))\big] \geq 1 - \mathbf{negl}(k)$$

where $pp_{\mathsf{com}} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^k)$ *and* $(pp'_{\mathsf{com}}, \tau_{\mathsf{TDC}}) \leftarrow \mathsf{SimSetup}(1^k)$.

Definition B.20 (Non-Interactive Zero Knowledge (NIZK)). *The definition below is taken from [3]. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{poly(k)}$ be a hash function modeled as a random oracle. A* NIZK *for a binary relation R consists of two PPT algorithms Prove and Verify with oracle access to $\mathcal{H}$.*

$\mathsf{Prove}^{\mathcal{H}}(s, w)$ *takes a statement s and a witness w as input; outputs a proof π if $(s, w) \in R$ and $\perp$ otherwise.*

$\mathsf{Verify}^{\mathcal{H}}(s, \pi)$ *takes a statement s and a proof π; outputs $b = 1$ if the proof is valid, $b = 0$ otherwise.*

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

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

Zero-knowledge. *There exists a pair of PPT simulators $(\mathcal{S}_1, \mathcal{S}_2)$ such that for all PPT adversary $\mathcal{A}$ there is a negligible function:*

$$\Big|\mathsf{Pr}\big[\mathcal{A}^{\mathcal{H}, \mathsf{Prove}^{\mathcal{H}}}(1^k) = 1\big] - \mathsf{Pr}\big[\mathcal{A}^{\mathcal{S}_1, \mathcal{S}'_2}(1^k) = 1\big]\Big| \leq \mathbf{negl}(k)$$

where $\mathcal{S}_1$ simulates the random oracle $\mathcal{H}$, $\mathcal{S}'_2$ returns a simulated proof $\pi \leftarrow \mathcal{S}_2(s)$ on input (s, w) if $(s, w) \in R$, $\perp$ otherwise. $\mathcal{S}_1$ and $\mathcal{S}_2$ share states.

Argument of knowledge. *There exists a PPT simulator $\mathcal{S}_1$ such that for all PPT adversary $\mathcal{A}$, there is a PPT extractor $\mathcal{E}^{\mathcal{A}}$ such that:*

$$\mathsf{Pr}\big[(s, w) \in R \textit{ and } \mathsf{Verify}^{\mathcal{H}}(s, \pi) = 1 \mid (s, \pi) \leftarrow\!\!\$\, \mathcal{A}^{\mathcal{S}_1}(1^k);\ w \leftarrow \mathcal{E}^{\mathcal{A}}(s, \pi, Q)\big] \leq \mathbf{negl}(k)$$

where $\mathcal{S}_1$ simulates the random oracle $\mathcal{H}$, Q is the list of (query, response) pairs obtained from $\mathcal{S}_1$.

Definition B.21 (Key Derivation Function (KDF) [24]). *A key derivation function as a deterministic function that takes as input four arguments; a value from a source of keying material x, an output length ℓ, a salt and a context. For simplicity's sake, we only include the first argument in the paper; we assume the salt and the context are either constant or null and that ℓ is of the correct length (i.e. $\mathsf{KDF}(x) \oplus m \implies \ell = \mathsf{len}(m)$).*
Pseudorandomness. Informally, a KDF is (t, q, ε)-secure with respect to a source of keying material if no attacker $\mathcal{A}$ running in time t, making at most q queries, can distinguish between the output of KDF with a chosen input and a random byte string of same length with probability greater than $1/2 + \varepsilon$.