The proceeding version of this work appears in INDOCRYPT2022. This is the full version.

ParaDiSE: Efficient Threshold Authenticated Encryption in Fully Malicious Model

Shashank Agrawal*[1], Wei Dai*[2], Atul Luykx*[3], Pratyay Mukherjee*[4], and Peter Rindal[5]

[1]Coinbase
[2]Bain Capital Crypto
[3]Google
[4]SupraOracles
[5]Visa Research

October 25, 2022

Abstract

Threshold cryptographic algorithms achieve robustness against key and access compromise by distributing secret keys among multiple entities. Most prior work focuses on threshold public-key primitives, despite extensive use of authenticated encryption in practice. Though the latter can be deployed in a threshold manner using multi-party computation (MPC), doing so incurs a high communication cost. In contrast, dedicated constructions of threshold authenticated encryption algorithms can achieve high performance. However to date, few such algorithms are known, most notably DiSE (distributed symmetric encryption) by Agrawal et al. (ACM CCS 2018). To achieve *threshold authenticated encryption* (TAE), prior work does not suffice, due to shortcomings in definitions, analysis, and design, allowing for potentially insecure schemes, an undesirable similarity between encryption and decryption, and insufficient understanding of the impact of parameters due to lack of concrete analysis. In response, we revisit the problem of designing secure and efficient TAE schemes. (1) We give new TAE security definitions in the fully malicious setting addressing the aforementioned concerns. (2) We construct efficient schemes satisfying our definitions and perform concrete and more modular security analyses. (3) We conduct an extensive performance evaluation of our constructions, against prior ones.

*Work done while at Visa Research.

Contents

1 Introduction

Cryptography is increasingly deployed within and across organizations to secure valuable data and enforce authorization. Due to regulations such as GDPR and PCI, or via chip- and cryptocurrency-based payments, a monetary amount can be attached to the theft, loss, or misuse of cryptographic keys. Securing cryptographic keys is only becoming more important.

Underlining the significance of securing keys is the proliferation of trusted hardware on a wide range of devices, with manufacturers installing *secure enclaves* on mobile devices (Apple [app], Google [goo]) and commodity processors (ARM [arm], Intel SGX [sgx]). In addition, organizations increasingly use hardware security modules (HSMs) to generate and secure their cryptographic keys. However, such trusted hardware is expensive to build and deploy, often difficult to use, offers limited flexibility in supported operations, and can be difficult to secure at large scale — for example SGX attacks such as [LSG+18, KHF+19] or the recent HSM attack [BC]. As a result, many seek to reduce reliance on trusted hardware.

Threshold cryptography considers a different, and complementary approach to using trusted hardware: instead of relying on the security of each individual device, a threshold number of devices must be compromised, thereby complicating attacks. Furthermore, as threshold cryptography can be deployed in software, it provides a method of securing keys that is cheaper, faster to deploy, and more flexible. It also provides a way to cryptographically enforce business policies that require a threshold number of stakeholders to sign off on decisions.

The benefits of threshold cryptography have caught the attention of practitioners. For example, the U.S. National Institute of Standards and Technology (NIST) has initiated an effort to standardize threshold cryptography [nis]. Furthermore, an increasing number of commercial products use the technology, such as the data protection services offered by Vault [vaua], Curv [cura], and Coinbase Custody [coi], and the HSM replacements by Unbound Tech [unb] and Sepior [sep].

Threshold symmetric encryption is used in many of the commercial products to protect stored data, generate tokens or randomness. The schemes used vary in sophistication, choosing different trade-offs between security, performance, and other deployment concerns.

1.1 Approaches to Threshold Symmetric Encryption

A naive way of deploying threshold symmetric encryption uses secret sharing. One takes an algorithm, for example the authenticated encryption (AE) scheme AES-GCM [SMC08], and applies a secret sharing scheme to its key. The key shares are sent to different parties so that one has to contact a threshold number of parties to reconstruct the key, to then perform encryption or decryption. However, this approach requires *reconstructing* the key at some point, thereby nearly negating the benefits of splitting the secret among multiple parties in the first place.

Instead, a proper threshold cryptographic implementation of AES-GCM would not require key reconstruction — even while encrypting or decrypting. One could secret share the plaintext instead of the key, and send each share to different parties holding different keys. This avoids key reconstruction, but significantly increases communication and storage if applied to an AE scheme like AES-GCM.

Secure multi-party computation (MPC) also enables implementations of cryptographic algorithms such as AES-GCM in a way that keys remain split during operation. MPC works with any algorithm and therefore is used in applications where backwards compatibility is important, such as the drop-in replacement for HSMs discussed by Archer et al. [ABL+18]. However, MPC has a significant performance cost, often requiring multiple rounds of high bandwidth communication among the different parties. Keller et al. [KOR+17] present the best-known performance results:

per 128 bit AES block, they need anywhere from 2.9 to 8.4 MB of communication and at least 10 communication rounds for two-party computation. For settings where backwards compatibility is not needed, using MPC-friendly ciphers such as MiMC [AGR+16] and LowMC [ARS+15] instead of AES can improve performance modestly.

Unlike MPC, Agrawal et al.'s threshold AE (TAE) schemes [AMMR18a] — named *DiSE* — operate in two communication rounds and require anywhere from 32 to 148 bytes per encryption in the two party setting. Furthermore, the DiSE protocols output integrity-protected ciphertext like conventional authenticated encryption schemes and communication complexity does not change with message length. As a result, the DiSE protocols can outperform MPC implementations of authenticated encryption by orders of magnitude. DiSE's efficiency makes it a prime candidate for applications seeking threshold security, which motivates the further study of dedicated TAE schemes.

1.2 Revisiting Threshold Authenticated Encryption

Agrawal et al. [AMMR18a] (hereafter AMMR) initiate the study of TAE schemes to achieve confidentiality and integrity in a threshold setting while ensuring the underlying master secret key remains distributed during encryption and decryption. Security is defined relative to the threshold t, which is one more than the number of malicious parties the protocols can tolerate. Confidentiality is defined as a CPA-like game where adversaries engage in encryption sessions and the goal is to break semantic security of a challenge ciphertext. Integrity is defined as "one-more ciphertext integrity" where an adversary must provide one more valid ciphertext than its "forgery budget".

We note the following three shortcomings of AMMR's formalization.

Confidentiality does not prevent key reconstruction. We show that AMMR's confidentiality definition does not prevent participants from reconstructing master secrets. We give a counter-example scheme that satisfies both confidentiality and integrity as formalized by AMMR, yet allows adversaries to reconstruct the master encryption key, which can then be used to perform encryption without contacting other participants. (See Appendix B for details.) This is because AMMR's CPA-like confidentiality game does not let the adversary initiate decryption sessions, so schemes can disseminate secret keys during decryption. Therefore, a scheme that is proven secure under AMMR's confidentiality notion may not prevent key reconstruction. Their protocols, however, *do* prevent that.

Loose notion of integrity. Participants in the DiSE protocols cannot distinguish whether they are participating in an encryption or a decryption, and adversaries can generate a valid ciphertext while running a decryption session — something that, ideally, should only be possible during encryption. In practice, this can cause difficulties in logging or enforcing permissions and is generally an undesirable property. The fact that participants cannot distinguish encryption from decryption is not just a property of the DiSE protocols but allowed by AMMR's integrity definition (decryption sessions count towards the "forgery budget").

In addition, AMMR's integrity definition allows for "malleable" TAE schemes, where adversaries can participate in an encryption session, make an honest party output an invalid ciphertext, and then "patch" this ciphertext to obtain the correct ciphertext that the honest party should have output (see Appendix C for more details). As a result, ciphertext integrity is not maintained. In AMMR's integrity game (termed *authenticity* by the authors), ciphertexts generated by an honest party are *not* returned to the adversary which deprives it of having a ciphertext to patch in the first place.

Abstract and non-concrete treatment of security. AMMR's definitional framework captures a wide class of protocols that could have parties arbitrarily interacting with each other. Although general,

this approach complicates the formalization of adversarial power. Moreover, the security analyses of the DiSE protocols are asymptotic, providing little guidance on how to securely instantiate parameters in practice.

1.3 Contributions

In light of AMMR's shortcomings, we seek to advance the state-of-the-art in the formalization, design and analysis of TAE schemes. In addition to a concrete security analysis of our schemes, we give the first concrete analysis of distributed pseudorandom functions (DPRFs) and its verifiable extension DVRF, which might be of independent interest. Our analyses use a new modular technique via formalizing a variant of Matrix-DDH assumption [EHK+13], called *Tensor DDH*, which helps in a tighter security reduction to DDH than AMMR's.

New definitions. We introduce new TAE security definitions which fix the aforementioned issues with AMMR's definitions. To do so, we depart from a more abstract description of TAE schemes, and instead only consider TAE schemes which operate in two communication rounds. We present IND-CCA-type definitions capturing confidentiality and integrity, and preventing key reconstruction. Our definitions require that protocols enable participants to distinguish encryption from decryption. Inspired by the definitions of the same name from the public-key literature [CKN03], we present two definitions — CCA and RCCA ("Replayable" CCA) — which capture two different integrity guarantees: RCCA guarantees plaintext integrity, whereas CCA guarantees ciphertext integrity. We believe RCCA to be sufficient for many applications, and propose CCA for settings where ciphertext integrity is important. Note that, as we show below, achieving CCA security comes at a performance cost relative to RCCA security.

New constructions. We present new constructions satisfying our security definitions. To achieve RCCA we present

1. an approach which departs from DiSE's design (Section 5), by using a type of *all-or-nothing transform* [Riv97, BF18] in combination with forward and inverse block cipher calls during encryption and decryption, respectively, and

2. a new scheme inspired by DiSE combining DPRF and threshold signature.

To achieve CCA, we use a distributed *verifiable* PRF combined with threshold signatures. For a more detailed overview and a comparison among our constructions we refer to Section 2.

Performance evaluation. We present an extensive performance study of our constructions (Appendix A), as well as a comparison with the DiSE protocols. In a three-party setting with threshold set to two, our RCCA-secure random injection-based construction achieves over 777,000 encryptions per second and a latency of 0.1 ms per encryption, and our CCA-secure construction achieves about 350 encryptions per second and a latency of 4 ms per encryption. Although these figures are about 0.7 times those of the comparable DiSE protocols, our constructions guarantee stronger security by satisfying RCCA and CCA notions.

By combining practical considerations, new theoretical design, and concrete analysis, we believe our TAE schemes—collectively named *ParaDiSE*—are sufficiently performant and secure for use in practice, while presenting interesting, novel designs of independent interest.

2 Technical Overview

2.1 Security Definitions

Fully malicious security model. As with AMMR, in our model the adversary obtains the secret keys of corrupt parties and can act on their behalf during encryption or decryption. Moreover, the adversary can initiate the protocols via these corrupt parties and receive the output of the honest parties.

AMMR's message privacy definition does not give the adversary decryption capability, which is what allows for the counter example scheme discussed in Appendix B. In contrast, our model guarantees that even if the adversary can decrypt honestly generated ciphertext, it still cannot decrypt the challenge ciphertexts. Furthermore, AMMR's authenticity definition assumes the decryption protocol is executed honestly. Instead, we require authenticity even if the adversary deviates from the honest decryption protocol.

Capturing Privacy via the Decryption Criteria. Our threshold IND-RCCA and IND-CCA definitions follow the standard left-or-right indistinguishability model: the adversary submits message pairs (m_0, m_1) to obtain challenge encryptions of message m_b for a hidden bit b, and it breaks privacy if it can guess b. If the adversary asks for a challenge ciphertext and then honestly executes the decryption protocol, it can trivially guess b seeing if decryption returns m_0 or m_1; as with standard AE definitions, we need to prevent such "trivial wins". However, how to prevent such trivial wins in the threshold setting is much less clear. For example, we cannot block the adversary from initiating decryption protocols associated with challenge ciphertext c (the participating parties do not get access to the input of the initiating party). Nevertheless, our security model should capture when the adversary has effectively executed an honest decryption session. This is done via the notion of *decryption criteria*. Informally, the *decryption criteria*, $\mathsf{Eval\text{-}MSet}(c, \mathcal{CR})$, for ciphertext c and corrupt set of parties $\mathcal{CR}$, captures the exact set of messages that needs to be sent (and responded to) from the adversary, in order for the adversary to know the decryption of c.

Capturing Authenticity. In standard AE, authenticity is captured via INT-CTXT, which says that the only valid ciphertext that the adversary can generate is what it receives from the encryption oracle. Similar to privacy, complications arise when moving to our setting. First, the adversary could generate ciphertext by initiating encryption sessions itself. Furthermore, an outside observer (even while seeing all the protocol messages), might have no idea what messages the adversary is trying to encrypt. Hence, to exclude trivial wins, we will give a *forgery* budget to the adversary based on the amount of interactions it has with the honest parties. Second, the notion of valid ciphertext requires the running of a decryption session. But we allow the adversary to deviate arbitrarily in *any* execution of all protocols. This means that an adversary could potentially deviate from the protocol to make the decryption valid for some ciphertext c, while the honest decryption of c would return $\perp$. Note that the previous notion from DiSE completely bypasses this difficulty by *assuming* that decryption is executed honestly. We take a different approach. We first consider a relaxation of authenticity of ciphertext to authenticity of plaintext (analogous to INT-PTXT). We ask that valid decryptions to always decrypt to previous seen messages, even if the adversary deviate arbitrarily during decryption (IND-RCCA). This notion still admits fast symmetric-key based schemes. We also consider providing integrity of ciphertext, while having malicious decryption (IND-CCA). Our model is given in full detail in Section 4.3.

2.2 Constructions

We provide an overview for each of our three constructions. First we discuss the random injection-based construction which utilizes a recent construction of indifferentiable authenticated encryption [BF18]. Then, we describe the generic DPRF-based approach which builds on the DiSE protocol and yields two constructions. The first adds a threshold signature to ensure IND-RCCA. The second adds verifiability to the DPRF to ensure IND-CCA.

Random Injection based approach. This approach is based on symmetric-key primitives only and achieves our RCCA security notion. The key is distributed in a t-out-of-n replicated format. In particular, $d = \binom{n}{t-1}$ random keys are shared among the parties such that any t parties together have all d keys, but any strictly smaller subset of them would fall short of at least one key. Let us call this sharing scheme combinatorial secret sharing.

Our construction requires two primitives. A random injection $\mathsf{I} : \mathcal{X} \to \mathcal{Y}$ and inverse $\mathsf{I}^{-1} : \mathcal{Y} \to \mathcal{X} \cup \{\perp\}$. Intuitively, each call to $\mathsf{I}(x)$ outputs a uniformly random element from $\mathcal{Y}$ where $|\mathcal{Y}| \gg |\mathcal{X}|$. The inverse function has an authenticity property that it is computationally hard find a y such that $\mathsf{I}^{-1}(y) \neq \perp$ and y was not computed as $y = \mathsf{I}(x)$ for some x. The second primitive is a keyed pseudo- random permutation $\mathsf{PRP} : \mathsf{PRP.kl} \times \{0,1\}^k \to \{0,1\}^k$ and inverse PRP^{-1}.

Our construction has the encryptor compute $(y_1, y_2, \ldots, y_\ell) \leftarrow y \leftarrow\!\!\$\ \mathsf{I}(m)$ where $y_i \in \{0,1\}^k$ and $\ell \geq d$. The encryptor chooses a set of $t-1$ other parties and computes $e_1 = \mathsf{PRP}_{k_1}(y_1), \ldots, e_d = \mathsf{PRP}_{k_d}(y_d)$ by sending y_i to one of them which possesses k_i. This party returns $e_i = \mathsf{PRP}_{k_1}(y_i)$ back to the encryptor. Importantly, the encryptor sends y_i only to a party that knows k_i. The final ciphertext is defined to be $c = (e_1, \ldots e_d, y_{d+1}, \ldots, y_\ell) \in \{0,1\}^{k\ell}$. During decryption, the decryptor computes all the $y_1 = \mathsf{PRP}^{-1}_{k_1}(e_1), \ldots, y_d = \mathsf{PRP}^{-1}_{k_d}(e_d)$ again by interacting with any other $t-1$ helpers in a similar fashion and then locally computes $m = \mathsf{I}^{-1}(y_1, \ldots, y_\ell)$ which could be $\perp$ if decryption fails.

The security of this scheme crucially builds on the hiding and authenticity properties of random injection. Suppose we have a ciphertext $c = (e_1, \ldots, e_d, y_{d+1}, \ldots, y_\ell)$ of either message m_0 or m_1. Since the adversary only gets to corrupt $t-1$ parties, there must be at least one key k_i that it does not know. Hence, it can only compute PRP_{k_i} via interaction with honest parties that holds this key. Connecting this to our security definition, the decryption criteria for ciphertext c is $\mathsf{Eval\text{-}MSet}(c, \mathcal{CR}) = \{(i, e_i)\}$—meaning that the adversary can trivially decrypt c if it queries e_i to some other honest parties holding key k_i. Hence, assuming that this does not happen, then the adversary have no information about $\mathsf{PRP}^{-1}(e_i)$. And, by the property of I^{-1}, the adversary should gain no information about the original message.

From a high-level, authenticity requires that given (m, y, c) computed as above (recall $y = (y_1, \ldots, y_\ell)$), one can not come up with another triple (m', y', c') without executing a legitimate encryption instance. Let us explore the options for a forgery attack: first if $m' \neq m$ then computing $y' \leftarrow\!\!\$\ \mathsf{I}(m')$ would completely change y' due to the property of I and hence either the attacker needs to predict the PRP outputs, which is hard, or it queries the honest parties for these $\mathsf{PRP}_{k_i}(y'_i)$ values, which we capture via the forgery budget. Also, keeping $m' = m$ and trying with another correctly generated $y^\star$ (since I is randomized) would not help for the same reason. Another forgery strategy could be to mix and match between several y's for which the PRP values are known from prior queries. This is where the property of I comes into play—since the ideal functionality of I generate a uniformly random output each time for any input, there cannot be should not be any collisions among any of the k-bit blocks. In other words, each y_i and e_i value corresponds to at most one message m.

A crucial property of this scheme is a clear distinction between encryption and decryption queries which, in particular, strengthen the security compared to DiSE. During encryption, the

(strong) PRP is used in the forward direction while decryption corresponds to an inverse PRP operation. It is then a relatively standard task to prove that mixing these operations would not aid the adversary either in constructing or inverting the I.

DPRF-based approach I: Achieving IND-RCCA via threshold signature Our first DPRF-based construction is based on the DiSE construction. We first briefly recall their construction. The DiSE constructions use a *distributed pseudorandom function* (DPRF) to force those encrypting or decrypting to communicate with sufficiently many participants. When encrypting a message m, a commitment $\alpha = \mathsf{Com}(m; r)$ is generated for m with commitment randomness r. The DPRF output, $\beta \leftarrow \mathsf{DP}(j\|\alpha)$, is used as an encryption key to encrypt (m, r) into ciphertext $c \leftarrow \mathsf{enc}_\beta(m\|r)$ with a symmetric-key encryption scheme. When decrypting a ciphertext (α, c), one must recompute $\beta \leftarrow \mathsf{DP}(j\|\alpha)$ and then compute $m \leftarrow \mathsf{dec}_\beta(c)$. $\mathsf{enc}, \mathsf{dec}$ here is a one-time secure encryption scheme. Note that in DiSE, the interactive part of both encryption and decryption protocols are exactly the same, which is just a DPRF query on (j, α).

We want the protocol to reject decryption of ciphertext that was not legitimately generated. We ensure this using a threshold signature scheme. In particular, while computing a $\beta \leftarrow \mathsf{DP}(j\|\alpha)$ during encryption, one also gets a threshold signature σ on $j\|\alpha$. During decryption, each party verifies the signature before responding, and aborts if the verification fails.

Even though adding a signature shouldn't affect privacy, we are proving security against a much stronger model than DiSE. In particular, we need to ensure that privacy is ensured even when the adversary can interact arbitrarily with other honest parties as well as deviate from the honest protocol. Intuitively, we need to ensure that the ability to initiate adversarial decryption sessions and the ability to initiate decryption sessions from honest parties with malicious responses, do not give the adversary extra information about any challenge ciphertext. This is done via a sequence of game hops that "trivialize" the corresponding oracles. Recall that in the standard setting, the decryption oracle can be "trivialized" by simply returning $\perp$ for all ciphertexts that were not seen before. In our setting, we need to be much more careful. First, we move to a game where α corresponds to at most one message m. For adversarially started sessions, we compare the number of valid ciphertexts generated against the *forgery budget*, $\lfloor \mathsf{ct}_{\mathsf{Eval}}/(n - |\mathcal{CR}|) \rfloor$. By the unforgeability of threshold signature, the adversary would need to contact the gap number of parties, $n - |\mathcal{CR}|$, to obtain a fresh signature. Hence, we can ensure integrity for maliciously generated ciphertext. But note that we only offer the integrity of $j\|\alpha$, and in extension, the underlying message.

For ciphertexts generated by sessions initiated by honest parties, we can ensure that the number of valid signatures is the same as the number of sessions that were executed. Moreover, the signature actually offers integrity of the value of $j\|\alpha$, hence we can achieve the notion of IND-RCCA. Using the above, we can "trivialize" the decryption oracle. The rest of the proof is more straight forward. The full detailed proof is given in Appendix **??**. Setting up for our next construction, we demonstrate why this scheme cannot achieve IND-CCA. Suppose an adversary starts an encryption session and scrambles its DPRF responses so that the honest party derives some β' to encrypt with. After learning β', the adversary would have gained enough information to know also the *correct* β if everything was executed honestly (this is because of the key-homomorphic properties [BLMR13] of the DPRFs). Hence, it can change the ciphertext $c' = \mathsf{enc}_{\beta'}(m\|r)$ to $c = \mathsf{enc}_\beta(m\|r)$, which will decrypt correctly. Whereas for IND-CCA, we would need to ensure that the generated ciphertext c' is the valid one.

DPRF based approach II: Achieving IND-CCA using DVRF. We take a natural approach to prevent the aforementioned IND-CCA attack by adding a verifiability feature to the above construction. This is achieved by adding a simple and efficient NIZK proof to each partial evaluation, similar to the *strongly-secure* DDH-based DiSE construction. In particular, this prevents an adversary from

sending a wrong partial evaluation without breaching the soundness of NIZK proofs, rendering the above attack infeasible. Notably, our formalization of DVRF differs from that of AMMR's and instead follows a recent formalization of Galindo et al. [GLOW20]. This allows us to modularize the proof: just by using a DVRF instead of DPRF in the above TAE construction, we upgrade the IND-RCCA secure scheme to an IND-CCA secure one. So, the main task of proving IND-CCA security of the upgraded construction boils down to arguing against CCA *only* attacks like above which are not RCCA.

A new analysis of DPRF. Finally, we provide a new simpler, modular and tighter analysis of the DDH-based DPRF construction of Naor et al. [NPR99]. Our analysis uses a new variant of Matrix-DDH [EHK+13] assumption, that we call Tensor-DDH assumption. This assumption captures the essence of the adversarial view of the DPRF security game (pseudorandomness) into an algebraic framework consisting of tensor products of two secret vectors (provided in the exponent). We show that irrespective of the dimensions of the vectors, this assumption is as hard as DDH with a minimal security loss of a factor 2. In a similar reduction step, AMMR's proof looses a factor that scales with the number of evaluation and challenge queries. [1]Our analysis provides better concrete security and may be of independent interest.

2.3 Related work

Threshold cryptography. Starting with the work of Desmedt [Des88], most work on threshold cryptography has focused on public-key encryption and digital signatures [DF90, BD10]. Starting from Micali and Sidney [MS95], some work has focused on threshold and distributed PRFs [NPR99, BLMR13, Dod03]. However, the pseudo-randomness requirements do not explicitly take into account several avenues of attacks. Agrawal et al. [AMMR18a] propose stronger notions for distributed DPRFs and build on the constructions of Naor et al. [NPR99] to achieve them.

Threshold Oblivious PRF. Another related notion is distributed/threshold oblivious PRF (TOPRF) [FIPR05], which can be thought of as a DPRF, but with an additional requirement of hiding input from the servers. This requirement makes TOPRF a stronger primitive, which is known to imply oblivious transfer [JL09]. Furthermore, despite the structural similarities between the DDH-based TOPRF [AMMM18, JKKX17] and the DDH-based DVRF [JKKX17] we used here, the TOPRF is proven assuming interactive variant of DDH, in contrast to the DVRF which can be reduced to DDH. Therefore, the proof techniques are also quite different.

Authenticated encryption. Authenticated encryption (AE) has seen a signficant amount research since being identified as a primitive worthy of study in its own right [BN00, KY01]. AE research has increasingly addressed practical concerns from a performance, security, and usability point-of-view. AE schemes evolved from a generic composition of encryption and authentication schemes [BN00] to dedicated schemes like GCM [MV04]. Initially the description of AE schemes did not match with how they were used in practice, leading to the introduction of nonces [Rog04] and associated data [Rog02]. Further security concerns with how AE schemes are used in practice lead to formalization of different settings and properties, such as varying degrees of nonce-misuse resistance and deterministic AE [RS06, FFL12], blockwise adaptive security [FJMV04], online authenticated encryption [BBKN12, RZ11], leakage concerns such as unverified plaintext and robustness against it [ABL+14, HKR15, BMOS17], and multi-user security [BT16]. Recently, Barbosa and Farshim proposed indifferentiable authenticated encryption [BF18], which has many of the properties of the all-or-nothing transform introduced by Rivest [Riv97]. In Section 5.1 we discuss how we use these

[1]We remark that the number of queries actually translate to the dimensions of the secret vectors. Our gain in security follows from the independence of the reduction from the dimensions.

primitves to create one of our TAE schemes.

Multi-party computation. MPC allows a set of mutually distrustful parties to evaluate a joint function of their inputs without revealing anything more than the function's output. The last 10-15 years have seen tremendous progress in bringing MPC closer to practice. The performance of MPC protocols has improved by several order of magnitude with the introduction of new techniques and optimizations.

General-purpose MPC protocols can help parties evaluate any function of their choice but they work with a circuit representation of the function, which leads to a large communication/computation complexity—typically, at least linear in the size of the circuit and the number of parties. Moreover, the parties have to engage in several rounds of communication, with every party talking to every other. (Some recent results reduce the round complexity but have substantially higher computational overhead.) So, general-purpose MPC protocols are not ideal for making a standard AE scheme like AES-GCM distributed. Nonetheless, such a solution would be fully compatible with the standards.

Adaptive DiSE. A recent work [Muk20] defined and constructed stronger TAE schemes that are secure against *adaptive* corruption, in contrast we only focus on static corruption. Nevertheless, the definitions considered in that work is based on the AMMR definitions and hence suffers from similar limitations. Our work can be perceived as orthogonal to that. Augmenting our definitional framework into adaptive setting and constructing secure scheme therein may be an interesting future work.

3 Preliminaries

For a positive integer n, let $[n]$ denote the set $\{1, \ldots, n\}$. For a finite set S, we use $x \leftarrow\!\!\$\ S$ to denote the process of sampling an element uniformly from S and assigning it to x. We $\uplus$ to denote the union of multi-sets.

For example, $\{\{1, 1, 3\}\}$ is the multiset containing 1 with multiplicity 2 and 3 with multiplicity 1.

We use $s \| t$ to denote the concatenation of strings a and b. For a set $\mathcal{X}$, we let $\mathbf{x} \in \mathcal{X}^n$ denote vectors consisting of n components from $\mathcal{X}$, with $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. We let ε denote a special "null value", representing absence of an element from a set. For example, we could write $\mathbf{x} \in \mathcal{X}^3$ with $\mathbf{x} = (x_1, \varepsilon, x_3)$. We write $|\mathbf{x}|$ for the number of components of $\mathbf{x}$, and $\|\mathbf{x}\|$ denotes the number of non-ε elements in $\mathbf{x}$. For $\mathbf{x} = (x_1, \varepsilon, x_3)$, $|\mathbf{x}| = 3$ and $\|\mathbf{x}\| = 2$. We assume $\perp$ is not contained in any set.

Let $\mathcal{A}$ be a randomized algorithm. We use $y \leftarrow\!\!\$\ \mathcal{A}(x_1, x_2, \ldots)$ to denote running $\mathcal{A}$ with inputs $x_1, x_2, \ldots$ and assigning its output to variable y. We use the notation $[\![\mathsf{sk}]\!]_n$ to denote the tuple $(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n)$. We assume that variables for strings, sets, numbers are initialized to the empty string, the empty set, and zero, respectively. We identify 1 with True and 0 with False.

We borrow some notation from Agrawal et al. [AMMR18a]. We use $[j : x]$ to denote that the value x is private to party j. For a protocol Π, we write $(c', [j : z']) \leftarrow \Pi(c, [\![\mathbf{k}]\!]_n, [i : (x, y)], [j : z])$ to denote that all parties have a common input c, party ℓ has private input k_ℓ (for all $\ell \in [n]$, this is usually the secret key), party i has two private inputs x and y, and party j has one private input z. After the execution, all parties receive an output c' and j additionally receives z'.

Threshold protocols. A threshold protocol, Π, is defined with respect to $n \in \mathbb{N}$ parties that are labeled from 1 to n. We use t to denote the threshold. At least t parties are needed to execute the scheme successfully. Formally, a protocol is specified by a setup algorithm Setup, a participant algorithm E and an initiator algorithm L. The setup algorithms takes inputs integer n (and possibly more parameters) to return n secret keys $[\![\mathsf{sk}]\!]_n$ and a public parameter pp. Fix some secret keys

$[\![\mathsf{sk}]\!]_n$ and public parameter pp. The participant algorithm E defines n participant oracles $\mathcal{P}_i.\mathsf{E}(\cdot) := \mathsf{E}(\mathsf{pp}, \mathsf{sk}_i, \cdot)$ for each $i \in [n]$. The initiator algorithm L is an algorithm with oracle access to $\mathcal{P}_i.\mathsf{E}(\cdot)$ for each $i \in [n]$ (including $i = j$). We write $y \leftarrow\!\!\$\ \mathsf{L}(\mathsf{pp}, [\![\mathsf{sk}]\!]_n, [j : x])$ to denote the execution of the protocol under public parameter pp, secret keys $[\![\mathsf{sk}]\!]_n$, with party $j \in [n]$ being the initiator taking (private) input x and producing (private) output y.

Security games. Every security game is defined with respect to a protocol Π and an adversary $\mathcal{A}$. Adversary $\mathcal{A}$ gets access to several procedures in the game. When Π is a threshold protocol, we assume that $\mathcal{A}$ consists of two stages $(\mathcal{A}_0, \mathcal{A}_1)$. The first stage adversary $\mathcal{A}_0$ takes input (pp, n, t) and produces some set $C \subset [n]$ with $|C| < t$. The second stage adversary $\mathcal{A}_1$ receives the list of secret keys for parties in C, i.e. $(\mathsf{sk}_i)_{i \in C}$, and access to the procedures defined by the security game. We write $\mathcal{A}_1^{\langle \mathbf{Proc} \rangle}$ to denote the execution of $\mathcal{A}_1$ where $\mathcal{A}_1$ has access to all the available game oracles.

For a game $\mathbf{G}$ with a protocol Π and an adversary $\mathcal{A}$, we use $\mathbf{P}\big[\mathbf{G}_\Pi(\mathcal{A})\big]$ to denote the probability that $\mathbf{G}$ outputs 1. Throughout the paper, n denotes the total number of parties and t the threshold. We define $\Delta_{\mathcal{A}}(O_1 ; O_2) := |\mathbf{P}\big[\mathcal{A}^{O_1} = 1\big] - \mathbf{P}\big[\mathcal{A}^{O_2} = 1\big]|$, where $\mathcal{A}^O$ denotes $\mathcal{A}$'s output after interacting with oracle O.

Random oracle model We work with random oracle model implicitly. In particular, any scheme Π could depend on a hash function H that is modeled as a random oracle, with input and output spaces implicitly defined by the scheme specifications. We require scheme Π to work given any particular H with the correct input and output spaces. It is understood that any correctness definition about Π is stated for all possible hash functions H. It is understood that in security games involving Π, say $\mathbf{G}_\Pi(\mathcal{A})$ with some adversary $\mathcal{A}$, the security game will sample a hash function H uniformly at random from the space of functions with the correct input and output spaces at the beginning of the game. Moreover, the sampled hash function will be exposed as an oracle both to the scheme Π (and all its constituent algorithms) and the adversary $\mathcal{A}$.

Threshold Signature. A threshold signature scheme allows for a signing key to be secret shared among n parties such that any t parties can collectively generate a signature. On a common message m, t parties call $\mathsf{PartSign}(\mathsf{sk}_i, m) \to \sigma_i$. The σ_i can then be collected and used to produce a signature $\mathsf{CombSig}(\mathsf{vk}, \{(i, \sigma_i)\}_{i \in S}) \to \sigma$. Signature verification is non-interactive and is performed as $\mathsf{VerSig}(\mathsf{vk}, m, \sigma)$.

A threshold signature scheme SIG consists of:

- $\mathsf{Setup}(n, t) \xrightarrow{\$} ([\![\mathsf{sk}]\!]_n, \mathsf{vk})$, where n, t are integers s.t. $1 \le t \le n$. Setup generates n signing key shares and a public verification key vk.
- $\mathsf{PartSign}(\mathsf{sk}_i, m) \to \sigma_i$, where $m \in \mathcal{M}$, the message space. $\mathsf{PartSign}$ generates a partial signature share on message m using signing key share sk_i.
- $\mathsf{CombSig}(\mathsf{vk}, \{(i, \sigma_i)\}_{i \in S}) \to \sigma$. The algorithm $\mathsf{Combine}$ combines the signature shares σ_i for $i \in S$, and returns a full signature or $\perp$.
- $\mathsf{VerSig}(\mathsf{vk}, m, \sigma) \to b$. The algorithm VerSig takes the verification key vk, a message $m \in \mathcal{M}$ and a signature σ to return a decision bit $b \in \{0, 1\}$, with 1 denoting that the signature is valid and 0 otherwise.

A threshold signature scheme SIG is *correct* if for any n, t with $1 \le t \le n$, for all $m \in \mathcal{M}$, $([\![\mathsf{sk}]\!]_n, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(n, t)$, and $S \subseteq [n]$ of size at least t, $\mathsf{VerSig}(\mathsf{vk}, m, \sigma) = 1$, where $\sigma \leftarrow\!\!\$\ \mathsf{CombSig}(\mathsf{vk}, \{(i, \mathsf{PartSign}(\mathsf{sk}_i, m))\}_{i \in S})$.

We also require the scheme to produce *unique* signatures, i.e. for any $m \in \mathcal{M}$, there does not exist σ_1, σ_2 s.t. $\sigma_1 \neq \sigma_2$ but both $\mathsf{VerSig}(\mathsf{vk}, m, \sigma_1)$ and $\mathsf{VerSig}(\mathsf{vk}, m, \sigma_1)$ output 1.

Game $\mathbf{G}^{\text{sig}}_{\mathsf{SIG},n,t}(\mathcal{A})$

$(\llbracket \mathsf{sk} \rrbracket_n, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(n,t)$
$(C, \mathsf{st}_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}_0(\mathsf{vk})$
$(m, \sigma) \leftarrow\!\!\$\ \mathcal{A}_1^{\langle \mathbf{Proc} \rangle}(\mathsf{st}_{\mathcal{A}}, (\mathsf{sk}_i)_{i \in C})$
Return $(\mathsf{VerSig}(\mathsf{vk}, m, \sigma) = 1) \wedge (\mathsf{ct}_m < t - |C|)$

Proc PartSign(i, m)

$\mathsf{ct}_m \leftarrow \mathsf{ct}_m + 1$
Return $\mathsf{PartSign}(\mathsf{sk}_i, m)$

Figure 1: Unforgeability game.

To capture *unforgeability*, consider the game $\mathbf{G}^{\text{sig}}$ given in Figure 1. The adversary $\mathcal{A}$ can corrupt up to $t-1$ parties, then gets access to the partial signing oracle for each party. It wins the game if it can forge a signature on a message m for which it has called fewer than $t - |C|$ partial signatures; ct_m denotes the number of partial signatures computed for message m. The advantage of adversary $\mathcal{A}$ against signature scheme SIG is

$$\mathbf{Adv}^{\text{sig}}_{\mathsf{SIG},n,t}(\mathcal{A}) = \mathbf{P}\left[\mathbf{G}^{\text{sig}}_{\mathsf{SIG},n,t}(\mathcal{A})\right] . \tag{1}$$

We take a concrete-security approach, we will define the advantage functions for security games with the understanding that the scheme is secure when the advantage is small enough for some set of adversaries.

4 Threshold Authenticated Encryption

4.1 Syntax

A threshold authenticated encryption scheme TAE consists of a setup algorithm and protocols for encryption and decryption. Throughout, we let H denote a random oracle that the algorithms can use. Parameter n denotes the number of parties involved in the protocol and parameter t denotes the *threshold* of the protocol. We use the shorthand $\mathcal{P}_i.\mathsf{E}(x, y, \ldots)$ to refer to party i running some algorithm $\mathsf{E}(\mathsf{pp}, \mathsf{sk}_i, \cdot)$ and returning the result.

Setup: takes integers n, t with $1 \le t \le n$, and generates n secret key shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ and public parameters pp, denoted $(\llbracket \mathsf{sk} \rrbracket_n, \mathsf{pp}) \leftarrow\!\!\$\ \mathsf{Setup}(n,t)$.

Encryption: a 2-round protocol, consisting of three algorithms $(\mathsf{Split}_{\mathsf{enc}}, \mathsf{Eval}_{\mathsf{enc}}, \mathsf{Combine}_{\mathsf{enc}})$. For an initiating party $j \in [n]$, input m, and set $S \subseteq [n]$, the Enc protocol is executed as follows:

Protocol $\mathsf{Enc}(\mathsf{pp}, \llbracket \mathsf{sk} \rrbracket_n, [j : m, S])$

$(L, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{P}_j.\mathsf{Split}^{\mathrm{H}}_{\mathsf{enc}}(m, S)$
For $(i,x) \in L$ do $r_{i,x} \leftarrow\!\!\$\ \mathcal{P}_i.\mathsf{Eval}^{\mathrm{H}}_{\mathsf{enc}}(j, x)$
$c \leftarrow\!\!\$\ \mathsf{Combine}^{\mathrm{H}}_{\mathsf{enc}}(\{r_{i,x}\}_{(i,x)\in L}, \mathsf{st})$
Return c

Protocol $\mathsf{Dec}(\mathsf{pp}, \llbracket \mathsf{sk} \rrbracket_n, [j : c, S])$

$(L, \mathsf{st}) \leftarrow \mathcal{P}_j.\mathsf{Split}^{\mathrm{H}}_{\mathsf{dec}}(c, S)$
For $(i,x) \in L$ do $r_{i,x} \leftarrow \mathcal{P}_i.\mathsf{Eval}^{\mathrm{H}}_{\mathsf{dec}}(j, x)$
$m \leftarrow \mathsf{Combine}^{\mathrm{H}}_{\mathsf{dec}}(\{r_{i,x}\}_{(i,x)\in L}, \mathsf{st})$
Return m

The list L contains tuples of the form (i,x), indicating that message x should be sent to party i for evaluation. Since we assume all parties communicate over authenticated channels, receivers will know the identity of the sender, hence the sending party index j is an input to the evaluation for each receiving party i.

We assume that S always contains the index j. We note that the size of the set L output by $\mathsf{Split}_{\mathsf{enc}}$ (and by $\mathsf{Split}_{\mathsf{dec}}$ below) does not have to match the size of S. This allows multiple messages to be sent to the same party for evaluation.

Decryption: a 2-round protocol, consisting of three deterministic algorithms ($\mathsf{Split}_{\mathsf{dec}}$, $\mathsf{Eval}_{\mathsf{dec}}$, $\mathsf{Combine}_{\mathsf{dec}}$). For an initiating party $j \in [n]$, input c, and set $S \subseteq [n]$, the Dec protocol is executed shown above. We define the finite sets X, Y as $\mathcal{P}_i.\mathsf{Eval}_{\mathsf{dec}} : X \to Y$.

Basic correctness We say that TAE satisfies basic correctness if for all positive integers n, t such that $t \leq n$, all $(\llbracket \mathsf{sk} \rrbracket_n, \mathsf{pp}) \leftarrow\!\!\$\ \mathsf{Setup}(n, t)$, any $m \in \{0,1\}^*$, any $S, S' \subseteq [n]$ with $|S|, |S'| \geq t$, and any $i \in S$, $j \in S'$, we have that $m = m'$ where

$$c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pp}, \llbracket \mathsf{sk} \rrbracket_n, [i : m, S])\ , m' \leftarrow \mathsf{Dec}(\mathsf{pp}, \llbracket \mathsf{sk} \rrbracket_n, [j : c, S'])\ .$$

4.2 Decryption criteria

As is common with CCA-style security games, our games need to prevent the "trivial win" where an adversary decrypts a challenge ciphertext simply by executing decryption as specified by the TAE scheme. What complicates preventing such trivial wins in our setting is the fact that there are many ways to decrypt since basic correctness requires that any group of t out of n parties may decrypt. Furthermore, not only can adversaries corrupt up to $t-1$ parties and recover their secret keys, but they can also ask honest parties to run $\mathsf{Eval}_{\mathsf{dec}}$.

Given a ciphertext, our goal is to detect when an adversary has collected $\mathsf{Eval}_{\mathsf{dec}}$ output from at least t parties, either through an $\mathsf{Eval}_{\mathsf{dec}}$ query to an honest party or via a corrupted party's key, so that it could decrypt the ciphertext as specified by the protocol. If we can detect such events, then we can rule out trivial decryptions of challenge ciphertexts.

Let $(\llbracket \mathsf{sk} \rrbracket_n, \mathsf{pp}) \leftarrow\!\!\$\ \mathsf{Setup}(n, t)$, let $S \subseteq [n]$ be a set of parties of size at least t, and $j \in S$ an initiator. When decrypting a ciphertext c, the initiator $\mathcal{P}_j$ first splits c into inputs for the other parties in S: $(L, \mathsf{st}) \leftarrow \mathcal{P}_j.\mathsf{Split}_{\mathsf{dec}}(c, S)$. Then, $\mathcal{P}_j$ sends x to $\mathcal{P}_i$ for evaluation for every $(i, x) \in L$.

Although $\mathsf{Split}_{\mathsf{dec}}(c, S)$ might assign x to $\mathcal{P}_i$, depending on the TAE scheme, there might be other parties with the key material to evaluate x. An adversary $\mathcal{A}$ can ask any party — including corrupt ones — to evaluate x, and is not restricted to the one prescribed in L. Although the corrupt parties can evaluate some of the x in L, a secure TAE scheme will require the adversary to interact with honest parties to evaluate what it cannot. If these x's are queried to honest parties, then $\mathcal{A}$ has enough information to execute the decryption protocol on c.

Without further details about how a given scheme TAE works, the only way to know whether TAE allows $\mathcal{P}_i$ to evaluate x, is to find a set S with $i \in S$ such that $(i, x) \in \mathcal{P}_j.\mathsf{Split}_{\mathsf{dec}}(c, S)$. Instead, we approach this as follows:

1. We require that for all S and $j \in S$, $\mathcal{P}_j.\mathsf{Split}^{\mathrm{H}}_{\mathsf{dec}}(c, S)$ always outputs the same multiset of messages, $\mathsf{Eval\text{-}MSet}^{\mathrm{H}}(c) := \{x\}_{(i,x)\in L}$. In other words, although the assignment to parties could change with S, the set of messages x and their multiplicity stays the same.

2. We assume that a party i can evaluate x if its execution of $\mathsf{Eval}_{\mathsf{dec}}$ produces *any* valid output. Formally, we define a relation $\mathcal{R}^{\mathrm{H}} \subseteq [n] \times X$ where $(i, x) \in \mathcal{R}^{\mathrm{H}} \iff \mathcal{P}_i.\mathsf{Eval}^{\mathrm{H}}_{\mathsf{dec}}(j, x) \neq \bot$.

With the above two assumptions, given a ciphertext c and a party i, we can determine the values that $\mathcal{P}_i$ can evaluate:

$$\left\{ x \in \mathsf{Eval\text{-}MSet}^{\mathrm{H}}(c) \mid \mathcal{R}^{\mathrm{H}}(i, x) \right\}. \tag{2}$$

We are now ready to define the *gap* set of messages, i.e. the messages an attacker cannot evaluate on its own. For a ciphertext c and a set of corrupt parties $\mathcal{CR}$,

$$\mathsf{Eval\text{-}MSet}^{\mathrm{H}}(c, \mathcal{CR}) = \mathsf{Eval\text{-}MSet}^{\mathrm{H}}(c) \setminus \left(\biguplus_{i \in \mathcal{CR}} \left\{ x \in \mathsf{Eval\text{-}MSet}^{\mathrm{H}}(c) \mid \mathcal{R}^{\mathrm{H}}(i, x) \right\} \right), \tag{3}$$

where $\biguplus$ and $\setminus$ denote union and set-difference over multisets.

Let us take a simple example. Suppose we have a threshold of 3, $S = \{2, 5, 7\}$, and $\mathcal{P}_j.\mathsf{Split}^{\mathrm{H}}_{\mathsf{dec}}(c, S)$ outputs $L = \{(2, \text{“}\mathtt{m1}\text{”}), (5, \text{“}\mathtt{m1}\text{”}), (7, \text{“}\mathtt{m1}\text{”}), (2, \text{“}\mathtt{m2}\text{”}), (5, \text{“}\mathtt{m3}\text{”}), (7, \text{“}\mathtt{m3}\text{”})\}$. Let us consider a rather peculiar $\mathcal{P}_i.\mathsf{Eval}^{\mathrm{H}}_{\mathsf{dec}}(j, x)$ function which has non-$\perp$ output if $x \in \{\text{“}\mathtt{m1}\text{”}, \text{“}\mathtt{m2}\text{”}, ..., \text{“}\mathtt{m}i\text{”}\}$. This in turn similarly defines $\mathcal{R}^{\mathrm{H}}$. Suppose $\mathcal{A}$ corrupts parties $\mathcal{CR} = \{1, 2\}$.

First of all, $\mathsf{Eval\text{-}MSet}^{\mathrm{H}}(c) = \{\text{“}\mathtt{m1}\text{”}, \text{“}\mathtt{m1}\text{”}, \text{“}\mathtt{m1}\text{”}, \text{“}\mathtt{m2}\text{”}, \text{“}\mathtt{m3}\text{”}, \text{“}\mathtt{m3}\text{”}\}$. Party $\mathcal{P}_i$ with $i = 2$ can evaluate $\{x \in \mathsf{Eval\text{-}MSet}^{\mathrm{H}}(c) \mid \mathcal{R}^{\mathrm{H}}(i, x)\} = \{\text{“}\mathtt{m1}\text{”}\}$ and $i = 2$ can evaluate $\{\text{“}\mathtt{m1}\text{”}, \text{“}\mathtt{m2}\text{”}\}$. Thus, the set of messages that can not be evaluated by the corrupt parties is $\mathsf{Eval\text{-}MSet}^{\mathrm{H}}(c, \mathcal{CR}) = \{\text{“}\mathtt{m1}\text{”}, \text{“}\mathtt{m3}\text{”}, \text{“}\mathtt{m3}\text{”}\}$ as per Equation 3.

$\mathcal{A}$ could get a “m1” message evaluated by $\mathcal{P}_1$ even though L does not prescribe to do so. On the other hand, L indicates that “m1” messages need to be evaluated by three different parties (under their respective keys) while $\mathcal{A}$ has only two under its control. Moreover, none of $\mathcal{P}_1$, $\mathcal{P}_2$ can evaluate “m3”. Thus, one can see that $\mathsf{Eval\text{-}MSet}^{\mathrm{H}}(c, \mathcal{CR})$ captures the messages $\mathcal{A}$ cannot process on its own.

4.3 Security

We give two security notions for TAE. First is the IND-RCCA notion, which mirrors the IND-RCCA security for PKE. This notion is relaxed from the standard IND-CCA by targeting the integrity of *plaintext*. Second is the IND-CCA notion, which mirrors the standard IND-CCA notion for standard PKE.

Consider the security game $\mathbf{G}^{\mathrm{ind-rcca}}_{\mathsf{TAE},n,t}$, given in Figure 2. The goal of the adversary is to either predict the bit b, or generate enough valid decryptions so that the flag forgery is set to TRUE. Several global variables keep track of the winning condition of the adversary: $\mathsf{ct}_{\mathsf{Eval}}$, EncCtxt, ChlCtxt, DecCtxt, and Q_{dec}.

- The counter $\mathsf{ct}_{\mathsf{Eval}}$ counts the number of times $\mathsf{Eval}_{\mathsf{enc}}$ is called. Note that calling $\mathsf{Eval}_{\mathsf{enc}}$ on honest parties helps the adversary to generate ciphertexts.
- The set EncCtxt contains the ciphertexts generated by challenge encryption processes where $m_0 = m_1$ which the adversary obtained via GETCTXT.
- The set ChlCtxt contains the ciphertexts generated by challenge encryption processes where $m_0 \neq m_1$ which the adversary obtained via GETCTXT.
- The set EncMsg contains the set of all encrypted messages (including the challenge messages). This is only used in the RCCA game.
- The set DecSet contains the *valid* and *fresh* decryptions that the adversary can generate. Valid means that the ciphertext c must decrypt correctly. For IND-RCCA, fresh means that ciphertext c decrypts to message $m \neq \perp$ than was not in the set EncMsg. For IND-CCA, fresh means that ciphertext c is not a ciphertext in either EncCtxt or ChlCtxt.
- Q_{dec} is a *multiset* containing all the queries to $\mathsf{Eval}_{\mathsf{dec}}$ made by the adversary.

Game $\mathbf{G}^{\mathrm{ind-cca}}_{\mathsf{TAE},n,t}(\mathcal{A})$

$b \leftarrow\!\!\$\ \{0,1\}$
$(\llbracket \mathsf{sk} \rrbracket, \mathsf{pp}) \leftarrow\!\!\$\ \mathsf{TAE.Setup}(n,t)$
$(\mathcal{CR}, \mathsf{st}_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}_0(\mathsf{pp})$
$b' \leftarrow\!\!\$\ \mathcal{A}_1^{\langle \mathbf{Proc} \rangle}(\mathsf{st}_{\mathcal{A}}, (\mathsf{sk}_k)_{k \in \mathcal{CR}})$
If $|\mathsf{DecSet}| > \left\lfloor \frac{\mathsf{ct}_{\mathsf{Eval}}}{t - |\mathcal{CR}|} \right\rfloor$ then $\mathsf{forgery} \leftarrow \textsc{True}$
If $(\exists c \in \mathsf{ChlCtxt} : \mathsf{Eval\text{-}MSet}(c, \mathcal{CR}) \subseteq Q_{\mathsf{dec}})$
  Return $(b'' \leftarrow\!\!\$\ \{0,1\}) \vee \mathsf{forgery}$
Return $(b = b') \vee \mathsf{forgery}$

Challenge encryption sessions

Proc $\textsc{Split}_{\mathsf{enc}}(\mathsf{id}, m_0, m_1, S)$
Require $\mathsf{id} \notin \mathcal{CR}$, $|m_0| = |m_1|$
$u \leftarrow u + 1$; $\mathsf{id}_u \leftarrow \mathsf{id}$
$m_{u,0} \leftarrow m_0$; $m_{u,1} \leftarrow m_1$
$(L_u, \mathsf{st}_u) \leftarrow\!\!\$\ \mathsf{Split}_{\mathsf{enc}}(\mathsf{sk}_{\mathsf{id}}, m_{u,b}, S)$
Return $\{(k,x) \in L_u \mid k \in \mathcal{CR}\}$

Proc $\textsc{Combine}_{\mathsf{enc}}(u, \mathsf{rsp})$
For $(k,x) \in L_u$ with $k \notin \mathcal{CR}$ do
  $\mathsf{rsp}[(k,x)] \leftarrow\!\!\$\ \mathsf{Eval}_{\mathsf{enc}}(\mathsf{sk}_k, \mathsf{id}_u, x)$
$c_u \leftarrow\!\!\$\ \mathsf{Combine}_{\mathsf{enc}}(\mathsf{rsp}[L_u], \mathsf{st}_u)$
If $m_{u,0} = m_{u,1}$ then
  $\mathsf{EncCtxt} \leftarrow \mathsf{EncCtxt} \cup \{c_u\}$
  RCCA: $\mathsf{EncMsg} \leftarrow \mathsf{EncMsg} \cup \{m_{u,0}\}$
Else
  $\mathsf{ChlCtxt} \leftarrow \mathsf{ChlCtxt} \cup \{c_u\}$
  RCCA: $\mathsf{ChlMsg} \leftarrow \mathsf{ChlMsg} \cup \{m_{u,0}, m_{u,1}\}$
Return c_u

Sessions initiated by adversary

Proc $\textsc{Eval}_{\mathsf{enc}}(\mathsf{eid}, \mathsf{id}, x)$
Require $\mathsf{eid} \notin \mathcal{CR}$, $\mathsf{id} \in \mathcal{CR}$
$\mathsf{ct}_{\mathsf{Eval}} \leftarrow \mathsf{ct}_{\mathsf{Eval}} + 1$
Return $\mathsf{Eval}_{\mathsf{enc}}(\mathsf{sk}_{\mathsf{eid}}, \mathsf{id}, x)$

Proc $\textsc{Eval}_{\mathsf{dec}}(\mathsf{eid}, \mathsf{id}, x)$
Require $\mathsf{eid} \notin \mathcal{CR}$, $\mathsf{id} \in \mathcal{CR}$
$Q_{\mathsf{dec}} \leftarrow Q_{\mathsf{dec}} \uplus \{x\}$
Return $\mathsf{Eval}_{\mathsf{dec}}(\mathsf{sk}_{\mathsf{eid}}, \mathsf{id}, x)$

Decryption sessions

Proc $\textsc{Split}_{\mathsf{dec}}(\mathsf{id}, c, S)$
Require $\mathsf{id} \notin \mathcal{CR}$
CCA: Require $c \notin \mathsf{ChlCtxt}$
$v \leftarrow v + 1$; $\mathsf{id}_v \leftarrow \mathsf{id}$; $c_v \leftarrow c$
$(L_v, \mathsf{st}_v) \leftarrow \mathsf{Split}_{\mathsf{dec}}(\mathsf{sk}_{\mathsf{id}}, c, S)$
Return $\{(k,x) \in L_v \mid k \in \mathcal{CR}\}$

Proc $\textsc{Combine}_{\mathsf{dec}}(v, \mathsf{rsp})$
For $(k,x) \in L_v$ with $k \notin \mathcal{CR}$ do
  $\mathsf{rsp}[(k,x)] \leftarrow\!\!\$\ \mathsf{Eval}_{\mathsf{dec}}(\mathsf{sk}_k, \mathsf{id}_v, x)$
$m_v \leftarrow\!\!\$\ \mathsf{Combine}_{\mathsf{dec}}(\mathsf{rsp}[L_v], \mathsf{st}_v)$
RCCA: Require $m_v \notin \mathsf{ChlMsg}$
RCCA: $\mathsf{fresh} \leftarrow (m_v \notin \mathsf{EncMsg})$
CCA: $\mathsf{fresh} \leftarrow (c_v \notin \mathsf{EncCtxt})$
If $m_v \neq \perp$ and fresh then
  RCCA: $\mathsf{DecSet} \leftarrow \mathsf{DecSet} \cup \{m_v\}$
  CCA: $\mathsf{DecSet} \leftarrow \mathsf{DecSet} \cup \{c_v\}$
Return m_v

Figure 2: IND-RCCA & IND-CCA games for Threshold Authenticated Encryption.

We say that the adversary trivially breaks privacy if

$$\exists c \in \mathsf{ChlCtxt} : \mathsf{Eval\text{-}MSet}(c, C) \subseteq Q_{\mathsf{dec}} \ . \tag{4}$$

We also refer to the above check as the *privacy condition*. Intuitively, this captures when the adversary has enough information, through calling $\mathsf{Eval}_{\mathsf{dec}}$, to decrypt a challenge ciphertext c. In this case, the adversary is essentially guaranteed to be able to predict the bit b by correctness of the encryption scheme. Similarly, for authenticity, we need to set aside a budget for the amount of ciphertexts that the adversary can generate herself via invoking $\mathsf{Eval}_{\mathsf{enc}}$ of honest parties. This is captured by the following *authenticity condition*

$$|\mathsf{DecCtxt}| > \left\lfloor \frac{\mathsf{ct}_{\mathsf{Eval}}}{n - |\mathcal{CR}|} \right\rfloor \ . \tag{5}$$

We say that $\mathcal{A}$ breaks authenticity if the above line is true (which will set $\mathsf{forgery}$ to $\textsc{True}$ in the game). We now describe the interfaces exposed to $\mathcal{A}_1$.

Sessions initiated by adversary. The adversary can initiate encryption and decryption sessions from corrupt parties. These are achieved via calling $\mathsf{Eval}_{\mathsf{enc}}$ and $\mathsf{Eval}_{\mathsf{dec}}$ on honest parties. Note that the adversary can run Split and $\mathsf{Combine}$ itself. Counters $\mathsf{ct}_{\mathsf{Eval}}$ is incremented every time $\mathsf{Eval}_{\mathsf{enc}}$ is invoked. Multiset Q_{dec} contains the list of all queries made to $\mathsf{Eval}_{\mathsf{dec}}$ (with $\mathsf{id}, \mathsf{eid}$ removed).
Challenge encryptions. The adversary can initiate challenge encryptions by calling $\textsc{Split}_{\mathsf{enc}}$ and $\textsc{Combine}_{\mathsf{enc}}$. This is done via keeping track of a session identifier u. The adversary first needs to ask the desired honest party to initiate a session via calling $\textsc{Split}_{\mathsf{enc}}$ with some message m_0 and m_1 as well as set $S \subseteq [n]$ (we require that $|m_0| = |m_1|$). Oracle $\textsc{Split}_{\mathsf{enc}}$ will call $\mathsf{Split}_{\mathsf{enc}}(\mathsf{id}, m_b, S)$ and return a session id u, which the adversary needs to supply to $\textsc{Combine}_{\mathsf{enc}}$. Via calling $\textsc{Combine}_{\mathsf{enc}}$ with some session id, the adversary can supply a set of corrupt eval responses, which are used instead of the honest ones for corrupt parties. Any session id u can be input to $\textsc{Combine}_{\mathsf{enc}}$ at most once.
Decryptions. The adversary is able to submit ciphertext to honest parties and initiate decryptions sessions. The adversary's goal here is to submit all the *valid* ciphertext that it can generate. Similar to challenge encryption, the adversary first needs to specify (to $\textsc{Split}_{\mathsf{dec}}$ oracle) an honest party id, a ciphertext c, and a set S of parties that are chosen to participate in the decryption process. The oracle shall return to the adversary the set of messages (k, x) designated to the corrupt parties k. Next, the adversary can choose to reply with any corrupt responses to these messages via the $\textsc{Combine}_{\mathsf{dec}}$ oracle.

We define the IND-RCCA and IND-CCA advantage of an adversary $\mathcal{A}$ against TAE to be $\mathbf{Adv}^{\text{ind-cca}}_{\mathsf{TAE},n,t}(\mathcal{A}) = 2 \cdot \mathbf{P}\left[\mathbf{G}^{\text{ind-cca}}_{\mathsf{TAE},n,t}(\mathcal{A})\right] - 1$, and $\mathbf{Adv}^{\text{ind-rcca}}_{\mathsf{TAE},n,t}(\mathcal{A}) = 2 \cdot \mathbf{P}\left[\mathbf{G}^{\text{ind-rcca}}_{\mathsf{TAE},n,t}(\mathcal{A})\right] - 1$, respectively.
Comparison with security model of DiSE. First, the DiSE model allows the adversary to generate valid ciphertext by engaging in the decryption protocol. Our model prevents this by not including the queries to $\mathsf{Eval}_{\mathsf{dec}}$ in the definition of the authenticity condition Equation 5. Second, our security notions allow for malicious adversaries during decryption, whereas DiSE required all parties to behave honestly during decryption. This is a significant strengthening of the model. Third, DiSE targeted privacy and authenticity separately, which allowed for schemes that reconstruct the (master) encryption scheme during decryption. We give such an example (in Appendix B) which is secure in the model of DiSE but not secure according to our IND-CCA notion.

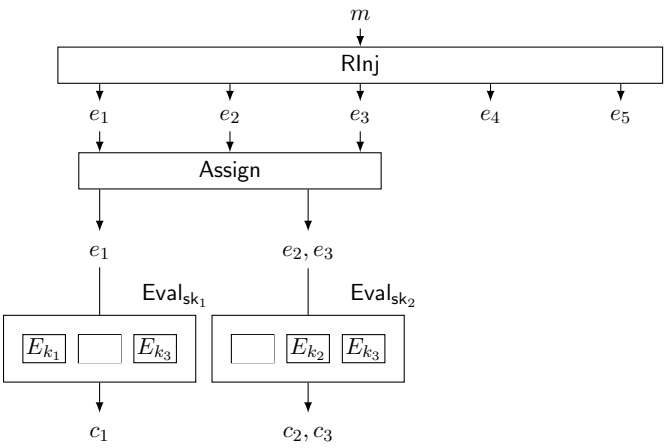
Finally, there are some subtle differences around what is considered a forgery and how they can be constructed. In DiSE, when an honest party initiates an encryption the resulting ciphertext is not revealed to the adversary in the authenticity game. However, it could be possible for the adversary to take such a ciphertext and generate another which decrypts properly. In fact, Appendix C discusses this exact scenario. Our definition prevents such attacks by explicitly providing ciphertext generated by honest parties to the adversary.

5 Construction from Indifferential AE

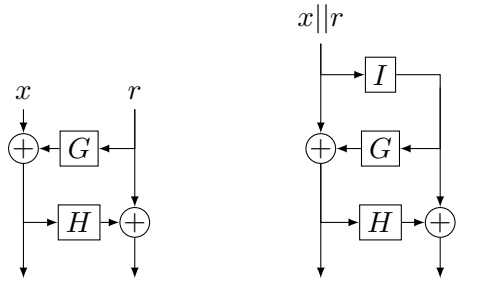
In this section we present our first construction $\mathsf{TAE1}$, based on *random injections*. We first define *random injections* before presenting our construction $\mathsf{TAE1}$ before presenting our construction which we prove secure against the threshold IND-RCCA notion defined previously.

5.1 Random Injection

The core of our first construction is a *random injection*. There are two equivalent ways to view and define this primitive. The first approach is to add authenticity to the notion of All-or-Nothing



(a) Execution of TAE1.enc, with $n = 3, t = 2$. here, the encryption protocols takes in some message m and set $S = \{1, 2\}$, indicating that party 1 and 2 are to be used for evaluation. The randomized transform gives 5 outputs blocks $e_1, \ldots, e_5$. The first $d = 3$ blocks are to be evaluated by parties 1 and 2. In the above scenario, Assign has assigned block 1 to party 1 and block 2 and 3 to party 2. These evaluations results in encrypted blocks c_1, c_2, c_3. The final ciphertext c' is $(c_1, c_2, c_3, e_4, e_5)$.



(b) $\mathsf{OAEP}(x; r)$ (left) and $\mathsf{RInj}(x; r) = \mathsf{AOAEP}(x; r)$ (right); x is the input, r is uniformly random.

Figure 3: TAE1.enc dataflow diagram (left) and random injection construction $\mathsf{AOAEP}(x; r)$ (right).

Transform[DSS01]. The second approach is to view a random injection as an un-keyed indifferentiable Authenticated Encryption [BF18].

All-or-nothing transform Consider oracles $\mathsf{I} : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ and $\mathsf{I}^{-1} : \mathcal{Y} \to \mathcal{X}$. We view I as a randomized transform (message space $\mathcal{X}$ and randomness space $\mathcal{R}$), with invese I^{-1}. Roughly, [DSS01] required indistinguishability of $\mathsf{I}(x_1), \mathsf{I}(x_2)$ given that ℓ bits of the transforms have been erased. An example of such a transform is OAEP [BR95], which is defined for two random oracles $G : \{0,1\}^k \to \{0,1\}^{nk}$ and $H : \{0,1\}^{nk} \to \{0,1\}^k$ as

$$\mathsf{OAEP}(x; r) = (G(r) \oplus x, H(G(r) \oplus x) \oplus r) \ , \tag{6}$$

where $r \leftarrow\!\!\$\ \{0,1\}^k$. The inverse function is defined in the straightforward way. Myers and Shull [MS18] prove adaptive security of OAEP as defined by [DSS01] (extending prior work by Boyko [Boy99]). The core idea of OAEP is to mask the input x by a random value $G(r)$. In turn, this masked $x \oplus G(r)$ is used to mask r as $r \oplus H(x \oplus G(r))$. Missing any part of the output prevents the function from being inverted efficiently.

Random Injection We add *authenticity* to the all-or-nothing transform. Specifically, we allow I^{-1} to output $\perp$, meaning $\mathsf{I}^{-1} : \mathcal{Y} \to \mathcal{X} \cup \{\perp\}$. Intuitively, we would like that all calls to $\mathsf{I}^{-1}(y)$, where y is *not* an output produced by I, to return $\perp$. This is formalized as follows. We define an *ideal random injection*, $\mathsf{I} : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ with associated inverse $\mathsf{I}^{-1} : \mathcal{X} \to \mathcal{Y} \cup \{\perp\}$, from input domain $\mathcal{X}$ to output domain $\mathcal{Y}$ to be the following.

Proc $\mathsf{I}(x)$	**Proc** $\mathsf{I}^{-1}(y)$
$y \leftarrow\!\!\$\ \mathcal{Y}$; $T[y] \leftarrow x$; Return y	Return $T[y]$

Above, table T is initialized to $\perp$ everywhere. Note that for I to be injective, it should be the case that the number of queries to I, say q, should be much less than $\sqrt{|\mathcal{Y}|}$. Formally, a random injection should be *indifferentialble* [MRH04] from an ideal random injection. Specifically, let $A : \mathcal{X} \to \mathcal{Y}$ be a randomized transform with inverse A^{-1}, both depending on some random oracle H. We say that A is a (q, ϵ)-RInj if there exists simulator S_H (which has access to $\mathsf{I}, \mathsf{I}^{-1}$) such that the RInj-advantage of any distinguisher $\mathcal{D}$, $\mathbf{Adv}^{\text{rinj}}_{A,A^{-1}}(\mathcal{D}) := \Delta_{\mathcal{D}}\left(A, A^{-1}, H\,;\, \mathsf{I}, \mathsf{I}^{-1}, S_H\right)$, is at most ϵ for distinguisher $\mathcal{D}$ making at most q queries to any oracle.

Indifferentiable Authenticated Encryption. The other approach to view our definition of random injection is through indifferentialble authenticated encryption [BF18]. Indifferentiable AE is essentially a key-ed version of random injection. [BF18] present an extension of the OAEP construction which meets this definition. Figure 3b shows the OAEP construction and the (un-keyed) construction of [BF18] which we denote as Authenticated OAEP (AOAEP). The core difference from OAEP is that the randomness used in OAEP is now chosen as $r' \leftarrow I(x\|r)$ where I is a random oracle. When inverting OAEP, $(x\|r)$ is reconstructed and checked to see if the randomness used to construct the injection is consistent with r'. They show that finding such a consistent OAEP output without querying the oracles in the forward direction is infeasible.

Let OAEP be defined in Equation 6, with input space $\{0,1\}^{nk+k}$ and $I : \{0,1\}^{nk+k} \to \{0,1\}^k$ be random oracle. $\mathsf{AOAEP} : \{0,1\}^{nk} \to \{0,1\}^{nk+2k}$ is defined as $\mathsf{AOAEP}(x; r) = \mathsf{OAEP}(x\|r; I(x\|r))$, where r is chosen uniformly at random from $\{0,1\}^k$. The inverse $\mathsf{AOAEP}^{-1} : \{0,1\}^{nk+2k} \to \{0,1\}^{nk}$ is defiend as

$$\mathsf{AOAEP}^{-1}(y) = \begin{cases} x & \text{if } y = \mathsf{OAEP}(x\|r; I(x\|r));\ (x\|r) \leftarrow \mathsf{OAEP}^{-1}(y), \\ \perp & \text{otherwise} \end{cases} \tag{7}$$

Theorem 1. *Let* $\mathsf{AOAEP} : \{0,1\}^{nk+2k} \to \{0,1\}^{nk}$ *be defined above. The proof specifies a simulator* S *such that*

$$\mathbf{Adv}^{\text{rinj}}_{\mathsf{AOAEP}}(\mathcal{D}) \leq \frac{9q^2 + q}{2^k} + \frac{3q^2}{2^{nk+k}}\ . \tag{8}$$

where q *is the maximum number of queries to any oracle that* $\mathcal{D}$ *make.*

This follows from [BF18, Theorem 5]. We also provide a standalone proof below.

Proof. We begin with $\mathcal{A}$ interacting with A, A^{-1}, I, G, H oracle which are implemented by the simulator/challenger. In the end $\mathcal{A}$ interacts directly with T, T^{-1} while the simulator controls I, G, H.

First we consider $\mathcal{A}$ making direct oracle queries to I, G, H. When $\mathcal{A}$ first makes an $I \circ G$ query with $(x\|r)$ (i.e. queries G for the first time on $I(x\|r)$), the simulation queries $y \leftarrow\!\!\$\ T(x)$ and programs the G oracle to output $(y_1\|...\|y_{n+1}) \oplus (x\|r)$ where $y = (y_1\|...\|y_{n+2})$ and $y_i \in \{0,1\}^k$. Conditioned on $I(x\|r)$ not colliding with a previous I query and G at $I(x\|r)$ not previously be queried, this change is identically distributed. Subsequently, if $\mathcal{A}$ queries $H(y_1\|...\|y_{n+1})$ the simulation programs H to return $y_{n+2} \oplus I(x\|r)$. Conditioned on $\mathcal{A}$ not previous querying H at this point, this change is identically distributed.

We now bound the probability of these conditions. Let E_I be the event that $\mathcal{A}$ queried I at distinct (x, r) and (x', r') such that $I(x\|r) = I(x'\|r')$. Then $\mathbf{P}\left[\mathsf{E}_I\right] \leq \frac{q_{\mathsf{I}}^2}{2^k}$ where $\mathcal{A}$ makes q_{I} distinct queries to I. Let $\mathsf{E}_{G\circ I}$ be the event that $\mathcal{A}$ queried G at y prior to querying I at some $(x\|r)$ such that $I(x\|r) = y$. Then $\mathbf{P}\left[\mathsf{E}_{G\circ I}\right] \leq \frac{q_{\mathsf{G}} q_{\mathsf{I}}}{2^k}$ where $\mathcal{A}$ makes q_{G} queries to G. When the simulator programs any of the G queries to $(y_1\|...\|y_{n+1}) \oplus (x\|r)$, let E_H be the event that H has previously

been queried at $(y_1\|...\|y_{n+1})$. Then $\mathbf{P}\left[\mathsf{E}_H\right] \leq \frac{q_{\text{HASH}} q_{\mathsf{G}}}{2^{kn+k}}$. In total we have these local queries agreeing with T except with probability

$$\mathbf{P}\left[\mathsf{E}_I \wedge \mathsf{E}_{G\circ I} \wedge \mathsf{E}_H\right] \leq \frac{q_{\mathsf{I}}^2 + q_{\mathsf{G}} q_{\mathsf{I}}}{2^k} + \frac{q_{\text{HASH}} q_{\mathsf{G}}}{2^{nk+k}}.$$

Now we consider $\mathcal{A}$ making $A(x)$ queries which internally samples $r \leftarrow\!\!\$\ \{0,1\}^k$. Let R be the set of previously sampled r values and E_r be the event that either $r \in R$ or $\mathcal{A}$ has previous queried $I(\cdot\|r)$. Over the q_{A} queries to A the probability of the former is at most $q_{\mathsf{A}}^2/2^k$ while the latter is at most $q_{\mathsf{A}} q_{\mathsf{I}}/2^k$ and therefore $\mathbf{P}\left[\mathsf{E}_r\right] \leq \frac{q_{\mathsf{A}}^2 + q_{\mathsf{A}} q_{\mathsf{I}}}{2^k}$. Given $\neg\mathsf{E}_r$, then $I(x\|r)$ has never previously been queried and outputs a uniform value $z \leftarrow\!\!\$\ \{0,1\}^k$. Let $\mathsf{E}_{A,I\circ G}$ be the event that two of these I queries return the same value or if $\mathcal{A}$ has previously queried G at any of the values. Clearly the former happens with probability at most $q_{\mathsf{A}}^2/2^k$ while the latter occurs with probability at most $q_{\mathsf{A}} q_{\mathsf{G}}/2^k$ and therefore $\mathbf{P}\left[\mathsf{E}_{A,I\circ G}\right] \leq \frac{q_{\mathsf{A}}^2 + q_{\mathsf{A}} q_{\mathsf{G}}}{2^k}$. Assuming $\neg\mathsf{E}_{A,I\circ G}$, A can program G to output $(y_1\|...\|y_{n+1}) \oplus (x\|r)$ where $(y_1\|...\|y_{n+2}) \leftarrow\!\!\$\ T(x)$. Finally, let $\mathsf{E}_{A,H}$ be the event that H has previously been queried at $(y_1\|...\|y_{n+1})$ by A or $\mathcal{A}$ which happens with $\mathbf{P}\left[\mathsf{E}_{A,H}\right] \leq \frac{q_{\mathsf{A}}^2 + q_{\mathsf{A}} q_{\text{HASH}}}{2^{nk+k}}$. Given $\neg\mathsf{E}_{A,H}$, A can program H to output $I(x\|r) \oplus y_{n+2}$. Now observe that the output of A agrees with T except with probability

$$\mathbf{P}\left[\mathsf{E}_r \wedge \mathsf{E}_{A,I\circ G} \wedge \mathsf{E}_{A,H}\right] \leq \frac{q_{\mathsf{A}}^2 + q_{\mathsf{A}} q_{\mathsf{I}} + q_{\mathsf{A}}^2 + q_{\mathsf{A}} q_{\mathsf{G}}}{2^k} + \frac{q_{\mathsf{A}}^2 + q_{\mathsf{A}} q_{\text{HASH}}}{2^{nk+k}}.$$

Therefore, $\mathcal{A}$ can now directly query T. This also means I, G, H are only programmed at points directly queried by $\mathcal{A}$.

Next, let us consider $\mathcal{A}$ making H queries on the output of $y = T(x)$. Let H^* be initialized as $H^* \leftarrow \emptyset$. For each $H(y_1\|...\|y_{n+1})$ query the simulator records the request as $H^* \leftarrow H^* \cup \{(y_1\|...\|y_{n+1})\}$ and then responds as normal. If $\mathcal{A}$ makes a $G(v)$ query, the simulator constructs $Y \leftarrow \{(y_1\|...\|y_{n+1}\|v \oplus H(y_1\|...\|y_{n+1})) \mid (y_1\|...\|y_{n+1}) \in H^*\}$ and checks if there exists a $y \in Y$ such that $T^{-1}(y) \neq \perp$. If no such v exists then G responds normally. Otherwise, let E_y be the event that more than one such y exists which occurs with $\mathbf{P}\left[\mathsf{E}_y\right] \leq {q_{\text{HASH}}}^2/2^k$. Given $\neg\mathsf{E}_y$, there is a unique y and the simulator queries $x \leftarrow T^{-1}(y)$. Prior to the corresponding $y \leftarrow\!\!\$\ T(x)$, let E_v be the event that $\mathcal{A}$ queried $G(v)$ for any of these (v, y) pairs. Given that y, and therefore $v = H(y_1\|...\|y_{n+1}) \oplus y_{n+2}$, is uniformly distributed, the probability of this is $\mathbf{P}\left[\mathsf{E}_v\right] \leq q_{\mathsf{G}} q_{\text{HASH}}/2^k$.

Given $\neg\mathsf{E}_v$, G is programmed to output $G(v) = (y_1\|...\|y_{n+1}) \oplus (x\|r)$ where $r \leftarrow\!\!\$\ \{0,1\}^k$ is uniformly sampled. Let $\mathsf{E}_{r'}$ be the event that G is programmed here with an r such that $I(\cdot\|r)$ has been queried by $\mathcal{A}$. Clearly it holds that $\mathbf{P}\left[\mathsf{E}_{r'}\right] \leq q_{\mathsf{G}} q_{\mathsf{I}}/2^k$. Given $\neg\mathsf{E}_{r'}$, if $\mathcal{A}$ queries $I(x\|r)$ the simulator programs it to return v. The advantage of $\mathcal{A}$ in distinguishing the output of T compared to H, G, I is therefore

$$\mathbf{P}\left[\mathsf{E}_y \wedge \mathsf{E}_v \wedge \mathsf{E}_{r'}\right] \leq \frac{{q_{\text{HASH}}}^2 + q_{\mathsf{G}} q_{\text{HASH}} + q_{\mathsf{G}} q_{\mathsf{I}}}{2^k}$$

Finally, we consider $A^{-1}(y)$ queries where y is not from a direct $y = T(\cdot)$ or indirect $G(I(\cdot))$ oracles. As such, $y \notin Y$ as defined above. Observe that since $G(I(\cdot))$ for y has not been queried there are two possibility. First, there could exist some $(x\|r)$ and that $I(x\|r)$ has been queried but $G(I(x\|r))$ has not. This case, $G(I(x\|r))$ is uniformly distributed and therefore the event E_f that $G(I(x\|r)) = (y_1\|...\|y_{n+1}) \oplus (x\|r)$ has a probability $\mathbf{P}\left[\mathsf{E}_f\right] \leq q_{\mathsf{A}^{-1}}/2^k$. In the second case, we can assume $\mathcal{A}$ evaluated $H(y_1\|...\|y_{n+1})$ and $G(s)$ but not $s \leftarrow I(\cdot)$ an arbitrary $s = H(y_1\|...\|y_{n+1}) \oplus y_{n+2}$. $A^{-1}(y) \neq \perp$ iff $s = I((y_1\|...\|y_{n+1}) \oplus G(s))$. Since I has not been evaluated at this point the probability of this event E_s is $\mathbf{P}\left[\mathsf{E}_s\right] \leq q_{\mathsf{A}^{-1}}/2^k$.

Scheme $\mathsf{TAE1}[\mathsf{RInj}, \mathrm{E}]$

Proc $\mathsf{Setup}(n, t)$
$d \leftarrow \binom{n}{n-t+1}$
For $i \in [d]$ do $\mathsf{sk}[i] \leftarrow \{0,1\}^{\mathrm{E.kl}}$
For $j \in [n]$ and $i \in D_{n,t}(i)$ do $\mathsf{sk}_j[i] \leftarrow \mathsf{sk}[i]$
$\mathsf{pp} \leftarrow (n, t, d)$
Return $((\mathsf{sk}_1, \ldots, \mathsf{sk}_n), \mathsf{pp})$

Proc $\mathsf{Split}_{\mathsf{enc}}(j, m, S)$
$(e_1, \ldots, e_\ell) \leftarrow\!\!\$\ \mathsf{RInj}_d(m)$
$\mathsf{st} \leftarrow (e_{d+1}, \ldots, e_\ell)$
$L \leftarrow\!\!\$\ \mathsf{Assign}(e_1, \ldots, e_d, S)$
Return (L, st)

Proc $\mathsf{Eval}_{\mathsf{enc}}(\mathsf{sk}, j, (i, e_i))$
If $(\mathsf{sk}[i] = \perp)$ then return $\perp$
$c_i \leftarrow \mathrm{E}(\mathsf{sk}[i], e_i)$
Return (i, c_i)

Proc $\mathsf{Combine}_{\mathsf{enc}}(R, \mathsf{st})$
$(e_{d+1}, ..., e_\ell) \leftarrow \mathsf{st}\ ;\ \{(i, c_i)\}_{i \in [d]} \leftarrow R$
Return $(\ell, c_1, ..., c_d, e_{d+1}, ..., e_\ell)$

Proc $\mathsf{Assign}(a_1, \ldots, a_d, S)$
For $i \in [d]$:
  $j \leftarrow\!\!\$\ D_{n,t}(i) \cap S$
  $L \leftarrow L \cup \{(j, (i, a_i))\}$
Return L

Proc $\mathsf{Split}_{\mathsf{dec}}(c', S)$
$(c_1, ..., c_d, e_{d+1}, ..., e_\ell) \leftarrow c'$
$\mathsf{st} \leftarrow (e_{d+1}, ..., e_\ell)$
$L \leftarrow\!\!\$\ \mathsf{Assign}(e_1, \ldots, e_d, S)$
Return (L, st)

Proc $\mathsf{Eval}_{\mathsf{dec}}(\mathsf{sk}, j, (i, c_i))$
If $(\mathsf{sk}[i] = \perp)$ then return $\perp$
$e_i \leftarrow \mathrm{E}^{-1}(\mathsf{sk}[i], c_i)$
Return (i, e_i)

Proc $\mathsf{Combine}_{\mathsf{dec}}(R, \mathsf{st})$
$(e_{d+1}, ..., e_\ell) \leftarrow \mathsf{st}\ ;\ \{(i, e_i)\}_{i \in [d]}$
$m \leftarrow \mathsf{RInj}_d^{-1}(e_1, ..., e_\ell)$
Return m

Figure 4: TAE construction based on random injection RInj and block cipher $\mathrm{E} : \{0,1\}^{\mathrm{E.kl}} \times \{0,1\}^k \to \{0,1\}^k$. Recall that $\mathsf{RInj}_d = \mathsf{RInj} \circ \mathsf{Pad}_{k,d}$.

Baring these events, A^{-1} agree on all outputs as T^{-1} and therefore $\mathcal{A}$ can now query T^{-1} directly. In total the advantage of $\mathcal{A}$ is bounded by the union of these events. Therefore,

$$\mathbf{Adv}^{\mathrm{rinj}}_{\mathsf{AOAEP}}(\mathcal{D}) \leq \frac{9q^2 + q}{2^k} + \frac{3q^2}{2^{nk+k}}$$

where q is the maximum of $q_{\mathsf{I}}, q_{\mathsf{G}}, q_{\mathrm{HASH}}, q_{\mathsf{A}}, q_{\mathsf{A}^{-1}}$. □

Extension to variable-input-length Let d be some integer. Consider the following padding function $\mathsf{Pad}_{k,d} : \{0,1\}^* \to (\{0,1\}^k)^*$. Upon input x, $\mathsf{Pad}_{k,d}$ first append a 1 at the end of x, then it appentd 0's until the length ℓ is some $m \cdot k$ for some integer $m \geq d$. Consider the variable-input-length transformation $\mathsf{RInj}_d := \mathsf{RInj} \circ \mathsf{Pad}_{k,d}$. It is not hard to show $\mathsf{RInj} \circ \mathsf{Pad}_{k,d}$ is indifferentiable to variable-input-length random injections with the Equation 8 bound for n set to d.

5.2 The Construction

Our TAE construction in Figure 4 builds on an random injection RInj as defined above. We define the sets $D_{n,t}(i)$ for integers $i \in [d]$ where $d = \binom{n}{n-t+1}$, with the following property: $D_{n,t}(1), \ldots, D_{n,t}(d)$ shall be all the subsets of $[n]$ with size exactly $n-t+1$. Each party $i \in [n]$ will hold secret key sk_j if and only if $i \in D_{n,t}(j)$. Together these secret keys form type of a t-out-of-n replicated secret sharing of the master key $((\mathsf{sk}_1, \ldots, \mathsf{sk}_d))$. To encrypt, the initiating party computes $e \leftarrow\!\!\$\ \mathsf{RInj}_d(m)$. e is then partitioned into *at least* d block $e_1, ..., e_d, ..., e_\ell$ with each $e_i \in \{0,1\}^k$. Each e_i is sent to a single party which holds the key sk_i. This party returns $c_i \leftarrow \mathsf{PRP}(\mathsf{sk}_i, e_i)$. The final ciphertext is the comprised of $c_1, ..., c_d$ plus any additional blocks of e. An illustration of this is given in Figure 3a. Decryption is defined in the straightforward way where the RInj ensure that the plaintext has not been modified.

5.3 Security

We show that given a secure block cipher E and a secure random injection RInj, scheme $\mathsf{TAE} = \mathsf{TAE1}[\mathsf{RInj}, \mathrm{E}]$ is IND-RCCA secure.

Theorem 2. *Let* RInj *be a random injection. Let* $\mathrm{E} : \mathrm{E.kl} \times \{0,1\}^k \to \{0,1\}^k$ *be a block cipher. Let* $\mathsf{TAE} = \mathsf{TAE1}[\mathsf{RInj}, \mathrm{E}]$. *Let* $\mathcal{A}$ *be an adversary against* TAE. *The proof gives adversaries* $\mathcal{B}, \mathcal{C}$, *whose running times are about that of* $\mathcal{A}$ *plus some simulation overhead, such that*

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathsf{TAE},n,t}(\mathcal{A}) \le 2 \cdot \mathbf{Adv}^{\text{d-prp}}_{\mathrm{E}}(\mathcal{B}) + 2 \cdot \mathbf{Adv}^{\text{rinj}}_{\mathsf{RInj}}(\mathcal{C}) \quad + \frac{(d+2) \cdot q^2}{2^{k-1}} \,, \tag{9}$$

where $q \ge 2$ *is the maximum number of queries to any oracle available to* $\mathcal{A}$, *and* $d = \binom{n}{n-t+1}$ *is assumed to be larger than* 2.

of Theorem 2. Let $\mathbf{G}_0 = \mathbf{G}^{\text{ind-cca}}_{\mathsf{TAE},n,t}(\mathcal{A})$. Let us also fix the adversary $\mathcal{A}$. For the rest of the proof, we let $\mathcal{CR} \subset [n]$ denote the set of corrupt parties that the adversary $\mathcal{A}$ choses to corrupt (formally, $\mathcal{CR}$ is random variable).

Consider $\mathbf{G}_1$, modified from $\mathbf{G}_0$ such that $\mathrm{E}(\mathsf{sk}_i, \cdot)$ is replaced with random permutations, P_i, for $i \in [n] \setminus C$. More precisely, these permutations shall be lazily sampled as follows, so that we can refer to the partial permuatation tables T_i and T_i^{-1}

Proc $P_i(x)$	**Proc** $P_i^{-1}(y)$
If $T_i[x] = \perp$ then	If $T_i^{-1}[y] = \perp$ then
$\quad y \leftarrow\!\!\$\ \{0,1\}^k - \{T_i^{-1}\}$	$\quad x \leftarrow\!\!\$\ \{0,1\}^k - \{T_i\}$
$\quad T_i[x] \leftarrow y$	$\quad T_i^{-1}[y] \leftarrow x$
$\quad T_i^{-1}[y] \leftarrow x$	$\quad T_i[x] \leftarrow y$
Return $T_i[x]$	Return $T_i^{-1}[y]$

It is standard to construct a PRP-adversary $\mathcal{B}$ whose advantage bounds the distance between $\mathbf{G}_0$ and $\mathbf{G}_1$. Specifically,

$$\mathbf{P}\big[\mathbf{G}_0\big] - \mathbf{P}\big[\mathbf{G}_1\big] \le \mathbf{Adv}^{\text{d-prp}}_{\mathrm{E}}(\mathcal{B}) \,. \tag{10}$$

Next, we modify $\mathbf{G}_1$ into $\mathbf{G}_2$ so that RInj is replaced with ideal random injection I and I^{-1}. It is standard to give adversary $\mathcal{C}$ such that

$$\mathbf{P}\big[\mathbf{G}_1\big] - \mathbf{P}\big[\mathbf{G}_2\big] \le \mathbf{Adv}^{\text{rinj}}_{\mathsf{RInj}}(\mathcal{C}) \,. \tag{11}$$

Consider the event E in $\mathbf{G}_2$ indicating when that *authenticity* condition is violated.

$$E : |\mathsf{DecSet}| > \left\lfloor \frac{\mathsf{ct}_{\mathsf{Eval}}}{n - |\mathcal{CR}|} \right\rfloor \,.$$

We claim that

$$\mathbf{P}\big[E\big] \le \frac{d \cdot q^2}{2^k} \,. \tag{12}$$

First, consider the event bad that there are some collisions in the first d blocks of the output of I. More precisely, bad is TRUE if and only if there is some x, x' that was input to I such that $\mathsf{I}(x)[i] = \mathsf{I}(x')[i]$ for some $i \in [d]$. Since, I is a random injection,

$$\mathbf{P}\big[\mathsf{bad}\big] \le \frac{d \cdot q(q-1)}{2^{k+1}} \tag{13}$$

via the birthday bound and a union bound. Now, we shall bound $\mathbf{P}\big[E \wedge \neg \mathsf{bad}\big]$. If bad does not happen, then each $\mathsf{Eval}_{\mathsf{enc}}$ and $\mathsf{Eval}_{\mathsf{dec}}$ query corresponds to *at most one* ciphertext. Note that $|\mathsf{DecSet}|$ is the number of valid y that adversary $\mathcal{A}$ can generate such that $\mathsf{I}^{-1}(y) \neq \perp$. Suppose E happens and $\neg\mathsf{bad}$, then it must be that during one of the queries to $\mathsf{Combine}_{\mathsf{dec}}$, one of P_i^{-1} samples a fresh point for some $i \in [n] - \mathcal{CR}$. The probability that this freshly-sampled point makes a valid y that goes through I^{-1} to make a valid decryption is at most

$$\frac{1}{2^k - |T_i|} \leq \frac{1}{2^{k-1}} \,, \tag{14}$$

since I^{-1} always output $\perp$ unless it is given some y that was generated by I. Putting (13) and (14) together, we obtain (12).

Consider $\mathbf{G}_3$, modified from $\mathbf{G}_2$ such that if $\mathsf{forgery}$ is set, then game $\mathbf{G}_3$ returns TRUE or FALSE with probability a half each (instead of always returning TRUE). By the fundamental lemma of game playing,

$$\mathbf{P}\big[\mathbf{G}_2\big] - \mathbf{P}\big[\mathbf{G}_3\big] \leq \mathbf{P}\big[E\big] \leq \frac{d \cdot q^2}{2^k} \,. \tag{15}$$

Lastly, we try to bound

$$2 \cdot \mathbf{P}\big[\mathbf{G}_3\big] - 1 \,. \tag{16}$$

Let us analyze the $\text{Split}_{\mathsf{enc}}$ oralce in $\mathbf{G}_3$.

Proc $\textsc{Split}_{\mathsf{enc}}(\mathsf{id}, m_0, m_1, S)$
Require $\mathsf{id} \notin \mathcal{CR}$, $|m_0| = |m_1|$
$u \leftarrow u + 1$; $\mathsf{id}_u \leftarrow \mathsf{id}$
$(e_1, \ldots, e_q) \leftarrow\!\!\$\ \mathsf{I}(m_b, d)$
$L_u \leftarrow\!\!\$\ \mathsf{Assign}(e_1, \ldots, e_d)$
$\mathsf{st}_u \leftarrow (e_{d+1}, \ldots, e_q)$
Return $\{(k, x) \in L_u \mid k \in \mathcal{CR}\}$

Note that since the *privacy* condition holds, one of the blocks e_i is not given to the adversary. Let us consider a game $\mathbf{G}_4$ in which the inverse of $(e_1, \ldots, e_q)$ is not set (to be m_b). Instead, we shall set bad if there is a query $(e_1, \ldots, e_q)$ to I^{-1} that would have returned some m_b in $\mathbf{G}_3$. Note that $\mathbf{G}_4$ does not leak any information about bit b and hence $2 \cdot \mathbf{P}\big[\mathbf{G}_4\big] - 1 = 0$. It is left to bound the distance between $\mathbf{G}_3$ and $\mathbf{G}_4$. It remains to bound the probability of bad being set in $\mathbf{G}_4$. Since one of these block is unknown to the adverasry. This probability is upperbounded by at most $\frac{1}{2^k - \max_i |T_i|} \leq \frac{1}{2^{k-1}}$ for each I^{-1} query and challenge ciphertext pair. Hence,

$$\mathbf{P}\big[\mathbf{G}_3\big] - \mathbf{P}\big[\mathbf{G}_4\big] \leq \frac{q^2}{2^{k-1}} \,.$$

Game $\mathbf{G}^{\mathrm{dprf}}_{\mathsf{DP},n,t}(\mathcal{A})$

$b \leftarrow\!\!\$\ \{0,1\}$
$(\llbracket \mathsf{sk} \rrbracket, \mathsf{pp}) \leftarrow\!\!\$\ \mathsf{DP.Setup}(n,t)$
$(C, \mathsf{st}_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}_0(\mathsf{pp})$
$b' \leftarrow\!\!\$\ \mathcal{A}_1^{\langle \mathbf{Proc} \rangle}(\mathsf{st}_{\mathcal{A}}, (\mathsf{sk}_i)_{i \in C})$
If $(\exists x \in \mathsf{chlSet} : |\mathsf{ct}[x] \cup C| \geq t)$:
  Return $b'' \leftarrow\!\!\$\ \{0,1\}$
Return $(b = b')$

Proc $\textsc{Eval}(i, x)$

Require $i \notin C$
$\mathsf{ct}[x] \leftarrow \mathsf{ct}[x] \cup \{i\}$
Return $\mathsf{DP.Eval}(\mathsf{sk}_i, x)$

Proc $\textsc{Chl}(x, S, \mathsf{rsp})$

Require $x \notin \mathsf{chlSet}$
$\mathsf{chlSet} \leftarrow \mathsf{chlSet} \cup \{x\}$
For $i \in S \setminus C$:
  $\mathsf{rsp}[i] \leftarrow \mathsf{DP.Eval}(\mathsf{sk}_i, x)$
$v_0 \leftarrow \mathsf{DP.Combine}(\mathsf{rsp})$
If $(v_0 = \perp)$ then Return $\perp$
$v_1 \leftarrow\!\!\$\ \{0,1\}^{|v_0|}$; Return v_b

Figure 5: The DPRF security game.

Putting things together, we have

$$
\begin{aligned}
&\mathbf{Adv}^{\text{ind-cca}}_{\mathsf{TAE},n,t}(\mathcal{A}) \\
&= 2 \cdot \mathbf{P}\big[\mathbf{G}_0\big] - 1 \\
&= 2 \cdot (\mathbf{P}\big[\mathbf{G}_1\big] + \mathbf{Adv}^{\text{d-prp}}_{\mathrm{E}}(\mathcal{B})) - 1 \\
&= 2 \cdot (\mathbf{P}\big[\mathbf{G}_2\big] + \mathbf{Adv}^{\text{d-prp}}_{\mathrm{E}}(\mathcal{B}) + \mathbf{Adv}^{\text{rinj}}_{\mathsf{RInj}}(\mathcal{C})) - 1 \\
&= 2 \cdot (\mathbf{P}\big[\mathbf{G}_3\big] + \mathbf{Adv}^{\text{d-prp}}_{\mathrm{E}}(\mathcal{B}) + \mathbf{Adv}^{\text{rinj}}_{\mathsf{RInj}}(\mathcal{C}) + \frac{d \cdot q^2}{2^k}) - 1 \\
&\leq 2 \cdot (\mathbf{P}\big[\mathbf{G}_4\big] + \mathbf{Adv}^{\text{d-prp}}_{\mathrm{E}}(\mathcal{B}) + \mathbf{Adv}^{\text{rinj}}_{\mathsf{RInj}}(\mathcal{C}) + \frac{(d+2) \cdot q^2}{2^k}) - 1 \\
&= 2 \cdot \mathbf{Adv}^{\text{d-prp}}_{\mathrm{E}}(\mathcal{B}) + 2 \cdot \mathbf{Adv}^{\text{rinj}}_{\mathsf{RInj}}(\mathcal{C}) + \frac{(d+2) \cdot q^2}{2^{k-1}} \,.
\end{aligned}
$$

□

6 Constructions from Threshold PRF & Signature

In this section, we provide two TAE constructions. One of them achieves IND-RCCA security, whereas the other one achieves IND-CCA security. The first construction is based on a DPRF and a threshold signature scheme. The second construction is achieved simply replacing the DPRF with a DVRF.

Here, we first formally define DPRF and DVRF. We then provide our instantiations of DPRF and DVRF with accompanying security analysis that is *modular* and *concrete*. Our instantiations include variants of the DDH-based DPRFs introduced by Naor et al. [NPR99] and used by Agrawal et al. [AMMR18a] to construct TAE schemes. Finally, we provide our TAE constructions that use them along with threshold signatures.

6.1 Definition and Constructions of DPRF and DVRF

6.1.1 DPRF.

A DPRF DP consists of algorithms Setup, Eval, Combine, as well as input and output spaces In, Out $\subseteq \{0,1\}^*$. The algorithms have the following syntax.

- $\mathsf{Setup}(n,t) \xrightarrow{\$} (\llbracket \mathsf{sk} \rrbracket_n, \mathsf{pp})$, where n, t are positive integers such that $t \leq n$. Setup generates n secret keys and public parameter pp.

- $\mathsf{Eval}(\mathsf{sk}_i, x) \to z_i$. Eval generates its response for the input $x \in \mathsf{In}$ using secret key sk_i. We require that $z_i = \perp$ if and only if $x \notin \mathsf{In}$.
- $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}) \to y$. Combines the partial shares z_i and returns an output $y \in \mathsf{Out} \cup \{\perp\}$.

The secret-key shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ are distributed amongst n parties. If an initiating party $\mathcal{P}_j$ intends to compute the DPRF on an input x, it sends x to a subset S of all parties. Each party $i \in S$ returns a partial evaluation $z_i \leftarrow \mathsf{Eval}(\mathsf{sk}_i, x)$. The initiator, on receiving the partial evaluations, runs combine to obtain the output, i.e., $y \leftarrow \mathsf{Combine}(\{i, z_i\}_{i \in S})$.

Correctness. For a DPRF DP to be correct, every input $x \in \mathsf{DP.In}$ must map to a unique output $y \in \mathsf{DP.Out}$.[2] Specifically, for any $0 < t \leq n$, any $(\llbracket \mathsf{sk} \rrbracket_n, \mathsf{pp}) \leftarrow\!\!\$\ \mathsf{DP.Setup}(n, t)$, $x \in \mathsf{DP.In}$, $j, j' \in [n]$, and $S, S' \subseteq [n]$ such that $|S|, |S'| \geq t$,

$$\mathsf{Combine}(\{i, z_i\}_{i \in S}) = \mathsf{Combine}(\{j, z'_j\}_{j \in S'}),$$

where $z_i \leftarrow \mathsf{Eval}(\mathsf{sk}_i, x)$ for $i \in S$ and $z'_j \leftarrow \mathsf{Eval}(\mathsf{sk}_j, x)$ for $j \in S'$. Note that this condition was named "consistency" in previous literature.

DPRF security. Consider the security game given in Figure 5. The goal of a DPRF adversary is to predict the value of the global variable bit b, while interacting with the following two oracles.

- $\textsc{Eval}(i, x)$. Adversary $\mathcal{A}$ calls this oracle to have party i run $\mathsf{Eval}(\mathsf{pp}, \mathsf{sk}_i, x)$ and obtain the resulting output. In doing so, the game adds party i to the set $\mathsf{ct}[x]$ so as to keep count of the number of Eval queries made under x. If $|\mathsf{ct}[x] \cup C|$ is at least the threshold value t, then $\mathsf{Eval}(i, x)$ will return $\perp$.
- $\textsc{Chl}(x, S, \mathsf{rsp})$. The adversary calls this oracle to receive challenge outputs, which it can use to help guess the challenge bit b. Each input x to Chl can only be queried once, hence Chl keeps a set chlSet storing the set of challenges seen so far by the game. Upon input (x, S, rsp), Chl sets $z_i \leftarrow \mathsf{rsp}[i]$ if $i \in C$ and $\mathsf{rsp}[i]$ is defined, and otherwise computes z_i honestly using sk_i. The oracle then outputs v_b, where
  - v_0 is the combination of the values z_i computed via Combine, and
  - v_1 is $\perp$ if v_0 is $\perp$, or an uniformly sampled string of length $|v_0|$ otherwise.

The adversary wins if bad is not set to True and the predicted bit b' is equal to b sampled by the game. The DPRF advantage function is $\mathbf{Adv}^{\mathrm{dprf}}_{\mathsf{DP},n,t}(\mathcal{A}) = 2 \cdot \mathbf{P}\left[\mathbf{G}^{\mathrm{dprf}}_{\mathsf{DP},n,t}(\mathcal{A})\right] - 1$.

Comparison with DiSE. Our DPRF security notion strengthens that of Agrawal et al. [AMMR18a] by allowing *multiple*, different challenge queries. This brings the DPRF security closer in-line with traditional definitions of pseudo-random functions. However, as shown below, this incurs an additional security loss proportional to the number of challenges in the reduction.

6.1.2 DVRF.

Our DVRF definition simply extends the DPRF definition above in a way that suffices for our purposes. DVRFs were recently formalized by Galindo et al. [GLOW20] who, in turn, build on DiSE [AMMR18a]. Their definition is similar to ours. However, they capture many other properties,

[2] The output is *unique* in the sense that no matter which parties participate in the protocol, the output must be the same — as long as the parties behave honestly. The only thing that matters is that the size of S must be at least t.

Game $\mathbf{G}^{\mathrm{dvrf}}_{\mathsf{DV},n,t}(\mathcal{A})$

$b \leftarrow\!\!\$\ \{0,1\}$; $(\llbracket \mathsf{sk} \rrbracket, \mathsf{pp}) \leftarrow\!\!\$\ \mathsf{DV.Setup}(n,t)$
$(C, \mathsf{st}_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}_0(\mathsf{pp})$
$b' \leftarrow\!\!\$\ \mathcal{A}_1^{\langle \mathbf{Proc} \rangle}(\mathsf{st}_{\mathcal{A}}, (\mathsf{sk}_i)_{i \in C})$
If $(\exists x \in \mathsf{chlSet} : |\mathsf{ct}[x] \cup C| \geq t)$ then
  Return $b'' \leftarrow\!\!\$\ \{0,1\}$
Return $(b = b')$

Proc $\textsc{Eval}(i, x)$

Require $i \notin C$
$\mathsf{ct}[x] \leftarrow \mathsf{ct}[x] \cup \{i\}$
Return $\mathsf{DV.Eval}(\mathsf{sk}_i, x)$

Proc $\textsc{Chl}(x, S, \mathsf{rsp})$

Require $x \notin \mathsf{chlSet}$
$\mathsf{chlSet} \leftarrow \mathsf{chlSet} \cup \{x\}$
For $i \in S \setminus C$ do $\mathsf{rsp}[i] \leftarrow \mathsf{DV.Eval}(\mathsf{sk}_i, x)$
If $\mathsf{DV.Verify}(x, \mathsf{rsp})$ do:
  $v_0 \leftarrow \mathsf{DV.Combine}(\mathsf{rsp})$
  If $(v_0 = \perp)$ then Return $\perp$
  $v_1 \leftarrow\!\!\$\ \{0,1\}^{|v_0|}$; Return v_b
Else Return $\perp$

(a) The DVRF security game.

Game $\mathbf{G}^{\mathrm{dvrf-rob}}_{\mathsf{DV},n,t}(\mathcal{A})$

$(\llbracket \mathsf{sk} \rrbracket, \mathsf{pp}) \leftarrow\!\!\$\ \mathsf{DV.Setup}(n,t)$
$(C, \mathsf{st}_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}_0(\mathsf{pp})$
$(S, x^*, \{i, z_i^*\}_{i \in S}) \leftarrow\!\!\$\ \mathcal{A}_1^{\langle \mathbf{Proc} \rangle}(\mathsf{st}_{\mathcal{A}}, (\mathsf{sk}_i)_{i \in C})$
If $|S| < t$ then return 0
Else do:
  For $i \in S \cap C$ do $\mathsf{rsp}[i] := z_i^*$
  For $i \in S \setminus C$ do $\mathsf{rsp}[i] := \mathsf{DV.Eval}(\mathsf{sk}_i, x^*)$
  $y^* := \mathsf{DV.Combine}(\mathsf{rsp})$
  $y := \mathsf{DV.Combine}(\{(i, \mathsf{Eval}(\mathsf{sk}_i, x^*))\}_{i \in S})$
  $b := \mathsf{Verify}(\mathsf{pp}, x^*, \mathsf{rsp})$
  Return $(y \neq y^*) \wedge (b = 1)$

Proc $\textsc{Eval}(i, x)$

Require $i \notin C$
Return $\mathsf{DV.Eval}(\mathsf{sk}_i, x)$

(b) The DVRF robustness game.

Figure 6: The DVRF games.

specifically useful in their context, that we do not take into account here. Nevertheless, we believe that their constructions satisfy our definition too.

Recall that a DPRF DP consists of algorithms $\mathsf{Setup}, \mathsf{Eval}, \mathsf{Combine}$, as well as input and output space $\mathsf{In}, \mathsf{Out} \subseteq \{0,1\}^*$. Consider an additional verification algorithm Verify for DP with the following syntax:

- $\mathsf{Verify}(\mathsf{pp}, x, \{i, z_i\}_{i \in S}) =: 1/0$. This is a deterministic procedure that takes an input value $x \in \mathsf{In}$, the public parameter pp[3] and a bunch of partial evaluations from parties in set S, and outputs a decision bit, where 1 implies that the verification is successful.

We require Verify to satisfy the following *completeness* property:

- *Completeness.* For any $x \in \mathsf{In}$, any pp generated by $\mathsf{Setup}(n,t)$, and for any set $S \subseteq [n]$ s.t. $|S| \geq t$, $\mathsf{Verify}(\mathsf{pp}, x, \{(i, \mathsf{Eval}(\mathsf{sk}_i, x))\}_{i \in S})$ always outputs 1.

DVRF security. A DVRF is a DPRF too. Therefore it should have a pseudorandomness property similar to DPRF. In particular we adapt the DPRF security game (from Figure 5) to DVRF in Figure 6a. We define an adversary $\mathcal{A}$'s advantage similarly: $\mathbf{Adv}^{\mathrm{dvrf}}_{(}(\mathcal{A})) = 2 \cdot \mathbf{P}\left[\mathbf{G}^{\mathrm{dvrf}}_{\mathsf{DV},n,t}(\mathcal{A})\right] - 1$.
DVRF robustness. We require the procedure Verify to satisfy another security property. In particular, we consider a *robustness* game as described in Figure 6b. In this game, the goal of the adversary is to produce a set of partial evaluations that result in a wrong computation, but make the verification succeed. The adversary is allowed to query the evaluation oracle multiple times to learn correct partial evaluation values for honest parties. The robustness advantage function is defined as: $\mathbf{Adv}^{\mathrm{dvrf-rob}}_{\mathsf{DP},n,t}(\mathcal{A}) = \mathbf{P}\left[\mathbf{G}^{\mathrm{dvrf-rob}}_{\mathsf{DV},n,t}(\mathcal{A})\right]$.
Comparing with strongly-secure DiSE DPRF. Our DVRF definition is reminiscent of the DiSE DPRF definition with (i) a (malicious) correctness property (Def. 5.4 of [AMMR18b]) and (ii) a public verifiability (formalization of the Remark 8.3 of [AMMR18b])), apart from the multi-challenge extension already incorporated into our DPRF. However, we remark that our formalization is slightly different as we explicitly mention existence of a verification algorithm that can be used in the protocol to immediately detect a malicious response, whereas their correctness definition guarantees that the adversary cannot force the "Combine" procedure to accept a valid (not $\perp$) value which is different from the correct value. Looking ahead, this modular extension helps us simplifying our proof for the IND-CCA secure TAE construction.

6.1.3 XOR DPRF [NPR99, AMMR18a]

Let $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a function family. We assume that there are three integers associated with F such that $\mathcal{K}, \mathcal{X}, \mathcal{Y}$ are equal to $\{0,1\}^{\mathsf{F.kl}}, \{0,1\}^{\mathsf{F.il}}, \{0,1\}^{\mathsf{F.ol}}$, respectively. Let μ be a positive integer. Let $k_i \in \{0,1\}^{\mathsf{F.kl}}$ and $\rho_i : \mathcal{X} \to \mathcal{Y}$ be chosen uniformly and independently at random for $i = 1, \ldots, \mu$, then the *μ-PRF advantage* of $\mathcal{A}$ against F is defined as

$$\mathbf{Adv}^{\mu\text{-prf}}_{\mathsf{F}}(\mathcal{A}) = \mathbf{P}\left[\mathcal{A}^{\mathsf{F}_{k_1}, \ldots, \mathsf{F}_{k_\mu}}\right] - \mathbf{P}\left[\mathcal{A}^{\rho_1, \ldots, \rho_\mu}\right]. \tag{17}$$

In general, we have

$$\mathbf{Adv}^{\mu\text{-prf}}_{f}(\mathcal{A}) \leq \mu \cdot \mathbf{Adv}^{1\text{-prf}}_{f}(\mathcal{A}), \tag{18}$$

[3] Usually verification keys are separated from public parameters. Nevertheless, for simplicity, we include our *verification keys* within pp. In particular, this also means that our verification procedure is public, which does not seem to be necessary for our purpose. In fact, we believe that a privately verifiable variant, similar to the one proposed in DiSE, also works. As noted in DiSE, private verifiability may lead to a slightly faster protocol. However, we stick to the public verifiability notion because it is a useful feature to have (as also pointed out in Remark 8.3 of [AMMR18b]) and is also simpler to describe.

Scheme $\mathsf{DP}_{\mathsf{F}}^{\mathrm{PRF}}$

Proc $\mathsf{Setup}(n,t)$
$d \leftarrow \binom{n}{n-t+1}$
For $i \in [d]$ do
  $\mathsf{sk}[i] \leftarrow \{0,1\}^{\mathsf{F.kl}}$
For $j \in [n]$ do
  For $i \in D_{n,t}(i)$ do
    $\mathsf{sk}_j[i] \leftarrow \mathsf{sk}[i]$
$\mathsf{pp} \leftarrow (n,t,d)$
Return $((\mathsf{sk}_1,\ldots,\mathsf{sk}_n),\mathsf{pp})$

Proc $\mathsf{Eval}(\mathsf{pp},\mathsf{sk},x)$
$V \leftarrow \{\mathsf{F}(\mathsf{sk}[i],x)\}_{i\in[d],\mathsf{sk}[i]\neq\perp}$
Return V

Proc $\mathsf{Combine}(\mathsf{pp},L)$
$\{(j,V_j)\} \leftarrow L$
$W \leftarrow \bigcup V_j$
Return $\bigoplus_{w\in W} w$

Figure 7: XOR DPRF construction.

and there are constructions where the above inequality is tight. However, there are constructions for which $\mathbf{Adv}_f^{\mu\text{-prf}}(\mathcal{A})$ equals $\mathbf{Adv}_f^{1\text{-prf}}(\mathcal{A})$.

Let $d := \binom{n}{n-t+1}$ and $D_{n,t}(1),\ldots,D_{n,t}(d)$ be defined as as in 5.2. The XOR DPRF protocol distributes PRF keys to the participants in such a way that no $t-1$ subset of the parties are able to reconstruct all the PRF keys, but any subset of t parties can. To evaluate the XOR DPRF, one requests t parties to evaluate a PRF f on a given input x under all the keys they have. After receiving all the PRF outputs, one XORs them together to get the final DPRF output. The protocol is specified in Figure 7.

Note that a naïve implementation of the protocol has all participants send back all PRF outputs, which results in redundant computation and unnecessary communication overhead. One can optimize the protocol further by letting the participants know which PRF keys to use. For example, a bit vector of length n can be sent which indicates which t parties will participate. Which keys to use can then be locally determined from this selection. Critically, the participating parties can then locally XOR together their set of partial DPRF evaluations. The overall communication is then $(t-1)(\mathsf{F.il}+\mathsf{F.ol}+n)$ bits.

Theorem 3. *Let $\mathcal{A}$ be a DPRF adversary. The proof gives d-PRF adversary $\mathcal{B}$, with running time similar to $\mathcal{A}$ plus some simulation overhead, such that*

$$\mathbf{Adv}^{\mathrm{dprf}}_{\mathsf{DP}_{\mathsf{F}}^{\mathrm{PRF}},n,t}(\mathcal{A}) \le 2\cdot\mathbf{Adv}^{\mathrm{d\text{-}prf}}_{\mathsf{F}}(\mathcal{B})\ .$$

Proof. Consider games $\mathbf{G}_0$ and $\mathbf{G}_1$ given in Figure 8. Note that $\mathbf{G}_0$ is simply a rewrite of $\mathbf{G}^{\mathrm{dprf}}_{\mathsf{DP}_{\mathsf{F}}^{\mathrm{PRF}},n,t}\mathcal{A}$. Game $\mathbf{G}_1$ differs from $\mathbf{G}_0$ in that all evaluations of F by honest parties are replaced with uniform random values. We claim that there exists adversary $\mathcal{B}$ such that

$$\mathbf{P}\big[\mathbf{G}_0\big] - \mathbf{P}\big[\mathbf{G}_1\big] \le \mathbf{Adv}^{\mathrm{d\text{-}prf}}_{\mathsf{F}}(\mathcal{B})\ . \tag{19}$$

Consider Multi-PRF adversary $\mathcal{B}$ against F is constructed as follows. Adversary $\mathcal{B}$ simply runs $\mathbf{G}_0$ exactly besides that it answers queries to $\textsc{Fn}$ using its own oracles. It is not hard to see that if $\mathcal{B}$ is given oracles $\mathsf{F}_{\mathsf{sk}_i}(\cdot)$, then it simulates $\mathbf{G}_0$; if $\mathcal{B}$ is given oracles ρ_i, then it simulates $\mathbf{G}_1$. This justifies (19).

Next, we claim that

$$\mathbf{P}\big[\mathbf{G}_1\big] = \frac{1}{2}\ . \tag{20}$$

Game $\mathbf{G}_0$ / $\mathbf{G}_1$

$b \leftarrow\!\!\$\ \{0,1\}$; $(\llbracket \mathsf{sk} \rrbracket, \mathsf{pp}) \leftarrow\!\!\$\ \mathsf{DP.Setup}(n,t)$
$(C, \mathsf{st}_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}_0(\mathsf{pp})$
$b' \leftarrow\!\!\$\ \mathcal{A}_1^{\langle \mathbf{Proc} \rangle}(\mathsf{st}_{\mathcal{A}}, (\mathsf{sk}_i)_{i \in C})$
If $(\exists x \in \mathsf{chlSet} : |\mathsf{ct}[x] \cup C| \geq t)$ then
  Return $b'' \leftarrow\!\!\$\ \{0,1\}$
Return $(b = b')$

Proc $\textsc{Eval}(i, x)$
If $i \in C$ then return $\perp$
$\mathsf{ct}[x] \leftarrow \mathsf{ct}[x] \cup \{i\}$
Return $\textsc{Fn}(i, x)$

Proc $\textsc{Chl}(x, S, \mathsf{rsp})$
If $x \in \mathsf{chlSet}$ then return $\perp$
$\mathsf{chlSet} \leftarrow \mathsf{chlSet} \cup \{x\}$
For $i \in S \setminus C$ do $\mathsf{rsp}[i] \leftarrow \textsc{Fn}(i, x)$
$v_0 \leftarrow \mathsf{DP.Combine}(\mathsf{rsp})$
If $(v_0 = \perp)$ then return $\perp$
$v_1 \leftarrow\!\!\$\ \{0,1\}^{|v_0|}$; Return v_b

Proc $\textsc{Fn}(i, x)$
If $T_i[x] = \perp$ then $T_i[x] \leftarrow \{0,1\}^k$
$\underline{\mathbf{G}_0}$: Return $\mathsf{F}(\mathsf{sk}_i, x)$
$\underline{\mathbf{G}_1}$: Return $T_i[x]$

Figure 8: Games $\mathbf{G}_0$ and $\mathbf{G}_1$ used in the proof of Theorem 3.

Consider the event

$$E : \exists x \in \mathsf{chlSet} : |\mathsf{ct}[x] \cup C| \geq t \ .$$

Note that, by construction, $\mathbf{P}\big[\mathbf{G}_1 \mid E\big] = \frac{1}{2}$. Next, suppose $\neg E$, we will show that $\mathbf{P}\left[\mathbf{G}_1 \mid \neg E\right] = \frac{1}{2}$. Suppose $\textsc{Chl}(x\,;\, S, \mathsf{rsp})$ is a query made by $\mathcal{A}$ in $\mathbf{G}_1$. We know that $|S| \geq t$ and $|\mathsf{ct}[x] \cup C| < t$, hence $\textsc{Chl}$ includes an $\textsc{Fn}(i, x)$ query where $i \notin C$ which $\mathcal{A}$ does not query directly. This output is independent and uniformly random of all other values in the interaction. As a result, when $\textsc{Fn}(j^*, x)$ is XORed with the other $\textsc{Eval}$ queries and rsp values, we know that $\textsc{Chl}(x\,;\, S, \mathsf{rsp})$ is independent and uniformly random as well. Hence, there is no leakage about the bit b in game $\mathbf{G}_1$. This justifies equation (20).

Scheme $\mathsf{DP}^{\mathrm{DDH}}_{\mathbb{G}}$

Proc $\mathsf{Setup}(n, t)$

$s \leftarrow\!\!\$\ \mathbb{Z}_p$
For $j \in [t-1]$ do $c_j \leftarrow\!\!\$\ \mathbb{Z}_p$
For $i \in [n]$
  $\mathsf{sk}_i \leftarrow s + \sum_{j\in[t-1]} c_j \cdot i^j$
Return $((\mathsf{sk}_1, \ldots, \mathsf{sk}_n), \epsilon)$

Proc $\mathsf{Eval}(\mathsf{pp}, \mathsf{sk}, x)$

Return $\mathrm{H}(x)^{\mathsf{sk}}$

Proc $\mathsf{Combine}(\mathsf{pp}, L)$

If $|L| < t$ then return $\perp$
$\{(i, z_i)\} \leftarrow L$
$y \leftarrow \prod_i (z_i)^{\lambda_{0,i,S}}$
Return y

Figure 9: DDH-based DPRF construction. $\lambda_{0,i,S}$ are the Lagrange coefficients.

Putting (19) and (20) together, we obtain that

$$\begin{aligned}\mathbf{Adv}^{\mathrm{dprf}}_{\mathsf{DP}^{\mathrm{PRF}}_{\mathsf{F}}}(\mathcal{A}) &= 2 \cdot \mathbf{P}\big[\mathbf{G}_0\big] - 1 \\ &= 2 \cdot (\mathbf{P}\big[\mathbf{G}_0\big] - \mathbf{P}\big[\mathbf{G}_1\big] + \mathbf{P}\big[\mathbf{G}_1\big]) - 1 \\ &= 2 \cdot \mathbf{Adv}^{\text{d-prf}}_{\mathsf{F}}(\mathcal{B})\ .\end{aligned}$$

□

6.1.4 Instantiating DDH-based DPRF and DVRF

Here we provide our instantiation of DDH-based DPRF and DVRF. We note that the other DPRF instantiation involving XORs of PRF values also satisfies our definition. We have provided details of XOR DPRF in Appendix 6.1.3.

Notation. Throughout this section, we let $\mathbb{G}$ be a group of prime order p with generator g. We require there be an efficient sampling algorithm that samples elements uniformly from $\mathbb{G}$ and we write $h \leftarrow\!\!\$\ \mathbb{G}$ to denote such sampling. For a vector $\mathbf{a}$ its i-th element is denoted by a_i. The inner product of two vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted by $\langle \mathbf{a}, \mathbf{b} \rangle$. For a matrix $\mathbf{A} \in \mathbb{Z}_p^{\alpha\times\beta}$ with the (i,j)-th entry denoted by a_{ij} the matrix $g^{\mathbf{A}}$ is given by $\begin{bmatrix} g^{a_{11}} & \ldots & g^{a_{1\beta}} \\ \vdots & \ddots & \vdots \\ g^{a_{\alpha 1}} & \ldots & g^{a_{\alpha\beta}} \end{bmatrix}$. For a matrix $\mathbf{A}$ of dimension $\alpha \times \beta$ we define the matrix $\mathbf{A}^{(-i)}$ of dimension $\alpha \times (\beta - 1)$ to be $\mathbf{A}$ without its i-th column. For a vector $\mathbf{b}$ of dimension α, $[\mathbf{A} \mid \mathbf{b}]$ denotes a matrix of dimension $\alpha \times (\beta + 1)$ that is $\mathbf{A}$ with the additional $(\beta+1)$-th column $\mathbf{b}$. Similarly row-extension of $\mathbf{A}$ is a matrix of dimension $(\alpha + 1) \times \beta$ and is denoted by $\begin{bmatrix} \mathbf{A} \\ \mathbf{c} \end{bmatrix}$ for a vector $\mathbf{c}$ of dimension β. $\mathbf{A}_{(-i)}$ denotes a matrix of dimension $(\alpha - 1) \times \beta$ that has the i-th row removed. The tensor product between two vectors is denoted $\otimes$.

6.1.5 A DPRF from DDH [NPR99, AMMR18a].

Naor et al.'s [NPR99] DDH-based DPRF construction $\mathsf{DP}^{\mathrm{DDH}}_{\mathbb{G},g}$ is depicted in Figure 9. Naor et al. prove the construction's security for semi-honest adversaries. Agrawal et al. [AMMR18a] prove the DDH-based DPRF construction secure against malicious attackers under the DDH assumption using their single-challenge security definition. We revisit the proof of the DDH-based DPRF to provide a more modular and simpler proof with concrete analysis based on a new variant of the Matrix-DDH assumption [EHK+13], that we call the Tensor-DDH (TDDH) assumption. We show

that this assumption is as hard as DDH. Let p be a prime. Let $\mathcal{D}^{\otimes}_{\alpha,\beta}$ denote the distribution of matrix $\mathbf{A} = (\mathbf{s}, 1) \otimes \mathbf{r} \in \mathbb{Z}_p^{(\alpha+1)\times\beta}$ such that $(\mathbf{s}, \mathbf{r}) \leftarrow\!\!\$\ \mathbb{Z}_p^{\alpha} \times \mathbb{Z}_p^{\beta}$ for $\alpha, \beta \in \mathbb{N}$. We define the (α, β)-*tensor-DDH* advantage of some adversary $\mathcal{A}$ in group $\mathbb{G} = \langle g \rangle$ of prime order p to be

$$\mathbf{Adv}^{(\alpha,\beta)\text{-tddh}}_{\mathbb{G},g}(\mathcal{A}) := \mathbf{P}\left[\mathcal{A}([g^{\mathbf{A}} \mid g^{\mathbf{A}\cdot\mathbf{u}}])\right] - \mathbf{P}\left[\mathcal{A}([g^{\mathbf{A}} \mid g^{\mathbf{z}}])\right],$$

where the probability is taken over $\mathbf{A} \leftarrow\!\!\$\ \mathcal{D}^{\otimes}_{\alpha,\beta}$, $\mathbf{u} \leftarrow\!\!\$\ \mathbb{Z}_p^{\beta}$, $\mathbf{z} \leftarrow\!\!\$\ \mathbb{Z}_p^{\alpha+1}$, and the randomness of $\mathcal{A}$.

Let us denote the advantage of a adversary $\mathcal{A}$ in breaking the DDH assumption over a group $\mathbb{G}$ generated by g as $\mathbf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{A})$. We prove that TDDH is at least as hard as DDH.

Lemma 1 (Hardness of TDDH from DDH)**.** *Let $\mathbb{G}$ be a group of prime order p and $g \leftarrow\!\!\$\ \mathbb{G}$ a generator. Then, for any $\alpha, \beta \in \mathbb{N}$ and any adversary $\mathcal{A}$, there exists another adversary $\mathcal{B}$ that runs in similar time as $\mathcal{A}$ such that*

$$\mathbf{Adv}^{(\alpha,\beta)\text{-tddh}}_{\mathbb{G},g}(\mathcal{A}) \leq 2\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g}(\mathcal{B})$$

Proof. The adversary $\mathcal{A}$ is given an instance of TDDH, a matrix $\mathbf{M} = [g^{\mathbf{A}} \mid g^{\mathbf{b}}] \in \mathbb{G}^{(\alpha+1)\times(\beta+1)}$ where $\mathbf{b}$ is either a uniformly chosen vector in $\mathbb{Z}_p^{\alpha+1}$ or equal to $\mathbf{A} \cdot \mathbf{u}$ for a uniformly chosen vector $\mathbf{u} \in \mathbb{Z}_p^{\beta}$ and $\mathbf{A} = (\mathbf{s}, 1) \otimes \mathbf{r}$ for some uniform random $\mathbf{s} \in \mathbb{Z}_p^{\alpha}, \mathbf{r} \in \mathbb{Z}_p^{\beta}$. Therefore, each element of $\mathbf{A}$, $A_{i,j}$ can be written as $s_i r_j$ where $s_{\alpha+1} = 1$. Let us call the mental experiment when $\mathcal{A}$ is given a uniform random $\mathbf{b}$ the Ideal game and a $\mathbf{b}$ in the column space of $\mathbf{A}$ the Real game. Let us define an intermediate security game (say, Hyb) where the adversary is given a matrix $\mathbf{M} = [g^{\mathbf{A}} \mid g^{\mathbf{b}}] \in \mathbb{G}^{(\alpha+1)\times(\beta+1)}$ such that $b_{\alpha+1} = \langle \mathbf{u}, \mathbf{r} \rangle$ for a uniform random $\mathbf{u} \in \mathbb{Z}_p^{\beta}$, but for all $i \in [\alpha]$, b_i is sampled uniformly at random from $\mathbb{Z}_p$. Now, given a DDH challenge (g^x, g^y, g^z) where z is either equal to xy mod p or a random element in $\mathbb{Z}_p$, we simulate either Real or Hyb (when $z = xy$ or uniform in $\mathbb{Z}_p$ respectively) as follows:

1. Choose uniform random $r_1, \ldots, r_\beta \leftarrow \mathbb{Z}_p$ and then define $g^{\langle \mathbf{u}, \mathbf{r} \rangle} := g^x$ for an unknown $\mathbf{u}$.
2. For all $i \in [\alpha]$, choose uniform random $w_i^{(1)}, w_i^{(2)} \leftarrow\!\!\$\ \mathbb{Z}_p$ and then define $g^{s_i} := g^{w_i^{(1)} + w_i^{(2)} y}$ for unknown s_i.
3. For all $i \in [\alpha]$ and $j \in [\beta]$ compute $g^{A_{ij}} := (g^{s_i})^{r_j}$ by raising each known r_j to the power of g^{s_i} for an unknown s_i.
4. For all $i \in [\alpha]$ compute $g^{b_i} := g^{w_i^{(1)} x + w_i^{(2)} z}$ for unknown b_i.
5. For all $j \in [\beta]$ define $g^{A_{\alpha+1 j}} := g^{r_i}$ and finally define $g^{b_{\alpha+1}} := g^x$.

Now, we observe that when $z = xy$ then Real is simulated perfectly. To see this, notice that, since each $w_i^{(1)}, w_i^{(2)}$ are chosen randomly, the distribution of $w_i^{(1)} + w_i^{(2)} y$ is identical to a uniform random element s_i in $\mathbb{Z}_p$. Furthermore, in that case we can re-write $b_i = w_i^{(1)} x + w_i^{(2)} xy = s_i \langle \mathbf{u}, \mathbf{r} \rangle$. On the other hand when z is uniformly random in $\mathbb{Z}_p$, the distribution of each $b_i = w_i^{(1)} x + w_i^{(2)} z$ is uniform random in $\mathbb{Z}_p$ for $i \in [\alpha]$. Also note that, in both cases the element $b_{\alpha+1}$ is implicitly defined to be x.

Next, given a DDH tuple (g^x, g^y, g^z) we will simulate either experiment Hyb or Ideal if $z = xy$ or $z \leftarrow\!\!\$\ \mathbb{Z}_p$ respectively. This is done as follows:

1. Choose $s_1, \ldots, s_\alpha \leftarrow\!\!\$\ \mathbb{Z}_p$ uniformly at random.

2. Choose $r_2, \ldots, r_\beta \leftarrow\!\!\$\ \mathbb{Z}_p$ and $u_2, \ldots, u_\beta \leftarrow\!\!\$\ \mathbb{Z}_p$ uniformly at random.
3. Define $g^{r_1} := g^x$ for an unknown r_1 and $g^{u_1} := g^y$ for an unknown u_1.
4. Define $g^{A_{ij}} = (g^{r_j})^{s_i}$ for $i \in [\alpha], j \in [\beta]$.
5. Choose $b_1, \ldots b_\alpha \leftarrow\!\!\$\ \mathbb{Z}_p$ uniformly at random.
6. Finally define $g^{b_{\alpha+1}} := g^z \cdot g^{\sum_{i=2}^{\beta} r_i u_i}$.

Now, if $z = xy$, then the distribution of $[\mathbf{A} \mid \mathbf{b}]$ is identical to HYB as then $b_{\alpha+1} = \sum_{i=1}^{\beta} r_i u_i$. On the other hand, if z is uniformly random in $\mathbb{Z}_p$, the $b_{\alpha+1}$ is statistically close to uniformly at random and hence the distribution of $[\mathbf{A} \mid \mathbf{b}]$ is identical with IDEAL.

Therefore, the overall advantage in distinguishing REAL and IDEAL is bounded by $2 \cdot \mathbf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{B})$. This concludes the proof of the lemma.

□

The security of the DPRF is formalized as follows.

Theorem 4. *Let $\mathbb{G} = \langle g \rangle$ be a group of prime order p. Let $\mathcal{A}$ be a DPRF adversary against $\mathsf{DP}^{\mathrm{DDH}}_{\mathbb{G},g}$. Let $q_{\mathrm{EVAL}}, q_{\mathrm{HASH}}, q_{\mathrm{CHL}}$ be the number of queries $\mathcal{A}$ makes to the* EVAL, HASH, CHL *oracles, respectively. The proof gives a (q_{EVAL}, t)-TDDH adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathrm{dprf}}_{\mathsf{DP}^{\mathrm{DDH}}_{\mathbb{G},g},n,t}(\mathcal{A}) \leq 2q_{\mathrm{CHL}}(q_{\mathrm{EVAL}} + q_{\mathrm{HASH}}) \cdot \mathbf{Adv}^{(q_{\mathrm{EVAL}}+q_{\mathrm{HASH}},t)\text{-}\mathsf{tddh}}_{\mathbb{G},g}(\mathcal{B}) \tag{21}$$

Proof. First notice that we are in a model of malicious corruption where the adversary can corrupt parties after seeing the public parameters, but before start making any query. Nonetheless, the public parameters has not information about the secrets and hence the choice of corrupt parties is also independent of the secrets. Therefore, in a mental game it is fine for the challenger to choose the secret keys after receiving the corrupt sets.

Let us assume that without loss of generality the adversary corrupts $\ell < t$ specific parties with identities from $n - \ell + 1$ to n. This leaves the gap between the threshold and the corrupt parties $t - \ell \geq 1$. Let us set $\tau = t - \ell - 1$; all parties with identity in $[\tau]$ honest as $\tau < n - \ell$.

Proof intuition. First the reduction needs to correctly guess each challenge query ahead of time for which would incur a security loss of factor $(q_{\mathrm{HASH}}+q_{\mathrm{EVAL}})$. This seems necessary to take into account for evaluation/random oracle queries on challenge inputs made ahead of respective challenge queries. We will do that one challenge at a time and by union bound this leads to an additional security loss of a multiplicative factor q_{CHL}. Now, for a fixed challenge and conditioning on the correct guess, assume that it is sufficient to show that (see below for the argument why this is sufficient) the following two distributions are computationally indistinguishable: the real distribution $[g^{\mathbf{A}} \mid g^{\mathbf{Ar}}]$ and the random distribution $\left[g^{\mathbf{A}} \mid \begin{bmatrix} g^u \\ g^{\mathbf{A}_{[-1]}\mathbf{r}'} \end{bmatrix}\right]$ where $\mathbf{A} \leftarrow \mathcal{D}^{\otimes}_{q_{\mathrm{EVAL}}+q_{\mathrm{HASH}},\tau+1}$, $\mathbf{r} \leftarrow\!\!\$\ \mathbb{Z}_p^{\tau+1}$ and $\mathbf{r}' \leftarrow\!\!\$\ \mathbb{Z}_p^{\tau}$ and $u \leftarrow\!\!\$\ \mathbb{Z}_p$. By $(q_{\mathrm{EVAL}} + q_{\mathrm{HASH}}, t)$-TDDH we have that: the real distribution is indistinguishable with another distribution (let us call it the ideal distribution) $[g^{\mathbf{A}} \mid g^{\mathbf{z}}]$ for $\mathbf{z} \leftarrow\!\!\$\ \mathbb{Z}_p^{(q_{\mathrm{EVAL}}+q_{\mathrm{HASH}})}$. However, we can use again $(q_{\mathrm{EVAL}} + q_{\mathrm{HASH}}, t)$-tensor DDH to show indistinguishability between the random and ideal distribution. Hence, we need to apply the $(q_{\mathrm{EVAL}}q_{\mathrm{HASH}}, t)$-tensor DDH assumption twice which incurs an additional loss of a factor 2.

Finally we argue that why it is sufficient to show that the real and random distributions are computationally indistinguishable. Note that, given the real distribution one can perfectly simulate the entire real experiment in the DPRF game (when $b = 0$ in the description in Figure 5). The

$(i+1, j)-$th entry for the matrix would be the response to the j-th evaluation query on party $i \in [\tau]$ is a unique index assigned to. The first row of the matrix consists of values with respect to the final DPRF values. Note that for $\mathbf{A} = [s_0 \mid \mathbf{s} \mid 1] \otimes \mathbf{r}$ we implicitly have that $\mathrm{H}(x_i) = g^{r_i}$, when H is a random oracle. The values s_j are implicitly set to the shares of party $j \in [\tau]$ whereas the value s_0 is the original secret-key. Combining this with other ℓ independently chosen values $s_{n-\ell+1}, \ldots, s_n$ we have t points in total for constructing a t-degree polynomial Similarly the random distribution can be used to perfectly simulate the DPRF security game with $b = 1$. This concludes the intuitions.

First let us define two mental games: Real which is the DPRF security game (see Fig, 5 conditioned on $b = 0$, in that all challenge queries are output of the DPRF; and a game corresponding to $b = 1$ called Rand. Overall there are q_{Chl} challenge queries to Chl. So we define $(q_{\text{Chl}} + 1)$ hybrid games Hyb_i in which all $j \in [i-1]$-th challenge queries are responded with random values and all challenge queries in $\{i, \ldots, q_{\text{Chl}}\}$ are responded with real DPRF outputs using Combine. Let us denote the challenge values by $\{x_1^\star, \ldots, x_{q_{\text{Chl}}}^\star\}$ Clearly Hyb_1 is identical to Real and $\text{Hyb}_{q_{\text{Chl}}+1}$ is identical to Rand. For each hybrid Hyb_i we define an additional hybrid $\overline{\text{Hyb}}_i$ which is the same as Hyb_i except that: in response to the evaluation queries to $\text{Eval}(j, x_i^\star)$ on the i-th challenge query to a honest party $j \in [\tau]$ is responded with a random value from $\mathbb{G}$.

We state two lemma now.

Lemma 2. *If there is a attacker $\mathcal{A}_i$ that can distinguish between hybrids Hyb_{i+1} and $\overline{\text{Hyb}}_i$ making q_{Eval} Eval queries and q_{Hash} Hash queries for all $i \in [q_{\text{Chl}}]$ with advantage δ_i, we are able to construct another attacker $\mathcal{B}_i$ running in similar time as $\mathcal{A}_i$ such that*

$$\delta_i \leq (q_{\text{Hash}} + q_{\text{Eval}}) \cdot \mathbf{Adv}_{\mathbb{G},g}^{(q_{\text{Eval}}+q_{\text{Hash}},\tau+1)\text{-tddh}}(\mathcal{B}_i)$$

Lemma 3. *If there is an attacker $\mathcal{A}_i'$ that can distinguish between hybrids Hyb_i and $\overline{\text{Hyb}}_i$ making q_{Eval} Eval queries and q_{Hash} Hash queries for all $i \in [q_{\text{Chl}}]$ with advantage ε_i, we are able to construct another attacker $\mathcal{B}_i'$ that runs in similar time as $\mathcal{A}_i'$ such that*

$$\varepsilon_i \leq (q_{\text{Hash}} + q_{\text{Eval}}) \cdot \mathbf{Adv}_{\mathbb{G},g}^{(q_{\text{Eval}}+q_{\text{Hash}},\tau)\text{-tddh}}(\mathcal{B}_i')$$

Notice that, combining the above two lemmas and applying this for all $i \in [q_{\text{Chl}}]$ we obtain the desired statement. In the rest we provide the proofs of these two lemma.

Proof of Lemma 2. We need $\mathcal{B}_i$ to simulate the games either $\overline{\text{Hyb}}_i$ or Hyb_{i+1} for $\mathcal{A}_i$ from the $(\tau + 1, q_{\text{Hash}} + q_{\text{Eval}})$-TDDH challenges. $\mathcal{B}_i$ works as follows:

1. Let $q_{\text{Eval}} + q_{\text{Hash}} = \gamma$. On receiving the public parameters $(p, g, \mathbb{G})$ forward them to $\mathcal{A}_i$. Receive the TDDH challenge $\mathbf{M} = [g^{\mathbf{A}} \mid g^{\mathbf{b}}]$ where $\mathbf{b}$ is either uniform random (random instance) or $\mathbf{A} \cdot \mathbf{u}$ for some uniform random $\mathbf{u}$ (real instance). We expand $\mathbf{A}$ and $\mathbf{b}$ as follows for notational convenience:

$$\begin{bmatrix} r_1 s_0 & \ldots & r_\gamma s_0 \\ \vdots & \ddots & \vdots \\ r_1 s_\tau & \ldots & r_\gamma s_\tau \\ r_1 & \ldots & r_\gamma \end{bmatrix} \in \mathbb{Z}_p^{(\tau+2)\times\gamma}$$

$$\mathbf{b} = [b_0 \mid b_1 \mid \cdots b_{\tau+1}] \in \mathbb{Z}_p^{\tau+2}$$

2. Receive the set of corrupt parties, which is the parties with identities in $\{n-\ell+1, n\}$ as assumed without loss of generality. Choose random values $s_{n-\ell+1}, \ldots, s_n \in \mathbb{Z}_p$ and give out to the attacker as the shares of corrupt parties. Note that these values, together with $s_0, s_1, \ldots, s_\tau$, are t random points that define the t-degree polynomial used for secret-sharing the value s_0 which is the implicit DPRF secret-key. The key shares for parties in $[\tau]$ are set to be $\{s_j\}_{j\in[\tau]}$. Initiate a counter $\eta = 0$ for simulating hash and evaluation queries.

3. Simulate the hash queries $\textsc{Hash}(x)$ and $\textsc{Eval}(j, x)$ queries (made in any order) as follows: guess an index $j^\star \leftarrow\!\!\$\, [q_{\textsc{Hash}} + q_{\textsc{Chl}}]$ for the i-th challenge query. If this is a new x that never appeared earlier, increment the counter η and if $\eta \neq j^\star$ return g^{r_η} for hash query or $g^{s_j r_\eta}$ for eval query from $\mathbf{M}$ implicitly. For the $j^\star$-th one, if it is a hash query reply with $g^{b_{\tau+1}}$, otherwise with $g^{s_j b_j}$ implicitly. If this is a repeated query reply consistently with that value without incrementing the counter.

4. Simulate the j-th challenge query $\textsc{Chl}$ by returning random values for $j < i$ and correct DPRF values for $j > i$. For the i-th challenge return b_0.

Clearly when the TDDH instance is real $\mathcal{B}$ simulates $\textsc{Hyb}_{i+1}$ perfectly and when it is random then it simulates $\overline{\textsc{Hyb}}_i$. This concludes the proof.

Proof of Lemma 3. $\mathcal{B}'$ needs to simulate $\textsc{Hyb}_i$ and $\overline{\textsc{Hyb}}_i$ from real and random instances of TDDH respectively. First note that, the hybrids $\textsc{Hyb}_i$ and $\overline{\textsc{Hyb}}_i$ gets exactly same response from the challenge oracle in all queries. The response only differs on evaluation queries on the i-th challenge $x_i^\star$. This is why the matrix $\mathbf{A}$ has one less row than the instance used in the previous proof. The proof is very similar to the previous proof with some necessary changes. We highlight the changes below in *red*.

1. Let $q_{\textsc{Eval}} + q_{\textsc{Hash}} = \gamma$. On receiving the public parameters $(p, g, \mathbb{G})$ forward them to $\mathcal{A}_i$. Receive the TDDH challenge $\mathbf{M} = [g^{\mathbf{A}} \mid g^{\mathbf{b}}]$ where $\mathbf{b}$ is either uniform random (random instance) or $\mathbf{A} \cdot \mathbf{u}$ for some uniform random $\mathbf{u}$ (real instance). We expand $\mathbf{A}$ as follows:

$$\begin{bmatrix} r_1 s_1 & \ldots & r_\gamma s_1 \\ \vdots & \ddots & \vdots \\ r_1 s_\tau & \ldots & r_\gamma s_\tau \\ r_1 & \ldots & r_\gamma \end{bmatrix} \in \mathbb{Z}_p^{(\tau+1)\times\gamma}$$

$$\mathbf{b} = [b_1 \mid \cdots b_{\tau+1}] \in \mathbb{Z}_p^{\tau+2}$$

2. Receive the set of corrupt parties, which is the parties with identities in $\{n-\ell+1, n\}$ as assumed without loss of generality. Choose random values $s_0, s_{n-\ell+1}, \ldots, s_n \in \mathbb{Z}_p$ and give out to the attacker as the shares of corrupt parties. Note that these values, together with $s_0, s_1, \ldots, s_\tau$, are t random points that define the t-degree polynomial used for secret-sharing the value s_0 which is the chosen DPRF secret-key. The key shares for parties in $[\tau]$ are set to be $\{s_j\}_{j\in[\tau]}$. Initiate a counter $\eta = 0$ for simulating hash and evaluation queries.

3. Simulate the hash queries $\textsc{Hash}(x)$ and $\textsc{Eval}(j, x)$ queries (made in any order) as follows: guess an index $j^\star \leftarrow\!\!\$\, [q_{\textsc{Hash}} + q_{\textsc{Chl}}]$ for the i-th challenge query. If this is a new x that never appeared earlier, increment the counter η and if $\eta \neq j^\star$ return g^{r_η} for hash query or $g^{s_j r_\eta}$ for eval query from $\mathbf{M}$ implicitly. For the $j^\star$-th one, if it is a hash query reply with $g^{b_{\tau+1}}$, otherwise with $g^{s_j b_j}$ implicitly. If this is a repeated query reply consistently with that value without incrementing the counter.

Scheme $\mathsf{DV}^{\mathrm{DDH}}_{\mathbb{G}}$

Proc $\mathsf{Setup}(n,t)$

$s \leftarrow\!\!\$\ \mathbb{Z}_p$
$\mathsf{pp}_{\mathsf{com}} \leftarrow \mathsf{Setup}_{\mathsf{com}}$
For $j \in [t-1]$ do $c_j \leftarrow\!\!\$\ \mathbb{Z}_p$
For $i \in [n]$
  $s_i \leftarrow s + \sum_{j\in[t-1]} c_j \cdot i^j$
  $r_i \leftarrow\!\!\$\ \{0,1\}^{\kappa}$
  $\gamma_i := \mathsf{Com}(s_i, \mathsf{pp}_{\mathsf{com}}; r_i)$
  $\mathsf{sk}_i := (s_i, r_i)$
Return $((\mathsf{sk}_1, \ldots, \mathsf{sk}_n),$
  $(\mathsf{pp}_{\mathsf{com}}, \gamma_1, \ldots \gamma_n))$

Proc $\mathsf{Eval}(\mathsf{pp}, \mathsf{sk}, x)$

$w := \mathrm{H}(x)$; $z := w^s$; $(s,r) := \mathsf{sk}$
$\pi \leftarrow \mathsf{NIZK.Prv}(\sigma, \mu)$
Return (z, π)

Proc $\mathsf{Combine}(\mathsf{pp}, L)$

If $|L| < t$ then return $\perp$
$\{(i, z_i, \pi_i)\} \leftarrow L$
$y \leftarrow \prod_i (z_i)^{\lambda_{0,i,S}}$
Return y

Proc $\mathsf{Verify}(\mathsf{pp}, x, L)$

$\{(i, z_i, \pi_I)\} \leftarrow L$
$w = \mathrm{H}(x)$
For $i \in L$ do:
  $\sigma_i := (w, \gamma_i)$
  $d_i := \mathsf{NIZK.Ver}(\sigma_i, \pi_i)$
Return $\wedge_i d_i$

Figure 10: DDH-based DVRF. $\lambda_{0,i,S}$ are the Lagrange coefficients.

4. Simulate the j-th challenge query $\textsc{Chl}$ by returning random values for $j \leq i$ and correct DPRF values for $j > i$ easily using the knowledge of s_0 without using the TDDH instance.

Clearly when the TDDH instance is real $\mathcal{B}$ simulates $\textsc{Hyb}_i$ perfectly and when it is random then it simulates $\overline{\textsc{Hyb}_i}$. This concludes the proof.

□

Combining Theorem 5 and Lemma 1, we obtain the following corollary.

Corollary 1. *Let $\mathbb{G} = \langle g \rangle$ be a group of prime order p. Let $\mathcal{A}$ be a DPRF adversary against $\mathsf{DP}^{\mathrm{DDH}}_{\mathbb{G},g}$. Let $q_{\textsc{Eval}}, q_{\textsc{Hash}}, q_{\textsc{Chl}}$ be the number of queries $\mathcal{A}$ makes to the $\textsc{Eval}, \textsc{Hash}, \textsc{Chl}$ oracles, respectively. Then the proof gives a DDH adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathrm{dprf}}_{\mathsf{DP}^{\mathrm{DDH}}_{\mathbb{G},g},n,t}(\mathcal{A}) \leq 4q_{\textsc{Chl}}(q_{\textsc{Hash}} + q_{\textsc{Eval}}) \cdot \mathbf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{B}) \tag{22}$$

6.1.6 Extension to DVRF.

The strongly-secure publicly verifiable version of the DPRF presented in DiSE [AMMR18a], which in turn builds on the Naor et al. [NPR99] construction, satisfies our definition. The algorithm Verify would simply verify the NIZK proofs included in each partial evaluation. For completeness, we provide the construction in Figure 10. It is an extension of the DPRF construction provided in Figure 9. It additionally uses a trapdoor commitment scheme $(\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ and a (simulation sound) NIZK proof system[4] $(\mathsf{NIZK.Prv}, \mathsf{NIZK.Ver})$ with perfect soundness and failure probability of zero-knowledge simulator $\varepsilon_{\mathsf{zk}}$. In particular, the NIZK system is used to prove relations of the form $\{\exists\ (s,r)\ :\ z = w^s \wedge \gamma = \mathsf{Com}(s, \mathsf{pp}_{\mathsf{com}}; r)\}$ with statement $\sigma := (z, w, \gamma, \mathsf{pp}_{\mathsf{com}})$ and witness $\mu := (s,r)$. For formal definitions of these we refer to the full version of DiSE [AMMR18b].

It is easy to see that the robustness advantage will be bounded by the failure probability of NIZK simulation soundness for t proofs. But since we use a NIZK with 0 soundness we get $\mathbf{Adv}^{\mathrm{dvrf-rob}}_{\mathsf{DP},n,t}(\mathcal{A}) = 0$. Furthermore, to prove the DVRF security as per Figure 6a, a reduction would need to produce simulated proofs. So the adversary's advantage will increase by an additive factor of $t \cdot q_{\textsc{Chl}} \cdot \varepsilon_{\mathsf{zk}}$. The security can be formalized as follows.

[4]The concrete implementation will be assuming random oracles, but here we keep it implicit for simplicity.

Scheme $\mathsf{TAE2}[\mathsf{DP}, \mathsf{SIG}, k]$

Proc $\mathsf{Split}_{\mathsf{enc}}(j, m, S)$
Require $|S| \geq t$
Let S' be a t-sized subset of S
$r \leftarrow \{0,1\}^k$; $\alpha \leftarrow \mathrm{H}(m\|r)$
$\mathsf{st} \leftarrow (j, \alpha, m, r)$
Return $(\{(i, \alpha)\}_{i \in S'}, \mathsf{st})$

Proc $\mathsf{Eval}_{\mathsf{enc}}(\mathsf{sk}_i, j, \alpha)$
$(\mathsf{sk}_{\mathsf{DP},i}, \mathsf{sk}_{\mathsf{SIG},i}) \leftarrow \mathsf{sk}_i$
$y \leftarrow \mathsf{DP.Eval}(\mathsf{sk}_{\mathsf{DP},i}, j\|\alpha)$
$\sigma \leftarrow \mathsf{SIG.PartSign}(\mathsf{sk}_{\mathsf{SIG},i}, j\|\alpha)$
Return (y, σ)

Proc $\mathsf{Combine}_{\mathsf{enc}}(R, \mathsf{st})$
$(j, \alpha, m, r) \leftarrow \mathsf{st}$; $\{(i, (y_i, \sigma_i))\} \leftarrow R$
$\beta \leftarrow \mathsf{DP.Combine}(\{(i, y_i)\})$
$\sigma \leftarrow \mathsf{SIG.CombSig}(\mathsf{vk}, \{(i, \sigma_i)\})$
If $\mathsf{VerSig}(\mathsf{vk}, j\|\alpha, \sigma) \neq 1$ then return $\perp$
$e \leftarrow \mathrm{G}(\beta) \oplus (m\|r)$
Return (j, α, σ, e)

Proc $\mathsf{Setup}(n, t)$
$(\llbracket \mathsf{sk}_{\mathsf{DP}} \rrbracket_n, \mathsf{pp}_{\mathsf{DP}}) \leftarrow\!\!\$\ \mathsf{DP.Setup}(n, t)$
$(\llbracket \mathsf{sk}_{\mathsf{SIG}} \rrbracket_n, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{SIG.Setup}(n, t)$
For $i \in [n]$ do $\mathsf{sk}_i \leftarrow (\mathsf{sk}_{\mathsf{DP},i}, \mathsf{sk}_{\mathsf{SIG},i})$
$\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{DP}}, \mathsf{vk})$
Return $((\mathsf{sk}_1, \ldots, \mathsf{sk}_n), \mathsf{pp})$

Proc $\mathsf{Split}_{\mathsf{dec}}(j, c, S)$
Require $|S| \geq t$
Let S' be a t-sized subset of S
$(j, \alpha, \sigma, e) \leftarrow c$; $\mathsf{st} \leftarrow (j, \alpha, e)$
Return $(\{(i, (j\|\alpha, \sigma))\}_{i \in S'}, \mathsf{st})$

Proc $\mathsf{Eval}_{\mathsf{dec}}(\mathsf{sk}_i, j, x)$
$(\mathsf{pp}_{\mathsf{DP}}, \mathsf{vk}) \leftarrow \mathsf{pp}$; $(j\|\alpha, \sigma) \leftarrow x$
$(\mathsf{sk}_{\mathsf{DP},i}, \mathsf{sk}_{\mathsf{SIG},i}) \leftarrow \mathsf{sk}_i$
If $\mathsf{SIG.VerSig}(\mathsf{vk}, j\|\alpha, \sigma)$ then
  Return $\mathsf{DP.Eval}(\mathsf{sk}_{\mathsf{DP},i}, j\|\alpha)$
Else return $\perp$

Proc $\mathsf{Combine}_{\mathsf{dec}}(R, \mathsf{st})$
$(j, \alpha, e) \leftarrow \mathsf{st}$
$\beta \leftarrow \mathsf{DP.Combine}(R)$
$m\|r \leftarrow \mathrm{G}(\beta) \oplus e$
If $(\mathrm{H}(m\|r) \neq \alpha)$ then return $\perp$
Return m

Figure 11: DPRF & threshold signature based TAE scheme.

Theorem 5. *Let $\mathbb{G} = \langle g \rangle$ be a group of prime order p. Let $\mathcal{A}$ be a DVRF adversary against $\mathsf{DV}^{\mathrm{DDH}}_{\mathbb{G},g}$. Let $q_{\mathrm{EVAL}}, q_{\mathrm{HASH}}, q_{\mathrm{CHL}}$ be the number of queries $\mathcal{A}$ makes to the* EVAL, HASH, CHL *oracles, respectively. The proof gives a (q_{EVAL}, t)-TDDH adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathrm{dvrf}}_{\mathsf{DV}^{\mathrm{DDH}}_{\mathbb{G},g},n,t}(\mathcal{A}) \leq 2q_{\mathrm{CHL}}(q_{\mathrm{EVAL}} + q_{\mathrm{HASH}}) \cdot \mathbf{Adv}^{(q_{\mathrm{EVAL}}+q_{\mathrm{HASH}},t)\text{-}\mathsf{tddh}}_{\mathbb{G},g}(\mathcal{B}) \ + t \cdot q_{\mathrm{CHL}} \cdot \varepsilon_{\mathsf{zk}}. \tag{23}$$

The proof of the above theorem is similar to the proof of Theorem 5 and is therefore omitted. Combining Theorem 5 and Lemma 1, we obtain:

Corollary 2. *Let $\mathbb{G} = \langle g \rangle$ be a group of prime order p. Let $\mathcal{A}$ be a DVRF adversary against $\mathsf{DV}^{\mathrm{DDH}}_{\mathbb{G},g}$. Let $q_{\mathrm{EVAL}}, q_{\mathrm{HASH}}, q_{\mathrm{CHL}}$ be the number of queries $\mathcal{A}$ makes to the* EVAL, HASH, CHL *oracles, respectively. Then the proof gives a DDH adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathrm{dvrf}}_{\mathsf{DV}^{\mathrm{DDH}}_{\mathbb{G},g},n,t}(\mathcal{A}) \leq 4q_{\mathrm{CHL}}(q_{\mathrm{HASH}} + q_{\mathrm{EVAL}}) \cdot \mathbf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{B}) + t \cdot q_{\mathrm{CHL}} \cdot \varepsilon_{\mathsf{zk}} \tag{24}$$

6.2 IND-RCCA TAE using DPRF and Threshold Signature

The construction TAE2 is parameterized by a DPRF DP, a threshold signature scheme SIG, and an integer k. The specification of the construction is given in Figure 11. We explain below the

high-level ideas of the scheme.

Keys. Each party holds a key share of the DPRF key and a key share of the threshold signature signing key.

Encryption. When initiator j is encrypting a message m, a commitment $\alpha = \mathrm{H}(m\|r)$ is generated for the message m by using hash function H (modeled as a random oracle with input space $\{0,1\}^*$ and output space $\mathsf{DP.In}$) and randomly generated r. The DPRF output, $\beta \leftarrow \mathsf{DP}(j\|\alpha)$, is used as an encryption key to encrypt message m and randomness r into $c \leftarrow \mathrm{G}(\beta) \oplus (m\|r)$, where G is a random oracle with input space $\{0,1\}^k$ and output space $\{0,1\}^\infty$. Meanwhile, a threshold signature σ on $j\|\alpha$ is also generated using SIG. The final ciphertext is (j, α, σ, c).

Decryption. When an initiator j' is decrypting a ciphertext (j, α, σ, c) (note that j' does not have to equal j), each party i receives $(j\|\alpha, \sigma)$ and first verifies if σ is a valid signature on $j\|\alpha$ before returning the Eval output of DP. After reconstructing the DPRF output β, the initiator can recover the message m and randomness r. It checks if $\mathrm{H}(m\|r) = \alpha$. If the check succeeds, then plaintext m is returned. Otherwise, decryption fails and $\perp$ is returned.

Capturing the decryption criteria. In the scheme $\mathsf{TAE2}$, for a ciphertext $c = (j, \alpha, \sigma, e)$, $\mathsf{Split}_{\mathsf{dec}}(c, S)$ returns a list $(\{(i, (j\|\alpha, \sigma))\}_{i \in S'}$, where S' is a t-sized subset of S. The multiset $\mathsf{Eval\text{-}MSet}(c)$ just has the element $(j\|\alpha, \sigma)$ repeated t times. $\mathsf{Eval}_{\mathsf{dec}}$, with inputs sk_i and $(j\|\alpha, \sigma)$, outputs a non-$\perp$ value if σ is a valid signature. Therefore, for a set $\mathcal{CR}$, $\mathsf{Eval\text{-}MSet}(c, \mathcal{CR})$ is either $\mathsf{Eval\text{-}MSet}(c)$ (if σ is invalid) or the set with $(j\|\alpha, \sigma)$ repeated $t - |\mathcal{CR}|$ times.

Theorem 6. *Let* $\mathsf{TAE} = \mathsf{TAE2}[\mathsf{DP}, \mathsf{SIG}, k]$. *Let* $\mathcal{A}$ *be an adversary in the IND-RCCA game against* TAE *(Figure 2). Suppose* $\mathcal{A}$ *makes* q_{H} *and* q_{G} *queries to oracles* H *and* G*, respectively. Further, it makes* q_{enc}, q_{Eval} *and* q_{dec} *queries to encryption* ($\textsc{Split}_{\mathsf{enc}}$, $\textsc{Combine}_{\mathsf{enc}}$)*, evaluation* ($\textsc{Eval}_{\mathsf{enc}}$, $\textsc{Eval}_{\mathsf{dec}}$) *and decryption procedures* ($\textsc{Split}_{\mathsf{dec}}$, $\textsc{Combine}_{\mathsf{dec}}$)*, respectively. Then there exist adversaries* $\mathcal{B}$ *and* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{ind-rcca}}_{\mathsf{TAE},n,t}(\mathcal{A}) \leq \frac{(q_{\mathsf{enc}} + q_{\mathsf{dec}})^2}{|\mathsf{DP.In}|} + \frac{q_{\mathsf{enc}}^2 + 2 \cdot q_{\mathrm{H}} \cdot q_{\mathsf{enc}}}{2^k} + \frac{2 \cdot q_{\mathrm{G}} \cdot q_{\mathsf{enc}}}{|\mathsf{DP.Out}|} + 2 \cdot \mathbf{Adv}^{\text{sig}}_{\mathsf{SIG},n,t}(\mathcal{B}) + 2 \cdot \mathbf{Adv}^{\text{dprf}}_{\mathsf{DP},n,t}(\mathcal{C}) \,. \tag{25}$$

6.2.1 Proof of Theorem 6

TAE game. As the first step in the proof, we instantiate the RCCA security game from Figure 2 with the scheme $\mathsf{TAE2}$. The instantiated game, say $\mathbf{G}_1$, is described in full detail in Figure 12. Let us fix an adversary $\mathcal{A}$ in the game. We assume that $\mathcal{A}$ makes q_{H} and q_{G} queries to oracles H and G, respectively. It also makes q_{enc}, q_{Eval} and q_{dec} queries to encryption, evaluation and decryption procedures, respectively. Note that q_{enc}, for example, is the *total* number of queries made to $\textsc{Split}_{\mathsf{enc}}$ and $\textsc{Combine}_{\mathsf{enc}}$ (under challenge encryption sessions).

Unique RO outputs. The first modification we make to $\mathbf{G}_1$ is a simple one. Observe that the random function H appears in exactly two places in the game, in the oracles $\textsc{Split}_{\mathsf{enc}}$ and $\textsc{Combine}_{\mathsf{dec}}$. We replace H with a new function that behaves exactly like H but for distinct inputs in $\textsc{Split}_{\mathsf{enc}}$ and $\textsc{Combine}_{\mathsf{dec}}$, it produces distinct outputs. We call the modified game $\mathbf{G}_2$. Using a birthday bound, we have

$$\mathbf{P}[\mathbf{G}_1] - \mathbf{P}[\mathbf{G}_2] \leq \frac{(q_{\mathsf{enc}} + q_{\mathsf{dec}})^2}{2|\mathsf{DP.In}|} \,.$$

Unique encryption randomness. The second step is to modify how the randomness r_u (a k-bit string) is sampled in $\textsc{Split}_{\mathsf{enc}}$. We define a new game $\mathbf{G}_3$ where r_u is still chosen at random but

Game $\mathbf{G}^{\mathrm{ind-cca}}_{\mathsf{TAE2},n,t}(\mathcal{A})$

$b \leftarrow\!\!\$\ \{0,1\}$
$(\llbracket \mathsf{sk}_{\mathsf{DP}} \rrbracket_n, \mathsf{pp}_{\mathsf{DP}}) \leftarrow\!\!\$\ \mathsf{DP.Setup}(n,t)$
$(\llbracket \mathsf{sk}_{\mathsf{SIG}} \rrbracket_n, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{SIG.Setup}(n,t)$
For $i \in [n]$ do $\mathsf{sk}_i \leftarrow (\mathsf{sk}_{\mathsf{DP},i}, \mathsf{sk}_{\mathsf{SIG},i})$
$\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{DP}}, \mathsf{vk})$
$(\mathcal{CR}, \mathsf{st}_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}_0(\mathsf{pp})$
$b' \leftarrow\!\!\$\ \mathcal{A}_1^{\langle \mathbf{Proc} \rangle}(\mathsf{st}_{\mathcal{A}}, (\mathsf{sk}_i)_{i \in \mathcal{CR}})$
If $|\mathsf{DecSet}| > \left\lfloor \frac{\mathsf{ct}_{\mathsf{Eval}}}{t - |\mathcal{CR}|} \right\rfloor$ then $\mathsf{forgery} \leftarrow \textsc{True}$
If $(\exists c \in \mathsf{ChlCtxt} : \mathsf{Eval\text{-}MSet}(c, \mathcal{CR}) \subseteq Q_{\mathsf{dec}})$
  Return $(b'' \leftarrow\!\!\$\ \{0,1\}) \vee \mathsf{forgery}$
Return $(b = b') \vee \mathsf{forgery}$

Challenge encryption sessions

Proc $\textsc{Split}_{\mathsf{enc}}(\mathsf{id}, m_0, m_1, S)$
Require $\mathsf{id} \notin \mathcal{CR}$, $|m_0| = |m_1|$, $|S| \geq t$
$u \leftarrow u + 1$; $\mathsf{id}_u \leftarrow \mathsf{id}$
$m_{u,0} \leftarrow m_0$; $m_{u,1} \leftarrow m_1$
Let S' be a t-sized subset of S
$r_u \leftarrow \{0,1\}^k$; $\alpha_u \leftarrow \mathrm{H}(m_{u,b} \| r_u)$
$L_u \leftarrow \{(i, \alpha_u)\}_{i \in S'}$
Return $\{(i,x) \in L_u \mid i \in \mathcal{CR}\}$

Proc $\textsc{Combine}_{\mathsf{enc}}(u, \mathsf{rsp})$
For $(i,x) \in L_u$ with $i \notin \mathcal{CR}$ do
  $y_i \leftarrow \mathsf{DP.Eval}(\mathsf{sk}_{\mathsf{DP},i}, \mathsf{id}_u \| \alpha_u)$
  $\sigma_i \leftarrow \mathsf{SIG.PartSign}(\mathsf{sk}_{\mathsf{SIG},i}, \mathsf{id}_u \| \alpha_u)$
For $(i,x) \in L_u$ with $i \in \mathcal{CR}$ do
  $(y_i, \sigma_i) \leftarrow \mathsf{rsp}[(i,x)]$
$\beta \leftarrow \mathsf{DP.Combine}(\{(i, y_i)\})$
$\sigma \leftarrow \mathsf{SIG.CombSig}(\mathsf{vk}, \{(i, \sigma_i)\})$
If $\mathsf{VerSig}(\mathsf{vk}, \mathsf{id}_u \| \alpha_u, \sigma) \neq 1$ then return $\perp$
$e \leftarrow \mathrm{G}(\beta) \oplus (m_{u,b} \| r_u)$
$c_u \leftarrow (\mathsf{id}_u, \alpha_u, \sigma, e)$
If $m_{u,0} = m_{u,1}$ then
  $\mathsf{EncCtxt} \leftarrow \mathsf{EncCtxt} \cup \{c_u\}$
  $\mathsf{EncMsg} \leftarrow \mathsf{EncMsg} \cup \{m_{u,0}\}$
Else
  $\mathsf{ChlCtxt} \leftarrow \mathsf{ChlCtxt} \cup \{c_u\}$
  $\mathsf{ChlMsg} \leftarrow \mathsf{ChlMsg} \cup \{m_{u,0}, m_{u,1}\}$
Return c_u

Sessions initiated by adversary

Proc $\textsc{Eval}_{\mathsf{enc}}(\mathsf{eid}, \mathsf{id}, x)$
Require $\mathsf{eid} \notin \mathcal{CR}$, $\mathsf{id} \in \mathcal{CR}$
$\mathsf{ct}_{\mathsf{Eval}} \leftarrow \mathsf{ct}_{\mathsf{Eval}} + 1$; $\alpha \leftarrow x$
$y \leftarrow \mathsf{DP.Eval}(\mathsf{sk}_{\mathsf{DP},\mathsf{eid}}, \mathsf{id} \| \alpha)$
$\sigma \leftarrow \mathsf{SIG.PartSign}(\mathsf{sk}_{\mathsf{SIG},\mathsf{eid}}, \mathsf{id} \| \alpha)$
Return (y, σ)

Proc $\textsc{Eval}_{\mathsf{dec}}(\mathsf{eid}, \mathsf{id}, x)$
Require $\mathsf{eid} \notin \mathcal{CR}$, $\mathsf{id} \in \mathcal{CR}$
$Q_{\mathsf{dec}} \leftarrow Q_{\mathsf{dec}} \uplus \{x\}$
$(j \| \alpha, \sigma) \leftarrow x$
If $\mathsf{SIG.VerSig}(\mathsf{vk}, j \| \alpha, \sigma)$ then
  Return $\mathsf{DP.Eval}(\mathsf{sk}_{\mathsf{DP},\mathsf{eid}}, j \| \alpha)$
Else return $\perp$

Decryption sessions

Proc $\textsc{Split}_{\mathsf{dec}}(\mathsf{id}, c, S)$
Require $\mathsf{id} \notin \mathcal{CR}$, $|S| \geq t$
$v \leftarrow v + 1$; $\mathsf{id}_v \leftarrow \mathsf{id}$; $c_v \leftarrow c$
Let S' be a t-sized subset of S
$(j_v, \alpha_v, \sigma_v, e_v) \leftarrow c$
$L_v \leftarrow \{(i, (j_v \| \alpha_v, \sigma_v))\}_{i \in S'}$
Return $\{(i,x) \in L_v \mid i \in \mathcal{CR}\}$

Proc $\textsc{Combine}_{\mathsf{dec}}(v, \mathsf{rsp})$
For $(i,x) \in L_v$ with $i \notin \mathcal{CR}$ do
  If $\mathsf{SIG.VerSig}(\mathsf{vk}, j_v \| \alpha_v, \sigma_v)$ then
    $y_i \leftarrow \mathsf{DP.Eval}(\mathsf{sk}_{\mathsf{DP},i}, j_v \| \alpha_v)$
  Else $y_i \leftarrow \perp$
For $(i,x) \in L_v$ with $i \in \mathcal{CR}$ do
  $y_i \leftarrow \mathsf{rsp}[(i,x)]$
$\beta \leftarrow \mathsf{DP.Combine}(\{(i, y_i)\})$
$m_v \| r_v \leftarrow \mathrm{G}(\beta) \oplus e_v$
If $(\mathrm{H}(m_v \| r_v) \neq \alpha_v)$ then $m_v \leftarrow \perp$
Require $m_v \notin \mathsf{ChlMsg}$
$\mathsf{fresh} \leftarrow (m_v \notin \mathsf{EncMsg})$
If $m_v \neq \perp$ and fresh then
  $\mathsf{DecSet} \leftarrow \mathsf{DecSet} \cup \{m_v\}$
Return m_v

Figure 12: Game from Figure 2 instantiated with TAE2.

Proc $\textsc{Combine}^{\star}_{\mathsf{dec}}(v, \mathsf{rsp})$

If $((j_v \notin \mathcal{CR}) \wedge ((j_v, \alpha_v, \sigma^{\star}, e^{\star}) \notin \mathsf{EncCtxt} \cup \mathsf{ChlCtxt}$ for any $\sigma^{\star}, e^{\star}))$
  Return $\perp$
For $(i, x) \in L_v$ with $i \notin \mathcal{CR}$ do
  If $\mathsf{SIG.VerSig}(\mathsf{vk}, j_v \| \alpha_v, \sigma_v)$ then
    $y_i \leftarrow \mathsf{DP.Eval}(\mathsf{sk}_{\mathsf{DP},i}, j_v \| \alpha_v)$
  Else $y_i \leftarrow \perp$
For $(i, x) \in L_v$ with $i \in \mathcal{CR}$ do
  $y_i \leftarrow \mathsf{rsp}[(i, x)]$
$\beta \leftarrow \mathsf{DP.Combine}(\{(i, y_i)\})$
$m_v \| r_v \leftarrow \mathrm{G}(\beta) \oplus e_v$
If $(\mathrm{H}(m_v \| r_v) \neq \alpha_v)$ then $m_v \leftarrow \perp$
Require $m_v \notin \mathsf{ChlMsg}$
$\mathsf{fresh} \leftarrow (m_v \notin \mathsf{EncMsg})$
If $m_v \neq \perp$ and fresh then
  $\mathsf{DecSet} \leftarrow \mathsf{DecSet} \cup \{m_v\}$
Return m_v

Figure 13: Combine decryption oracle for $\mathbf{G}_4$. The difference from $\textsc{Combine}_{\mathsf{dec}}$ is highlighted in red.

---

without repetition. We can see that

$$\mathbf{P}\left[\mathbf{G}_2\right] - \mathbf{P}\left[\mathbf{G}_3\right] \leq \frac{q_{\mathsf{enc}}^2}{2 \cdot 2^k} .$$

Note that in $\mathbf{G}_3$, every ciphertext $c_u = (\mathsf{id}_u, \alpha_u, \sigma, e)$ produced by $\textsc{Combine}_{\mathsf{enc}}$ is unique because a different α_u is generated every time in $\textsc{Split}_{\mathsf{enc}}$.

New combine oracle for decryption. The next step is to change the behavior of $\textsc{Combine}_{\mathsf{dec}}$. In the new game, $\mathbf{G}_4$, we have a new oracle $\textsc{Combine}^{\star}_{\mathsf{dec}}$, described formally in Figure 13. The behavior of $\textsc{Combine}^{\star}_{\mathsf{dec}}$ is different from $\textsc{Combine}_{\mathsf{dec}}$ when $(j_v \notin \mathcal{CR})$ and $(j_v, \alpha_v, \sigma^{\star}, e^{\star}) \notin \mathsf{ChlCtxt} \cup \mathsf{EncCtxt}$ for any $\sigma^{\star}, e^{\star}$. In this case, $\textsc{Combine}^{\star}_{\mathsf{dec}}$ immediately returns $\perp$. In particular, no attempt at decryption is made and nothing is added to DecSet. On the other hand, $\textsc{Combine}_{\mathsf{dec}}$ will attempt to decrypt the ciphertext, which can succeed or fail. If the attempt fails, then note that nothing is added to DecSet and $\perp$ is returned.

There are only two oracles that generate (partial) signatures, $\textsc{Combine}_{\mathsf{enc}}$ and $\textsc{Eval}_{\mathsf{enc}}$. $\textsc{Eval}_{\mathsf{enc}}$ actually generates partial signatures only on $\mathsf{id} \| \alpha$ for $\mathsf{id} \in \mathcal{CR}$. So the only way to get any type of signature on $j \| \alpha$ for $j \notin \mathcal{CR}$ and some α is through $\textsc{Combine}_{\mathsf{enc}}$. Moreover, $\textsc{Combine}_{\mathsf{enc}}$ only returns a full signature. Therefore, if $\textsc{Combine}_{\mathsf{dec}}$ is called with a ciphertext $c_v = (j_v, \alpha_v, \sigma_v, e_v)$ where $j_v \notin \mathcal{CR}$, either σ_v is not a valid signature on $j_v \| \alpha_v$, or some signature $\sigma^{\star}$ on it must have been returned by $\textsc{Combine}_{\mathsf{enc}}$ as part of some ciphertext. Thus a ciphertext of the form $(j_v, \alpha_v, \sigma^{\star}, e^{\star})$ for some $e^{\star}$ must be in either $\mathsf{EncCtxt}$ or $\mathsf{ChlCtxt}$.

In other words, if $(j_v, \alpha_v, \sigma^{\star}, e^{\star}) \notin \mathsf{ChlCtxt} \cup \mathsf{EncCtxt}$ for any choice of $\sigma^{\star}, e^{\star}$, then either $\sigma^{\star}$ is not a valid signature (in which case $\textsc{Combine}_{\mathsf{dec}}$'s attempt at decryption will fail) or adversary has managed to forge a signature. Thus, we can construct an attacker $\mathcal{B}_0$ such that

$$\mathbf{P}\left[\mathbf{G}_3\right] - \mathbf{P}\left[\mathbf{G}_4\right] \leq \mathbf{Adv}^{\mathrm{sig}}_{\mathsf{SIG},n,t}(\mathcal{B}_0) .$$

Proc $\textsc{Combine}^{\star\star}_{\mathsf{dec}}(v, \mathsf{rsp})$

If $(j_v \notin \mathcal{CR})$
  If $((j_v, \alpha_v, \sigma^\star, e^\star) \notin \mathsf{EncCtxt} \cup \mathsf{ChlCtxt}$ for any $\sigma^\star, e^\star)$
    Return $\perp$
  Else If $((j_v, \alpha_v, \sigma^\star, e^\star) \in \mathsf{ChlCtxt}$ for some $\sigma^\star, e^\star)$
    Return $\perp$
For $(i, x) \in L_v$ with $i \notin \mathcal{CR}$ do
  If $\mathsf{SIG.VerSig}(\mathsf{vk}, j_v \| \alpha_v, \sigma_v)$ then
    $y_i \leftarrow \mathsf{DP.Eval}(\mathsf{sk}_{\mathsf{DP},i}, j_v \| \alpha_v)$
  Else $y_i \leftarrow \perp$
For $(i, x) \in L_v$ with $i \in \mathcal{CR}$ do
  $y_i \leftarrow \mathsf{rsp}[(i, x)]$
$\beta \leftarrow \mathsf{DP.Combine}(\{(i, y_i)\})$
$m_v \| r_v \leftarrow \mathrm{G}(\beta) \oplus e_v$
If $(\mathrm{H}(m_v \| r_v) \neq \alpha_v)$ then $m_v \leftarrow \perp$
Require $m_v \notin \mathsf{ChlMsg}$
$\mathsf{fresh} \leftarrow (m_v \notin \mathsf{EncMsg})$
If $m_v \neq \perp$ and fresh then
  $\mathsf{DecSet} \leftarrow \mathsf{DecSet} \cup \{m_v\}$
Return m_v

Figure 14: Combine decryption oracle for $\mathbf{G}_5$.

Another change to combine. We define another game $\mathbf{G}_5$ that modifies $\textsc{Combine}^{\star}_{\mathsf{dec}}$ further. We call the new oracle $\textsc{Combine}^{\star\star}_{\mathsf{dec}}$ and describe it formally in Figure 14. The only difference from $\textsc{Combine}^{\star}_{\mathsf{dec}}$ is the addition of a new abort condition: If $(j_v, \alpha_v, \sigma^\star, e^\star) \in \mathsf{ChlCtxt}$ for some $\sigma^\star, e^\star$, then $\textsc{Combine}^{\star\star}_{\mathsf{dec}}$ immediately returns $\perp$. The new condition is added under $j_v \notin \mathcal{CR}$.

Let us consider what happens when $\textsc{Combine}^{\star}_{\mathsf{dec}}$ (from previous game) tries to decrypt $(j_v, \alpha_v, \sigma_v, e_v)$ such that $(j_v, \alpha_v, \sigma^\star, e^\star) \in \mathsf{ChlCtxt}$ for some $\sigma^\star, e^\star$. Either the decryption succeeds or fails. If it fails, then nothing is added to DecSet and $\perp$ is returned (the same effect as in $\textsc{Combine}^{\star\star}_{\mathsf{dec}}$). If the decryption succeeds, then a message m_v is recovered.

Note that the condition $(j_v, \alpha_v, \sigma^\star, e^\star) \in \mathsf{ChlCtxt}$ could be satisfied for at most one $\sigma^\star, e^\star$ because α is unique for every ciphertext added to $\mathsf{ChlCtxt}$. Let $m^\star$ be the message that was encrypted to $(j_v, \alpha_v, \sigma^\star, e^\star)$ in $\textsc{Combine}_{\mathsf{enc}}$ (thus, $m^\star \in \mathsf{ChlMsg}$). Either m_v (from previous paragraph) is same as $m^\star$ or it isn't. If $m_v = m^\star$ then $m_v \in \mathsf{ChlMsg}$, which forces $\textsc{Combine}^{\star}_{\mathsf{dec}}$ to abort. If $m_v \neq m^\star$ then $\mathrm{H}(m_v \| r_v) = \alpha_v$ (because decryption succeeded) and $\alpha_v = \mathrm{H}(m^\star \| r)$ for some r (this is how α_v is derived in $\textsc{Split}_{\mathsf{enc}}$). Thus $\mathrm{H}(m_v \| r_v) = \mathrm{H}(m^\star \| r)$ for $m_v \neq m^\star$. This, however, is not possible because H maps distinct inputs in $\textsc{Split}_{\mathsf{enc}}$ and $\textsc{Combine}^{\star}_{\mathsf{dec}}$ to distinct outputs.

In a nutshell, we can see that adding the new abort condition in $\textsc{Combine}^{\star\star}_{\mathsf{dec}}$ does not matter. Put simply,

$$\mathbf{P}\big[\mathbf{G}_4\big] - \mathbf{P}\big[\mathbf{G}_5\big] = 0\,.$$

There are a few things to note about the newest version of combine decryption, $\textsc{Combine}^{\star\star}_{\mathsf{dec}}$. It starts the decryption process only under two cases: either $j_v \in \mathcal{CR}$ or $(j_v, \alpha_v, \sigma^\star, e^\star) \in \mathsf{EncCtxt}$ for some $\sigma^\star, e^\star$. In the latter case, we now argue that nothing is added to DecSet. Clearly, if the decryption fails, then nothing is added. On the other hand, if a valid message m_v is recovered,

then it could be same as $m^\star$ or not (where $m^\star$ is defined the same way as above). We know that $m_v \neq m^\star$ is not possible and, if they are equal, $m_v \in \mathsf{EncMsg}$. In the latter case, fresh is set to FALSE so, once again, nothing is added to DecSet. To summarize, DecSet can only be modified if $j_v \in \mathcal{CR}$.

Removing forgery check. In the next game, $\mathbf{G}_6$, we get rid of the forgery check but the rest remains the same as before (Figure 15). The output of $\mathbf{G}_6$ can differ from $\mathbf{G}_5$ only when the size of DecSet is larger than $\lfloor \mathsf{ct}_{\mathsf{Eval}}/g \rfloor$, where $g := t - |\mathcal{CR}|$. Let us consider what happens when this condition is satisfied in the previous game, $\mathbf{G}_5$.

Game $\mathbf{G}_5(\mathcal{A})$

$b \leftarrow\!\!\$\ \{0,1\}$
$(\llbracket \mathsf{sk}_{\mathsf{DP}} \rrbracket_n, \mathsf{pp}_{\mathsf{DP}}) \leftarrow\!\!\$\ \mathsf{DP.Setup}(n,t)$
$(\llbracket \mathsf{sk}_{\mathsf{SIG}} \rrbracket_n, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{SIG.Setup}(n,t)$
For $i \in [n]$ do $\mathsf{sk}_i \leftarrow (\mathsf{sk}_{\mathsf{DP},i}, \mathsf{sk}_{\mathsf{SIG},i})$
$\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{DP}}, \mathsf{vk})$
$(\mathcal{CR}, \mathsf{st}_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}_0(\mathsf{pp})$
$b' \leftarrow\!\!\$\ \mathcal{A}_1^{\langle \mathbf{Proc} \rangle}(\mathsf{st}_{\mathcal{A}}, (\mathsf{sk}_i)_{i \in \mathcal{CR}})$
~~If $|\mathsf{DecSet}| > \left\lfloor \frac{\mathsf{ct}_{\mathsf{Eval}}}{t - |\mathcal{CR}|} \right\rfloor$ then forgery $\leftarrow$ TRUE~~
If $(\exists c \in \mathsf{ChlCtxt} : \mathsf{Eval\text{-}MSet}(c, \mathcal{CR}) \subseteq Q_{\mathsf{dec}})$
  Return $(b'' \leftarrow\!\!\$\ \{0,1\})$ ~~$\vee$forgery~~
Return $(b = b')$ ~~$\vee$forgery~~

Figure 15: Forgery check removed

Previously, we argued that the size of DecSet can only increase if $j_v \in \mathcal{CR}$. If the size of DecSet is d after the adversary is done querying the oracles (i.e., after $\mathcal{A}_1$ outputs b'), then it must be that d *distinct* messages were added to DecSet. Thus, at least d ciphertexts with $j_v \in \mathcal{CR}$ must have been decrypted successfully, each with a different α_v. (If two ciphertexts have the same α_v, they cannot decrypt to two valid messages.) Therefore, signature verification on at least d distinct values of the form $j_v \| \alpha_v$ must have succeeded.

Now observe that only the oracle $\textsc{Eval}_{\mathsf{enc}}$ generates signatures on $\mathsf{id} \| \alpha$ for $\mathsf{id} \in \mathcal{CR}$. A single call returns just one partial signature and at least g partial signatures are needed to produce a full signature. Therefore, if the oracle has been called $\mathsf{ct}_{\mathsf{Eval}}$ times, signatures on at most $\lfloor \mathsf{ct}_{\mathsf{Eval}}/g \rfloor$ values can be produced. So, if $d > \lfloor \mathsf{ct}_{\mathsf{Eval}}/g \rfloor$, adversary was able to forge a signature successfully. We can use this adversary to build an attacker $\mathcal{B}_1$ such that

$$\mathbf{P}\big[\mathbf{G}_5\big] - \mathbf{P}\big[\mathbf{G}_6\big] \leq \mathbf{Adv}^{\mathrm{sig}}_{\mathsf{SIG},n,t}(\mathcal{B}_1) \ .$$

Declining evaluation & encryption. The next step is to modify $\textsc{Eval}_{\mathsf{dec}}$ and $\textsc{Combine}_{\mathsf{enc}}$ for a new game $\mathbf{G}_7$. The new oracle $\textsc{Eval}^\star_{\mathsf{dec}}$ adds an extra check right after adding x to Q_{dec}: if the query $x = (j \| \alpha, \sigma)$ belongs to a ciphertext in $\mathsf{ChlCtxt}$ and has been queried $g-1$ times or more before, then return $\perp$. The new oracle $\textsc{Combine}^\star_{\mathsf{enc}}$ adds an extra check right after σ is computed: if $m_{u,0} \neq m_{u,1}$ and $(\mathsf{id}_u \| \alpha_u, \sigma)$ has already been queried to $\textsc{Eval}^\star_{\mathsf{dec}}$ g times or more, then return $\perp$. (Also, $\textsc{Combine}^\star_{\mathsf{enc}}$ defers the computation of β till after this check is complete.)

Let us define an event E. We say that E happens if $\exists c \in \mathsf{ChlCtxt}$ s.t. $\mathsf{Eval\text{-}MSet}(c, \mathcal{CR}) \subseteq Q_{\mathsf{dec}}$ (see Figure 15, after $\mathcal{A}$ outputs b'). We want to show that the games $\mathbf{G}_6$ and $\mathbf{G}_7$ have the same

output distribution. We break this down into two cases, depending on whether E happens or not. If E does happen, then both games just output a random bit (so their output distribution is clearly the same). If E does not happen, then the output of the games is determined by the output of $\mathcal{A}$.

We will argue that the probability that E happens is the same in both the games. Moreover, if E does not happen, the output distribution of both the games is the same.

Fix a random tape for the adversary $\mathcal{A}$. Also fix the random choices made in $\mathbf{G}_6$ and $\mathbf{G}_7$. In fact, consider the *same* random choices for both the games. Suppose E happens in $\mathbf{G}_6$. This implies that for some $c = (j, \alpha, \sigma, e) \in \mathsf{ChlCtxt}$, $\mathsf{Eval\text{-}MSet}(c, \mathcal{CR}) \subseteq Q_{\mathsf{dec}}$. Recall that for a ciphertext $c' = (j', \alpha', \sigma', e')$, the multiset $\mathsf{Eval\text{-}MSet}(c', \mathcal{CR})$ just contains $(j' \| \alpha', \sigma')$, but repeated g times. Also note that in $\mathbf{G}_6$, the multiset Q_{dec} is modified only in the oracle $\textsc{Eval}_{\mathsf{dec}}$; whenever $(j \| \alpha, \sigma)$ is queried to $\textsc{Eval}_{\mathsf{dec}}$, it is added to Q_{dec}. (For game $\mathbf{G}_7$, just replace $\textsc{Eval}_{\mathsf{dec}}$ with $\textsc{Eval}^{\star}_{\mathsf{dec}}$.) Therefore, if $\mathsf{Eval\text{-}MSet}(c, \mathcal{CR}) \subseteq Q_{\mathsf{dec}}$, then $(j \| \alpha, \sigma)$ must have been input to $\textsc{Eval}_{\mathsf{dec}}$ (or $\textsc{Eval}^{\star}_{\mathsf{dec}}$) at least g times (and vice versa). The g-th $(j \| \alpha, \sigma)$ query could have been made either before or after c is added to $\mathsf{ChlCtxt}$.

Recall that we are under the assumption that E happens in $\mathbf{G}_6$. Let $c' = (j', \alpha', \sigma', e') \in \mathsf{ChlCtxt}$ be the first ciphertext for which a g-th query, on $(j' \| \alpha', \sigma')$, is made to $\textsc{Eval}_{\mathsf{dec}}$. Let Q be this query. (Note that c' may not have been added to $\mathsf{ChlCtxt}$ before Q.) Before Q is made, for all ciphertexts that will eventually be added to $\mathsf{ChlCtxt}$, $\mathcal{A}$ has not yet made the corresponding g-th query to $\textsc{Eval}_{\mathsf{dec}}$. Let us focus on the period before Q in the new game $\mathbf{G}_7$. We can easily see that the new condition added to $\textsc{Eval}^{\star}_{\mathsf{dec}}$ will not be true in this period. On the other hand, the new condition added to $\textsc{Combine}^{\star}_{\mathsf{enc}}$ could be true when some query Q' is made. However, when the same query is made to $\textsc{Combine}_{\mathsf{enc}}$, it must not recover a valid ciphertext (else for this ciphertext, at least g queries have already been made). Thus, the output of $\textsc{Combine}^{\star}_{\mathsf{enc}}$ and $\textsc{Combine}_{\mathsf{enc}}$ is the same for any query made before Q.

To sum up, up to the point Q, $\mathcal{A}$ has the same view in both the games. So if it makes the Q-th query in $\mathbf{G}_6$, it will also make the same query in $\mathbf{G}_7$. In other words, if E happens in $\mathbf{G}_6$, it will also happen in $\mathbf{G}_7$.

We can extend this argument to the case when E does *not* happen in $\mathbf{G}_6$. A query of type Q will never be made in this case, so $\mathcal{A}$'s view until the end of the game will be same in both the games. E will not happen in $\mathbf{G}_7$ either and $\mathcal{A}$ will output the same bit. To conclude,

$$\mathbf{P}\big[\mathbf{G}_6\big] - \mathbf{P}\big[\mathbf{G}_7\big] = 0\,.$$

Pseudorandomness of DPRF. For the next game, $\mathbf{G}_8$, we change the oracle $\textsc{Combine}^{\star}_{\mathsf{enc}}$. When $m_{u,0} \neq m_{u,1}$, $\mathsf{DP.Eval}$ and $\mathsf{DP.Combine}$ are not used to compute β. Instead, a random value of the same length is used. We call the new oracle $\textsc{Combine}^{\star\star}_{\mathsf{enc}}$.

We can show that if the DPRF DP is pseudorandom, then $\mathbf{G}_7$ and $\mathbf{G}_8$ are indistinguishable. In other words, we can construct an attacker $\mathcal{C}$ s.t.

$$\mathbf{P}\big[\mathbf{G}_7\big] - \mathbf{P}\big[\mathbf{G}_8\big] \leq \mathbf{Adv}^{\mathrm{dprf}}_{\mathsf{DP},n,t}(\mathcal{C})\,.$$

$\mathcal{C}$ will take part in the DPRF security game (Figure 5), and try to simulate games $\mathbf{G}_7$ and $\mathbf{G}_8$ for $\mathcal{A}$. In the DPRF game, $\mathcal{C}$ has access to oracles $\textsc{Eval}$ and $\textsc{Chl}$. For clarity in the text below, we will refer to them as $\mathsf{DP.}\textsc{Eval}$ and $\mathsf{DP.}\textsc{Chl}$, respectively.

Let us first see where and how $\mathsf{DP.Eval}$ is used in the two games, and compare that with its use in the two oracles, $\textsc{Combine}^{\star}_{\mathsf{enc}}$ and $\textsc{Combine}^{\star\star}_{\mathsf{enc}}$. $\textsc{Eval}_{\mathsf{enc}}$ uses $\mathsf{DP.Eval}$ but only on $\mathsf{id} \| \alpha$ s.t. $\mathsf{id} \in \mathcal{CR}$. $\textsc{Eval}^{\star}_{\mathsf{dec}}$ uses $\mathsf{DP.Eval}$ too but for a ciphertext $c = (j, \alpha, \sigma, e) \in \mathsf{ChlCtxt}$, it returns $\perp$ for the g-th onwards call on $(j \| \alpha, \sigma)$. $\textsc{Combine}^{\star\star}_{\mathsf{dec}}$ also uses $\mathsf{DP.Eval}$ but only when $j_v \in \mathcal{CR}$ or $(j_v, \alpha_v, \sigma^{\star}, e^{\star}) \in \mathsf{EncCtxt}$ for some $\sigma^{\star}, e^{\star}$.

Now we are ready to discuss $\mathcal{C}$'s strategy. In the simulation of combine encryption oracle, when $m_{u,0} \neq m_{u,1}$, it would not try to evaluate DP. Instead, it would query $\mathsf{DP}.\textsc{Chl}$ with inputs $\mathsf{id}_u \| \alpha_u, S'$ and rsp, where $\mathsf{id}_u \notin \mathcal{CR}$, and use the value returned in place of β. In the simulation of other oracles, if $\mathsf{DP.Eval}$ is needed for i-th party on input $j \| \alpha$, then $\mathcal{C}$ invokes the oracle $\mathsf{DP}.\textsc{Eval}$ on inputs i and $j \| \alpha$. At the end, $\mathcal{C}$ outputs whatever $\mathcal{A}$ does.

This strategy can work only if $\mathsf{id}_u \| \alpha_u$ is queried to $\mathsf{DP}.\textsc{Eval}$ strictly less than g times. $\mathcal{C}$'s queries to $\mathsf{DP}.\textsc{Eval}$ in the simulation of $\textsc{Eval}_{\mathsf{enc}}$ and $\textsc{Combine}^{\star\star}_{\mathsf{dec}}$ are not a problem because at least one of j or α is different. However, simulation of $\textsc{Eval}^{\star}_{\mathsf{dec}}$ may require invoking $\mathsf{DP}.\textsc{Eval}$ with $\mathsf{id}_u \| \alpha_u$. Nonetheless, two important things must be noted here. First, for $\mathcal{C}$ to even get to the stage of invoking $\mathsf{DP}.\textsc{Chl}$ in the simulation of combine encryption, $\mathcal{A}$ must have queried $\textsc{Eval}^{\star}_{\mathsf{dec}}$ with input $(\mathsf{id}_u \| \alpha_u, \sigma)$ strictly less than g times (else, the combine oracle is supposed to just return $\perp$). Second, after $(\mathsf{id}_u, \alpha_u, \sigma, e)$ is added to $\mathsf{ChlCtxt}$, if subsequent calls on $(\mathsf{id}_u \| \alpha_u, \sigma)$ are made to $\textsc{Eval}^{\star}_{\mathsf{dec}}$ so that the total number of them exceeds $g-1$, then $\mathcal{C}$ does not need to invoke $\mathsf{DP}.\textsc{Eval}$ because $\textsc{Eval}^{\star}_{\mathsf{dec}}$ is supposed to just return $\perp$.

Final steps. In the next game, $\mathbf{G}_9$, e is replaced with a random value in $\textsc{Combine}^{\star\star}_{\mathsf{enc}}$ when $m_{u,0} \neq m_{u,1}$. In the previous game, e is the XOR of $\mathrm{G}(\beta)$ and $(m_{u,b} \| r_u)$, where β is a random value which is not used anywhere else. If G is modeled as a random oracle, then the e values appear completely random to $\mathcal{A}$ as long as it never queries for any of the β values. So, by applying a union bound, we have

$$\mathbf{P}\big[\mathbf{G}_8\big] - \mathbf{P}\big[\mathbf{G}_9\big] \leq \frac{q_{\mathrm{G}} \cdot q_{\mathsf{enc}}}{|\mathsf{DP.Out}|} \ .$$

The final step is to replace α_u with a random value in $\textsc{Split}_{\mathsf{enc}}$. The new game is called $\mathbf{G}_{10}$. In this game, no information about the bit b is available to the adversary. In the previous game, α_u is defined to be $\mathrm{H}(m_{u,b} \| r_u)$, where r_u is random k-bit value which is not used anywhere else. If H is modeled as a random oracle, then the α_u values appear completely random to $\mathcal{A}$ as long as it never queries for any of the r_u values. Thus, we have

$$\mathbf{P}\big[\mathbf{G}_9\big] - \mathbf{P}\big[\mathbf{G}_{10}\big] \leq \frac{q_{\mathrm{H}} \cdot q_{\mathsf{enc}}}{2^k} \ .$$

6.3 IND-CCA construction from DVRF & threshold signatures

In this section, we construct an IND-CCA secure TAE scheme. We start with the construction in Section 6 which was based on DPRF and threshold signature. We replace the DPRF in $\mathsf{TAE2}$ with a DVRF. The new construction, $\mathsf{TAE3}$, can actually be shown to be IND-CCA secure. The DVRF, in turn, can be instantiated in several ways. We discussed one instantiation, that adds efficient zero-knowledge proofs to the Naor et al. [NPR99] DDH-based DPRF construction similar to AMMR, in Appendix 6.1.4.

We present $\mathsf{TAE3}$ formally in Figure 16. The construction is the same as $\mathsf{TAE2}$ but DPRF is replaced with a DVRF. The important thing to note is that both $\mathsf{Combine}_{\mathsf{enc}}$ and $\mathsf{Combine}_{\mathsf{dec}}$ verify the PRF shares before combining them. If verification fails, then they return $\perp$. The security is formalized as follows.

Theorem 7. *Let* $\mathsf{TAE} = \mathsf{TAE3}[\mathsf{DV}, \mathsf{SIG}, k]$. *Let* $\mathcal{A}$ *be an adversary in the IND-CCA game against* TAE *(Figure 2). Suppose* $\mathcal{A}$ *makes* q_{H} *and* q_{G} *queries to oracles* H *and* G*, respectively. Further, it makes* q_{enc}*,* q_{Eval} *and* q_{dec} *queries to encryption* ($\textsc{Split}_{\mathsf{enc}}$, $\textsc{Combine}_{\mathsf{enc}}$)*, evaluation* ($\textsc{Eval}_{\mathsf{enc}}$, $\textsc{Eval}_{\mathsf{dec}}$) *and decryption procedures* ($\textsc{Split}_{\mathsf{dec}}$, $\textsc{Combine}_{\mathsf{dec}}$)*, respectively. Then there exist adver-*

saries $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ such that

$$\mathbf{Adv}^{\text{ind-rcca}}_{\mathsf{TAE},n,t}(\mathcal{A}) \le \frac{(q_{\mathsf{enc}}+q_{\mathsf{dec}})^2}{|\mathsf{DV.In}|} + \frac{q_{\mathsf{enc}}^2 + 2\cdot q_{\mathrm{H}}\cdot q_{\mathsf{enc}}}{2^k} + \frac{2\cdot q_{\mathrm{G}}\cdot q_{\mathsf{enc}}}{|\mathsf{DV.Out}|} + 2\cdot \mathbf{Adv}^{\text{sig}}_{\mathsf{SIG},n,t}(\mathcal{B}) + 2\cdot \mathbf{Adv}^{\text{dvrf}}_{(}(\mathcal{C})) + 4\cdot \mathbf{Adv}^{\text{dvrf-rob}}_{\mathsf{DP},n,t}(\mathcal{D})\ . \tag{26}$$

We can prove the theorem in a manner similar to Theorem 6. We need a stronger security guarantee now though. IND-RCCA allows ciphertexts to be modified as long as the underlying message remains the same but IND-CCA does not allow that. However, we have a DVRF now.

Scheme TAE3[DV, SIG, k]

Proc $\mathsf{Split}_{\mathsf{enc}}(j, m, S)$
Require $|S| \ge t$
Let S' be a t-sized subset of S
$r \leftarrow \{0,1\}^k$; $\alpha \leftarrow \mathrm{H}(m\|r)$
$\mathsf{st} \leftarrow (j, \alpha, m, r)$
Return $(\{(i,\alpha)\}_{i\in S'}, \mathsf{st})$

Proc $\mathsf{Eval}_{\mathsf{enc}}(\mathsf{sk}_i, j, \alpha)$
$(\mathsf{sk}_{\mathsf{DV},i}, \mathsf{sk}_{\mathsf{SIG},i}) \leftarrow \mathsf{sk}_i$
$y \leftarrow \mathsf{DV.Eval}(\mathsf{sk}_{\mathsf{DV},i}, j\|\alpha)$
$\sigma \leftarrow \mathsf{SIG.PartSign}(\mathsf{sk}_{\mathsf{SIG},i}, j\|\alpha)$
Return (y, σ)

Proc $\mathsf{Combine}_{\mathsf{enc}}(R, \mathsf{st})$
$(j, \alpha, m, r) \leftarrow \mathsf{st}$; $\{(i,(y_i,\sigma_i))\} \leftarrow R$
If $\mathsf{DV.Verify}(j\|\alpha, \{(i,y_i)\})$ then
  $\beta \leftarrow \mathsf{DV.Combine}(\{(i,y_i)\})$
Else return $\perp$
$\sigma \leftarrow \mathsf{SIG.CombSig}(\mathsf{vk}, \{(i,\sigma_i)\})$
If $\mathsf{VerSig}(\mathsf{vk}, j\|\alpha, \sigma) \ne 1$ then return $\perp$
$e \leftarrow \mathrm{G}(\beta) \oplus (m\|r)$
Return (j, α, σ, e)

Proc $\mathsf{Setup}(n, t)$
$(\llbracket \mathsf{sk}_{\mathsf{DV}} \rrbracket_n, \mathsf{pp}_{\mathsf{DV}}) \leftarrow\!\!\$\ \mathsf{DV.Setup}(n, t)$
$(\llbracket \mathsf{sk}_{\mathsf{SIG}} \rrbracket_n, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{SIG.Setup}(n, t)$
For $i \in [n]$ do $\mathsf{sk}_i \leftarrow (\mathsf{sk}_{\mathsf{DV},i}, \mathsf{sk}_{\mathsf{SIG},i})$
$\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{DV}}, \mathsf{vk})$
Return $((\mathsf{sk}_1, \ldots, \mathsf{sk}_n), \mathsf{pp})$

Proc $\mathsf{Split}_{\mathsf{dec}}(j, c, S)$
Require $|S| \ge t$
Let S' be a t-sized subset of S
$(j, \alpha, \sigma, e) \leftarrow c$; $\mathsf{st} \leftarrow (j, \alpha, e)$
Return $(\{(i,(j\|\alpha,\sigma))\}_{i\in S'}, \mathsf{st})$

Proc $\mathsf{Eval}_{\mathsf{dec}}(\mathsf{sk}_i, j, x)$
$(\mathsf{pp}_{\mathsf{DV}}, \mathsf{vk}) \leftarrow \mathsf{pp}$; $(j\|\alpha, \sigma) \leftarrow x$
$(\mathsf{sk}_{\mathsf{DV},i}, \mathsf{sk}_{\mathsf{SIG},i}) \leftarrow \mathsf{sk}_i$
If $\mathsf{SIG.VerSig}(\mathsf{vk}, j\|\alpha, \sigma)$ then
  Return $\mathsf{DV.Eval}(\mathsf{sk}_{\mathsf{DV},i}, j\|\alpha)$
Else return $\perp$

Proc $\mathsf{Combine}_{\mathsf{dec}}(R, \mathsf{st})$
$(j, \alpha, e) \leftarrow \mathsf{st}$
If $\mathsf{DV.Verify}(j\|\alpha, R)$ then
  $\beta \leftarrow \mathsf{DV.Combine}(R)$
Else return $\perp$
$m\|r \leftarrow \mathrm{G}(\beta) \oplus e$
If $(\mathrm{H}(m\|r) \ne \alpha)$ then return $\perp$
Return m

Figure 16: DVRF & threshold signature based TAE scheme.

6.4 Proof of Theorem 7

We will prove Theorem 7 in a manner similar to Theorem 6. Instead of the weaker RCCA security though, we have to prove CCA security now (both are defined in Figure 2). Fix an adversary $\mathcal{A}$ in the CCA security game.

We will go through a series of games. Proof of Theorem 6 in Appendix **??** (henceforth referred to as RCCA proof) defines games $\mathbf{G}_1$ through $\mathbf{G}_{10}$. We will consider similar games here but will denote them with $\mathbf{G}'_i$ for clarity.

Games 1-3. In RCCA proof, $\mathbf{G}_1$ is just the RCCA TAE game instantiated with TAE2. Here, $\mathbf{G}'_1$ will be the instantiation of the CCA version of TAE game with TAE3. We can then define $\mathbf{G}'_2$ and $\mathbf{G}'_3$ in a manner similar to RCCA proof and show that the security loss in going from $\mathbf{G}'_1$ to $\mathbf{G}'_3$ will be the same.

New combine oracle for decryption. Recall that $\mathbf{G}_4$ defines the oracle $\textsc{Combine}^{\star}_{\mathsf{dec}}$ which checks if $(j_v \notin \mathcal{CR})$ and $(j_v, \alpha_v, \sigma^{\star}, e^{\star}) \notin \mathsf{ChlCtxt} \cup \mathsf{EncCtxt}$ for any $\sigma^{\star}, e^{\star}$. If so, it immediately returns $\perp$. Indistinguishability of $\mathbf{G}_3$ and $\mathbf{G}_4$ in RCCA proof follows from the unforgeability of the signature scheme: σ_v must be a valid signature for decryption to succeed but such signatures for $j_v \notin \mathcal{CR}$ can only come from $\textsc{Combine}_{\mathsf{enc}}$.

We can define a stronger form of $\mathbf{G}_4$ here, called $\mathbf{G}'_4$. $\textsc{Combine}^{\star}_{\mathsf{dec}}$ will check if $(j_v \notin \mathcal{CR})$ and $(j_v, \alpha_v, \sigma_v, e^{\star}) \notin \mathsf{ChlCtxt} \cup \mathsf{EncCtxt}$ for any $e^{\star}$. In simple words, we require that if $j_v \notin \mathcal{CR}$, then the first three components of the ciphertext (not just two) must match with some ciphertext in $\mathsf{ChlCtxt}$ or $\mathsf{EncCtxt}$, otherwise $\textsc{Combine}^{\star}_{\mathsf{dec}}$ will abort immediately. This does not have any effect on the analysis though if SIG produces unique signatures.

Verifiability of DVRF. We will use the verifiability of DVRF to define a new intermediate game here, a game that was not present in RCCA proof. Thus, we will call the new game $\mathbf{G}'_{4.5}$. In this game, we will again modify the combine decryption oracle. The new oracle is called $\textsc{Combine}'_{\mathsf{dec}}$. In $\textsc{Combine}^{\star}_{\mathsf{dec}}$ from the previous game, if $j_v \notin \mathcal{CR}$, then $(j_v, \alpha_v, \sigma_v, e^{\star})$ must be in $\mathsf{ChlCtxt}$ or $\mathsf{EncCtxt}$ for some $e^{\star}$ (otherwise it aborts). We know that there could be at most one ciphertext c in $\mathsf{ChlCtxt} \cup \mathsf{EncCtxt}$ with which the first three components of c_v match (because α is unique for every ciphertext in $\textsc{Combine}_{\mathsf{enc}}$). Let β be the DVRF value generated in $\textsc{Combine}_{\mathsf{enc}}$ for c. (This value may not be the true DPRF value though because adversary can provide malformed partial values.) In $\textsc{Combine}'_{\mathsf{dec}}$, when $j_v \notin \mathcal{CR}$, instead of first verifying the y_i values and then computing a new β, say β_v, we compute β_v directly. If $\beta_v \neq \beta$, then $\textsc{Combine}'_{\mathsf{dec}}$ returns $\perp$.

We can show the advantage of $\mathcal{A}$ in distinguishing $\mathbf{G}'_4$ and $\mathbf{G}'_{4.5}$ can be bounded by twice the advantage of an adversary $\mathcal{D}$ in the DVRF robustness game (Figure 6b). The idea is simple: Consider the ciphertext $c_v = (j_v, \alpha_v, \sigma_v, e^{\star})$ from above. Suppose it is successfully decrypted by $\textsc{Combine}^{\star}_{\mathsf{dec}}$ but $\textsc{Combine}'_{\mathsf{dec}}$ returns $\perp$ because of the new condition. Then DVRF verification must succeed in $\textsc{Combine}^{\star}_{\mathsf{dec}}$ but β_v recovered is different from β. Note that DVRF verification for β has already succeeded in $\textsc{Combine}_{\mathsf{enc}}$. Now, if verification succeeds for two different DPRF values on the same $j_v \| \alpha_v$, one of them must not be correct.

Thus, the adversary $\mathcal{D}$ looks for a $\beta_v \neq \beta$ in the simulation of combine decryption for which verification succeeds. When it finds such a pair, it outputs one of β_v or β at random. We have that

$$\mathbf{P}\left[\mathbf{G}'_4\right] - \mathbf{P}\left[\mathbf{G}'_{4.5}\right] \leq 2 \cdot \mathbf{Adv}^{\mathrm{dvrf-rob}}_{\mathsf{DP},n,t}(\mathcal{D}) \; .$$

Another change to combine. In RCCA proof, $\mathbf{G}_5$ defines another combine decryption oracle, $\textsc{Combine}^{\star\star}_{\mathsf{dec}}$. If $(j_v, \alpha_v, \sigma^{\star}, e^{\star}) \in \mathsf{ChlCtxt}$ for some $\sigma^{\star}, e^{\star}$, then $\textsc{Combine}^{\star\star}_{\mathsf{dec}}$ immediately returns $\perp$. $\textsc{Combine}^{\star\star}_{\mathsf{dec}}$ for $\mathbf{G}'_5$ will be slightly different as expected. It returns $\perp$ if $(j_v, \alpha_v, \sigma_v, e^{\star}) \in \mathsf{ChlCtxt}$ for some $e^{\star}$.

We know that $c^{\star} = (j_v, \alpha_v, \sigma_v, e^{\star}) \in \mathsf{ChlCtxt}$ could be satisfied for at most one $e^{\star}$. Let $m^{\star}$ be the message that was encrypted to $c^{\star}$ in $\textsc{Combine}_{\mathsf{enc}}$ and $\beta^{\star}$ be the recovered DPRF value. In RCCA proof, we were able to argue that $m_v \neq m^{\star}$ is not possible (outputs of H do not collide) and $m_v = m^{\star}$ is not a problem (because $m_v \in \mathsf{ChlMsg}$ is required).

Here, the situation is different because we want CCA security. If $c^\star$ successfully decrypts, then it will be added to DecSet. However, we know that e_v cannot be equal to $e^\star$, else $\textsc{Split}_{\mathsf{dec}}$ itself would have aborted. When decryption of $c^\star$ starts, $\textsc{Combine}'_{\mathsf{dec}}$ would only proceed if $\beta = \beta^\star$ (see the previous paragraph for $\beta^\star$). If $\beta = \beta^\star$ but $e_v \neq e^\star$, an m_v different from $m^\star$ is recovered, which is not possible. Thus, we have:

$$\mathbf{P}\left[\mathbf{G}'_{4.5}\right] - \mathbf{P}\left[\mathbf{G}'_{5}\right] = 0 \,.$$

As defined, $\textsc{Combine}^{\star\star}_{\mathsf{dec}}$ starts the decryption process only under two cases: either $j_v \in \mathcal{CR}$ or $(j_v, \alpha_v, \sigma_v, e^\star) \in \mathsf{EncCtxt}$ for some $e^\star$. As in RCCA proof, we can argue that in the latter case, nothing is added to DecSet but the argument here would be different. We know that $e^\star$ is unique and if $e_v = e^\star$, then $c_v = c^\star$. As a result, fresh will be set of $\textsc{False}$ and DecSet will not be affected. On the other hand, when $e_v \neq e^\star$, we can argue that decryption will fail (in the same manner as above).

Remaining games. The remaining games in RCCA proof can also be defined here (with appropriate adjustments) and their analysis would essentially be the same. We will also incur the same security loss when switching between the games. We skip the details and conclude the proof here.

References

[ABL+14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 105–125. Springer, Heidelberg, December 2014.

[ABL+18] David W. Archer, Dan Bogdanov, Yehuda Lindell, Liina Kamm, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P. Smart, and Rebecca N. Wright. From keys to databases - real-world applications of secure multi-party computation. *Comput. J.*, 61(12):1749–1771, 2018.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219. Springer, Heidelberg, December 2016.

[AMMM18] Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. PASTA: PASsword-based threshold authentication. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 2042–2059. ACM Press, October 2018.

[AMMR18a] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed symmetric-key encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1993–2010. ACM Press, October 2018.

[AMMR18b] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed symmetric-key encryption. Cryptology ePrint Archive, Report 2018/727, 2018. https://eprint.iacr.org/2018/727.

[app] Secure Enclave overview - Apple Support. support.apple.com/guide/security/secure-enclave-overview-sec59b0b31ff/1/web/1.

[arm] TrustZone. developer.arm.com/ip-products/security-ip/trustzone.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, April 2015.

[BBKN12] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. On-line ciphers and the hash-CBC constructions. *Journal of Cryptology*, 25(4):640–679, October 2012.

[BC] Jean-Baptiste Bedrune and Gabriel Campana. Everybody be cool, this is a robbery! www.sstic.org/media/SSTIC2019/SSTIC-actes/hsm/SSTIC2019-Article-hsm-campana_bedrune_neNSDyL.pdf.

[BD10] Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 201–218. Springer, Heidelberg, February 2010.

[BF18] Manuel Barbosa and Pooya Farshim. Indifferentiable authenticated encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 187–220. Springer, Heidelberg, August 2018.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Heidelberg, August 2013.

[BMOS17] Guy Barwell, Daniel P. Martin, Elisabeth Oswald, and Martijn Stam. Authenticated encryption in the face of protocol and side channel leakage. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 693–723. Springer, Heidelberg, December 2017.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, Heidelberg, December 2000.

[Boy99] Victor Boyko. On the security properties of OAEP as an all-or-nothing transform. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 503–518. Springer, Heidelberg, August 1999.

[BR95] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 92–111. Springer, Heidelberg, May 1995.

[BT16] Mihir Bellare and Björn Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 247–276. Springer, Heidelberg, August 2016.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 565–582. Springer, Heidelberg, August 2003.

[coi] Coinbase custody. custody.coinbase.com/. Use of secret sharing described in [coi19].

[coi19] [Podcast] Institutional Cryptoasset Custody w/ Sam McIngvale of Coinbase Custody - (Eps. 0028 - 0029). blog.nomics.com/flippening/coinbase-custody-sam-mcingvale/, 07 2019.

[cura] Curv. www.curv.co/. Use of MPC claimed in [curb].

[curb] Curv Technology. www.curv.co/technology/.

[Des88] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Heidelberg, August 1988.

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, August 1990.

[Dod03] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 1–17. Springer, Heidelberg, January 2003.

[DSS01] Yevgeniy Dodis, Amit Sahai, and Adam Smith. On perfect and adaptive security in exposure-resilient cryptography. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 301–324. Springer, Heidelberg, May 2001.

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Heidelberg, August 2013.

[FFL12] Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A family of almost foolproof on-line authenticated encryption schemes. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 196–215. Springer, Heidelberg, March 2012.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324. Springer, Heidelberg, February 2005.

[FJMV04] Pierre-Alain Fouque, Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. Authenticated on-line encryption. In Mitsuru Matsui and Robert J. Zuccherato, editors, *SAC 2003*, volume 3006 of *LNCS*, pages 145–159. Springer, Heidelberg, August 2004.

[GLOW20] David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. Cryptology ePrint Archive, Report 2020/096, 2020. https://eprint.iacr.org/2020/096.

[goo] Titan M makes Pixel 3 our most secure phone yet. www.blog.google/products/pixel/titan-m-makes-pixel-3-our-most-secure-phone-yet/.

[HKR15] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 15–44. Springer, Heidelberg, April 2015.

[JKKX17] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 39–58. Springer, Heidelberg, July 2017.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 577–594. Springer, Heidelberg, March 2009.

[KHF+19] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[KOR+17] Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl, Eduardo Soria-Vazquez, and Srinivas Vivek. Faster secure multi-party computation of AES and DES using lookup tables. In *ACNS-*, 2017.

[KY01] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In Bruce Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 284–299. Springer, Heidelberg, April 2001.

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

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

[MS95] Silvio Micali and Ray Sidney. A simple method for generating and sharing pseudo-random functions, with applications to clipper-like escrow systems. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 185–196. Springer, Heidelberg, August 1995.

[MS18] Steven Myers and Adam Shull. Practical revocation and key rotation. In Nigel P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 157–178. Springer, Heidelberg, April 2018.

[Muk20] Pratyay Mukherjee. Adaptively secure threshold symmetric-key encryption. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 465–487. Springer, Heidelberg, December 2020.

[MV04] David A. McGrew and John Viega. The security and performance of the galois/counter mode (GCM) of operation. In *INDOCRYPT-*, 2004.

[nis] NIST tcg. csrc.nist.gov/Projects/threshold-cryptography.

[NPR99] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 327–346. Springer, Heidelberg, May 1999.

[Riv97] Ronald L. Rivest. All-or-nothing encryption and the package transform. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 210–218. Springer, Heidelberg, January 1997.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 98–107. ACM Press, November 2002.

[Rog04] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 348–359. Springer, Heidelberg, February 2004.

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006.

[RZ11] Phillip Rogaway and Haibin Zhang. Online ciphers from tweakable blockciphers. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 237–249. Springer, Heidelberg, February 2011.

[sep] Unbound Security. sepior.com/threshold-kmaas. Use of MPC and secret sharing mentioned in [ABL+18].

[sgx] Intel Software Guard Extensions. software.intel.com/en-us/sgx.

[SMC08] Joseph A. Salowey, David McGrew, and Abhijit Choudhury. AES Galois Counter Mode (GCM) Cipher Suites for TLS. RFC 5288, August 2008.

[unb] Unbound Tech. www.unboundtech.com/. Use of MPC mentioned in [ABL+18].

[vaua] Vault. www.vaultproject.io/. Use of secret sharing described in [vauc, vaub].

[vaub] Vault Architecture. www.vaultproject.io/docs/internals/architecture.html.

[vauc] Vault Seal. www.vaultproject.io/docs/concepts/seal.html.

Appendix

A Performance Experiments

We implement our protocols TAE1 of Figure 4 and TAE3 of Figure 16 and report on their performance. All performance results are obtained on a single laptop with an Intel i7 9th Gen (9740H) CPU and 16GB of RAM. Network communication was routed over local host with a theoretical bandwidth of 10Gbps and a measured latency of 0.1 milliseconds. Each party is run on a single thread.

t	n	Throughput (enc/s)				Throughput ($Mbps$)				Latency (ms/enc)			
		TAE1	DiSE1	TAE3	DiSE2	TAE1	DiSE1	TAE3	DiSE2	TAE1	DiSE1	TAE3	DiSE2
$n/3$	6	730,627	1,123,555	346	444	88	111	0.3	0.4	0.1	0.1	4.0	3.3
	12	86,588	326,193	145	172	581	97	0.4	0.5	0.4	0.3	7.9	6.7
	18	2,179	13,464	91	105	633	8	0.5	0.5	1.0	0.7	12.1	10.5
$n/2$	4	745,486	1,123,555	346	452	77	111	0.3	0.4	0.1	0.1	4.0	3.3
	6	561,777	722,285	222	259	333	143	0.4	0.5	0.2	0.2	5.6	4.8
	12	24,543	131,324	91	106	988	78	0.5	0.5	0.5	0.5	12.0	10.4
	18	311	3,351	58	68	738	3	0.6	0.5	2.7	1.0	17.9	15.8
$2n/3$	3	777,846	1,123,555	348	445	77	111	0.3	0.4	0.1	0.1	4.0	3.3
	6	421,333	505,600	143	174	500	150	0.4	0.5	0.3	0.3	7.9	6.9
	12	24,845	129,641	65	77	1,400	103	0.6	0.5	0.6	0.6	16.0	14.2
	18	483	6,347	42	49	1,149	8	0.6	0.5	2.3	1.0	24.1	21.4
$n-2$	12	81,548	297,411	51	60	1,637	324	0.6	0.5	0.6	0.6	19.9	17.5
	18	23,905	219,826	30	36	1,983	391	0.6	0.5	1.0	1.0	32.5	28.6
2	12	674,133	1,011,200	337	445	67	100	0.3	0.4	0.2	0.2	4.1	3.4
	18	594,823	919,272	345	441	59	91	0.3	0.4	0.2	0.2	4.1	3.5

Table 1: Encryption performance metrics with various number of parties n and threshold t. Throughput is computed by performing many encryptions concurrently (single thread per party). Mbps denotes network bandwidth. Latency is computed by performing sequential encryptions.

Table 1 contains the results of two experiments. 1) peak encryptions per second each scheme can perform. In particular, 32 byte messages are repeatedly encrypted in an asynchronous manner, where a single party repeatedly initiates 10 batches of 128 encryptions which are processed concurrently. 2) latency of one encryption by running multiple encryptions one at a time in a sequential manner. We report the average time required to perform a single encryption.

We compare with the less secure DiSE schemes [AMMR18a]. In particular, DiSE was proven secure in an arguably weaker model and does not provide a way to distinguish if the initiating party is performing an decryption or encryption query. We consider the pure symmetric-key based DiSE protocol DiSE1 which utilizes an AES/PRF based DPRF. Like our TAE1 Protocol, DiSE1 does not guarantee that a ciphertext output by encryption is "well formed" if some of the parties behave maliciously. We also consider the DDH-key based DiSE protocol DiSE2 which utilizes ZK-proofs to ensure the correctness of any ciphertext output by the encryption procedure.

Our protocols are very competitive given the added security guarantees. Our symmetric-key based protocol TAE1 achieves a throughput of 778 thousand encryptions per second for $n = 3, t = 2$ while our public-key based protocol TAE3 achieves 346 encryptions per second. This is approximately 0.7 times the throughput of the weaker DiSE protocol. We observe a similar relative performance for other parameter choices when t is close to n or 2. The largest differences occurs for our TAE1 protocol when n is large and $t \approx n/2$. This results in the largest relative communication overhead compared to DiSE1 due to their protocol achieving $O(t)$ communication while ours achieves $O\binom{n}{t}$ which is maximized for $t = n/2$.

With respect to encryption latency our protocols perform similarly well. Both TAE1 and DiSE1 achieve a latency of 0.1 milliseconds for $n = 3, t = 2$ which is effectively the network latency of just sending the messages. For the public-key based protocol we again observe that the DiSE2 protocol achieves times 0.7 times improvement in latency compared to our TAE3 protocol. This added overhead consists of performing the additional threshold signature.

We argue that the presented performance evaluation shows that our protocols achieve highly practical performance. In particular, the majority of the practical applications of threshold authenticated encryption only require relatively small n, e.g. $n \in \{3, 4, 5\}$. For this range of parameters both of our protocols are highly competitive with the DiSE protocols while providing stronger security guarantees. Our schemes also preserve the property that the network communication overhead is independent of the length of the message being encryption. This property is not enjoyed by generic MPC based approaches, e.g. [KOR+17].

B Key-Reconstruction Scheme

We present a scheme which 1) reconstructs encryption key during encryption and decryption, 2) satisfies the message privacy and authenticity notion of DiSE, 3) is insecure against our IND-RCCA notion. First, we recall the message privacy notion of DiSE [AMMR18a, Section 6.2]. Roughly, the message privacy game of DiSE can be obtained by making the following modifications to our game $\mathbf{G}^{\text{ind-cca}}$ (Figure 2).

- Remove checking of privacy and authenticity conditions. The resulting game only checks if $b = b'$ after running the adversary.
- Remove oracles $\text{Eval}_{\mathsf{dec}}$, $\text{Split}_{\mathsf{dec}}$, and $\text{Combine}_{\mathsf{dec}}$.
- Allow only one call to $\text{Split}_{\mathsf{enc}}$ and $\text{Combine}_{\mathsf{enc}}$.

Note that for such a security notion, it suffices to keep the messages encrypted by honest parties private. The notion does not rule out a scheme where each party $i \in [n]$ has their own encryption key sk_i, which is secret shared among all n parties.

To construct a scheme satisfying the two properties above, we need a nonce-based authenticated encryption scheme nAE, t-out-of-n threshold signature scheme TS, and a t-out-of-n secret sharing scheme SS. The high level idea of the scheme is as follows: each party $i \in [n]$ has its own symmetric encryption key k_i, which it uses to encrypt messages. To make the scheme threshold, we secret share each k_i into n shares and distribute it amongst the n-parties. To ensure authenticity, we add a threshold signature on the commitment of the message.

First, we argue why this scheme is secure againt the DiSE notions. For message privacy, the adversary is not allowed to query any decryption oracles or even starting its own decrypton sessions. Hence, there is no way for the adversary to gain information regarding encryption keys k_i for honest parties i. Authenticity is guaranteed by the use of threshold signature.

Next, we argue why our notion of IND-RCCA rules this scheme insecure. To see this, consider the adversary that queries a challenge encryption, $\text{Split}_{\mathsf{enc}}(i, m, m)$, on two identical messages m to obtain some ciphertext c, which is then decrypted by the adversary via $\text{Split}_{\mathsf{dec}}$ and $\text{Combine}_{\mathsf{dec}}$. Note that during this process, the adversary to reconstruct the encryption key k_i for party i. Hence, the adversary is able to distinguish the bit b by decrypting any challenge ciphertext generated by party i with distinct messages.

Scheme TAE0

Proc $\mathsf{Setup}(1^\lambda, n, t)$
$(\llbracket \mathsf{sigk} \rrbracket_n, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{TS.Kg}(1^\lambda, n, t)$
For $i \in [n]$ do
  $k_i \leftarrow\!\!\$\ \mathsf{nAE.Kg}(1^\lambda)$
  $(k_{i,1}, \ldots, k_{i,n}) \leftarrow\!\!\$\ \mathsf{SS.Share}(n, t, k_i)$
For $j \in [n]$ do
  $\mathsf{sk}_j \leftarrow (j, k_{1,j}, \ldots, k_{n,j}, \mathsf{sigk}_j)$
Return $((\mathsf{sk}_1, \ldots, \mathsf{sk}_n), \mathsf{pp} \leftarrow \mathsf{vk})$

Proc $\mathsf{Split}_{\mathsf{dec}}(\mathsf{pp}, c, S)$
$(j, N, \overline{c}, \mathsf{com}, \sigma) \leftarrow c$; $\mathsf{st} \leftarrow c$
Return $((j, \mathsf{com}, \sigma), \mathsf{st})$

Proc $\mathsf{Eval}_{\mathsf{dec}}(\mathsf{pp}, \mathsf{sk}, x)$
$(j, \mathsf{com}, \sigma) \leftarrow x$
If $\mathsf{TS.Verify}(\mathsf{pp}, \mathsf{com}, \sigma) = \perp$ then return $\perp$
$(j, k_{1,j}, \ldots, k_{n,j}, \mathsf{sigk}_j) \leftarrow \mathsf{sk}$; Return $k_{i,j}$

Proc $\mathsf{Combine}_{\mathsf{dec}}(\mathsf{pp}, \mathsf{sk}, R, \mathsf{st})$
$(j, N, \overline{c}, \mathsf{com}, \sigma) \leftarrow\!\!\$\ \mathsf{st}$; $\mathsf{vk} \leftarrow \mathsf{pp}$
$(i, k_{1,i}, \ldots, k_{n,i}, \mathsf{sigk}_i) \leftarrow \mathsf{sk}$
$k \leftarrow \mathsf{SS.Recover}(R)$
If not $r \| m \leftarrow \mathsf{nAE.dec}(k, \overline{c})$ then return $\perp$
If $\mathsf{com} \neq \mathsf{Com}((m, N); r)$ then return $\perp$
If not $\mathsf{TS.Verify}(\mathsf{vk}, \mathsf{com}, \sigma)$ then return $\perp$
Return m

Proc $\mathsf{Split}_{\mathsf{enc}}(\mathsf{pp}, \mathsf{sk}, m, S)$
$N \leftarrow\!\!\$\ \{0,1\}^{\mathsf{nAE.NL}(|m|)}$
$r \leftarrow\!\!\$\ \{0,1\}^{\mathsf{Com.RL}(|(m,N)|)}$
$\mathsf{com} \leftarrow\!\!\$\ \mathsf{Com}((m, N); r)$
$\mathsf{st} \leftarrow (\mathsf{com}, m, N, r)$
Return $(\mathsf{com}, \mathsf{st})$

Proc $\mathsf{Eval}_{\mathsf{enc}}(\mathsf{pp}, \mathsf{sk}, i, x)$
$(j, k_{1,j}, \ldots, k_{n,j}, \mathsf{sigk}_j) \leftarrow \mathsf{sk}$
$\sigma \leftarrow \mathsf{TS.Sign}(\mathsf{sigk}_j, (x, i)$
Return $(k_{i,j}, \sigma)$

Proc $\mathsf{Combine}_{\mathsf{enc}}(\mathsf{pp}, \mathsf{sk}, R, \mathsf{st})$
$(i, k_{1,i}, \ldots, k_{n,i}, \mathsf{sigk}_i) \leftarrow \mathsf{sk}$
$(\mathsf{com}, m, N, r) \leftarrow \mathsf{st}$
$\{(k_{i,j}, \sigma_j) \mid j \in S\} \leftarrow R$
$\sigma \leftarrow \mathsf{TS.Combine}(\{\sigma_j \mid j \in S\})$
$k \leftarrow \mathsf{SS.Recover}(\{k_{i,j} \mid j \in S\})$
$\overline{c} \leftarrow \mathsf{nAE.enc}(k, N, r \| m)$
$c \leftarrow (i, N, \overline{c}, \mathsf{com}, \sigma)$
Return c

Figure 17: Counter example TAE scheme TAE0

C DiSE construction in Our Model

For the most part, the DiSE constructions [AMMR18a] can be proven secure in a modified version of our definition where the $\mathsf{ct}_{\mathsf{Eval}}$ counter is not only incremented on calls to $\textsc{Eval}_{\mathsf{enc}}$ but also to $\textsc{Eval}_{\mathsf{dec}}$. The addition of threshold signatures is what allows our DPRF and DVRF based constructions to avoid incrementing this counter when the adversary makes a $\textsc{Eval}_{\mathsf{dec}}$ query.

However, there is an additional issue when proving the security of DiSE construction (without strong correctness) in this modified model. Specifically, when an honest party j makes DPRF queries on α, it is possible for an adversary to manipulate the protocol so that the honest party computes the DPRF output $\beta' = \beta + \delta$ where $\beta = \mathsf{DP}(j \| \alpha)$ is the correct DPRF output and δ is some nonzero error term that the adversary introduced. In the DiSE construction which achieves strong correctness using zero knowledge proofs, this modification would be detected and the honest party would abort. However, DiSE (and this work) also considers a weaker notion of DPRF where the honest party will accept β' as a valid output. This party will then output the ciphertext

$c' = (j, \alpha, e')$ where $e' = \mathrm{G}(\beta') \oplus (m \| r)$.

For a pseudo-random generator G, it may be possible for an adversary to compute $\Delta = \mathrm{G}(\beta') \oplus \mathrm{G}(\beta)$ without knowledge of β. Doing so would require G to have some sort of homomorphic property. Adversary could then "correct" the ciphertext c' to construct a new ciphertext $c = (j, \alpha, e)$ where $e = e' \oplus \Delta = \mathrm{G}(\beta) \oplus (m \| r)$. The adversary can then submit the valid ciphertext c as a forgery.

Such attacks are not captured in the DiSE security model because the ciphertexts generated by honest parties, such as c', are not returned to the adversary. Instead, the authenticity game of DiSE only allows the adversary to perform *targeted decryption queries* wherein it could ask an honest party to initiate a decryption session with c' (by providing a reference to it).

In our security model, we *do* capture the attack described above. Adversary can call $\textsc{Split}_{\mathsf{enc}}$ and $\textsc{Combine}_{\mathsf{enc}}$ to encrypt any message m of its choice through an honest party (it picks both m_0 and m_1 to be m). In the encryption process, the adversary can provide malformed responses on behalf of corrupt parties. $\textsc{Combine}_{\mathsf{enc}}$ returns the resulting ciphertext c to the adversary, which may not be valid. If adversary could "fix" c, it could get the fixed ciphertext c' decrypted back to m with the help of $\textsc{Split}_{\mathsf{dec}}$ and $\textsc{Combine}_{\mathsf{dec}}$. In the CCA version of our security game, fresh is set to $\textsc{True}$ because $c' \notin \mathsf{EncCtxt}$. As a result, c' is added to DecSet and adversary wins the game. Adversary could even take the help of various oracles to fix c like $\textsc{Eval}_{\mathsf{dec}}$ which do not increment $\mathsf{ct}_{\mathsf{Eval}}$.

In our construction we replace the pseudo-random generator G with a random oracle. This prevents the adversary from computing Δ because the outputs $G(\beta)$ and $G(\beta')$ are completely independent of each other. We leave it to future work to determine if a weaker property on G would be sufficient to prove security of DiSE in our modified model where $\textsc{Eval}_{\mathsf{dec}}$ queries also increment $\mathsf{ct}_{\mathsf{Eval}}$.