DiSE: Distributed Symmetric-key Encryption

Shashank Agrawal[1], Payman Mohassel[2], Pratyay Mukherjee[3], and Peter Rindal*[4]

[1]Visa Research, shashank.agraval@gmail.com
[2]Visa Research, payman.mohassel@gmail.com
[3]Visa Research, pratyay85@gmail.com
[4]Oregon State University, rindalp@oregonstate.edu

Abstract

Threshold cryptography provides a mechanism for protecting secret keys by sharing them among multiple parties, who then jointly perform cryptographic operations. An attacker who corrupts upto a threshold number of parties cannot recover the secrets or violate security. Prior works in this space have mostly focused on definitions and constructions for public-key cryptography and digital signatures, and thus do not capture the security concerns and efficiency challenges of symmetric-key based applications which commonly use long-term (unprotected) master keys to protect data at rest, authenticate clients on enterprise networks, and secure data and payments on IoT devices.

We put forth the first formal treatment for *distributed symmetric-key encryption*, proposing new notions of *correctness*, *privacy* and *authenticity* in presence of malicious attackers. We provide strong and intuitive game-based definitions that are easy to understand and yield efficient constructions.

We propose a *generic construction* of threshold authenticated encryption based on *any* distributed pseudorandom function (DPRF). When instantiated with the two different DPRF constructions proposed by Naor, Pinkas and Reingold (Eurocrypt 1999) and our enhanced versions, we obtain several efficient constructions meeting different security definitions. We implement these variants and provide extensive performance comparisons. Our most efficient instantiation uses *only symmetric-key primitives* and achieves a throughput of upto *1 million* encryptions/decryptions per seconds, or alternatively a *sub-millisecond* latency with upto 18 participating parties.

*Work done as an intern at Visa Research.

Contents

1 Introduction

A central advantage of using cryptographic primitives such as symmetric-key encryption is that the safety of a large amount of sensitive data can be reduced to the safety of a very small key. To get any real benefit from this approach, however, the key must be protected securely. One could encrypt the key with another key, protect it using secure hardware (e.g. HSM, SGX or SE), or split it across multiple parties. Clearly, the first approach only shifts the problem to protecting another key. On the other hand secure hardware, co-processors and the like provide reasonable security but are not always available, are expensive or not scalable, lack programmability and are prone to side-channel attacks.

Splitting the key among multiple parties, i.e. threshold cryptography, is an effective general-purpose solution, that has recently emerged in practice as an alternative software-only solution [dya, por, sep]. Surprisingly, prior to our work, there was no formal treatment of *distributed symmetric-key encryption*. Prior formal treatments of threshold cryptography typically focus on the asymmetric-key setting, namely public-key encryption and signature schemes [DF90, DDFY94, GJKR96, CG99, DK01, AMN01, SG02, Bol03, BBH06, GHKR08, BD10] where the signing/decryption key and algorithms are distributed among multiple parties. This is despite the fact that a large fraction of applications that can benefit from stronger secret-key protection primarily use symmetric-key cryptographic primitives wherein secret keys persist for a long time. We review three such examples below:

Secret Management Systems. An increasing number of tools and popular open source software such as Keywhiz, Knox, and Hashicorp Vault (e.g. see [sec]) are designed to automate the management and protection of secrets such as sensitive data and credentials in cloud-based settings by encrypting data at rest and managing keys and authentication. These tools provide a wide range of features such as interoperability between clouds and audit/compliance support. By far, the most commonly adopted primitive for encrypting secrets in the storage backend is authenticated encryption with a *master data encryption key* that encrypts a large number of records. Some of these systems use secret sharing to provide limited key protection in an initialization stage but once keys are reconstructed in memory they remain unencrypted until the system is rebooted. Consider the following statement from Hashicorp Vault's architecture documentation [vaua]:

> "Once started, the Vault is in a sealed state ... When the Vault is initialized it generates an encryption key which is used to protect all the data. That key is protected by a master key. By default, Vault uses a technique known as Shamir's secret sharing algorithm to split the master key into 5 shares, any 3 of which are required to reconstruct the master key ... Once Vault retrieves the encryption key, it is able to decrypt the data in the storage backend, and enters the unsealed state."

Enterprise Network Authentication. Network authentication protocols such as Kerberos [kerb] are widely used to provide a single-sign-on experience to users by enabling them to authenticate periodically (e.g. once a day) to a ticket-granting service using their credentials, to obtain a ticket-granting ticket (TGT) that they use to get access to various services such as mail, printers and internal web. The recommended approach for generating the TGT is authenticated encryption (e.g. see [kera]) using a master secret key in order to provide both confidentiality and integrity for the information contained in the ticket. This renders the master secret key an important attack target, as it remains unprotected in memory over a long period.

Multi-device IoT Authentication. The proliferation of a wide range of Internet of Things (IoT) has provided users with new and convenient ways to interact with the world around them. Such devices are increasingly used to store secrets that are used to authenticate users or enable secure payments. Many IoT devices are not equipped with proper environments to store secret keys, and even when they are, provide developers with little programmability for their applications. It is therefore desirable to leverage the fact that many users own multiple devices (smart phone, smart watch, smart TV, etc.) to distribute the key material among them (instead of keeping it entirely on any single device) to enable multi-device cryptographic functionalities without making strong assumptions about a device's security features. Given the limited computation and communication power of many such IoT devices, such distributed primitives should require minimal interaction and limited cryptographic capabilities (e.g. only block-ciphers).

1.1 Technical Challenges

Modeling security As discussed earlier, existing threshold cryptographic definitions and constructions are primarily focused on public-key primitives such as digital signatures and public-key encryption. In fact, to the best of our knowledge, there is *no* standard symmetric-key security notions in the distributed setting.

To help highlight the challenges with defining a robust security model, consider a software-based encryption/authentication service wherein long-term secret keys are shared among multiple servers who collectively perform symmetric encryption, decryption, and MAC operations to store data in the cloud in an encrypted form, or to generate authentication tokens that are used to gain access to an external service. For example, the service can be used by cloud storage customers to encrypt/decrypt data on the cloud using a key that is never reconstructed after being distributed among the servers, or can be used to generate authentication tokens for a single-sign-on access control system that provides access to multiple services. A subset of these servers (below a threshold) are corrupted by an active adversary and can behave arbitrarily malicious but a secure point-to-point channel is assumed between the honest parties.

Observe that threshold authenticated encryption (TAE) is the appropriate and natural notion here as it would simultaneously solve the confidentiality and the authenticity problem, such that a ciphertext generated by the TAE scheme could be both an authentication tickets and an encrypted message. Unfortunately, while definitions for threshold public-key encryption are well-understood (e.g. see [SG98, CG99, DP08, BD10, BBH06]), they fail to capture important subtleties that only arise in the symmetric-key setting when considering standard AE notions of message privacy and ciphertext integrity [BN00, KY01, RS06].

First note that in the above scenario, servers or parties are simply workers and have no special roles in the application that uses the service. In particular, a party who initiates the ciphertext generation may not be the one initiating the decryption process, and for availability reasons, we do not assume that the same encryptor is online/available during a decryption call. This necessitates a *consistency* property where a ciphertext generated by any subset of parties should be decryptable by any other subset that is larger than a threshold.

However, what truly separates TAE from threshold public-key encryption is that in TAE a corrupted party should not be able to encrypt or decrypt messages on her own or even generate valid ciphertexts, without "being online" (i.e. without interaction with the honest parties in a distributed encryption/decryption protocol), and this should hold even if the adversary engages in other distributed encryption and decryption protocols.

Capturing all legitimate adversarial interactions in our security games is quite critical and subtle. For example, note that unlike the non-interactive setting, chosen plaintext attack (CPA) security is not sufficient to capture message privacy in the distributed setting where we need to guarantee message privacy not only in the presence of encryption queries but also during decryption queries initiated by the honest parties. In other words, the transcripts of such decryption queries should not reveal *anything* about the message being decrypted to the adversary. Second, unlike the standard (non-interactive) ciphertext integrity notions where it is shown that decryption queries cannot help the adversary and hence can be safely removed from the security game (e.g. see [BGM04]), it is easy to observe that allowing for decryption queries in the threshold setting makes the adversary strictly stronger. For instance, consider a contrived threshold scheme where all parties contacted in the decryption protocol simply return their secrets. Clearly, this scheme is not secure, but it would still satisfy a ciphertext integrity notion that does not allow the adversary to invoke the decryption protocol.

Furthermore, adversarial encryption and decryption queries are of various different flavors. Those where the adversary is the initiator (i.e. the encryptor/decryptor), and those where an honest party initiates the query (indirect queries) but the adversary arbitrarily controls the corrupted parties taking part in the protocol. In case of indirect encryption queries, in the message privacy game, we let the adversary choose the message being encrypted and learn the resulting ciphertext. This captures, for example, a scenario where a cloud storage provider that uses the service is compromised and ciphertexts generated by honest encryptors are revealed. On the other hand, in the ciphertext integrity game, it is crucial not to reveal the ciphertext to the adversary in the indirect encryption queries and require that it cannot learn the full ciphertext based on its interactions. Otherwise, an honest party's call to the encryption protocol provides the adversary with a valid ciphertext (token) that may give him access to an external service. Similar subtleties arise for decryption queries which we discuss in more detail in Section 2.

Finally, unlike the non-interactive case, defining what constitutes a valid forgery in the ciphertext integrity game is non-trivial. First, note that standard AE requires that ciphertexts produced via encryption queries are distinct from the forged ciphertext. In the interactive setting where the adversary takes part in the encryption protocol, however, generated ciphertexts may not be well-defined or valid. Moreover, there are two possible ways of testing validity of a forged ciphertext in the integrity game: (i) decrypt the forgery using an honest decryption (i.e. an execution of the decryption protocol that does not involve any corruption), or (ii) run the decryption protocol wherein adversary controls the corrupted parties. This leads to two different notions of authenticity.

Performance Challenges. In addition to not meeting our security notions, existing threshold public-key constructions are too expensive for symmetric-key use cases, as they are dominated by more expensive public-key operations and/or require extensive interaction and communication between the parties. Applications that use symmetric-key cryptography often have stringent latency or throughput requirements. Ideally, one would like the distributed encryption/decryption protocols to not be significantly more expensive than their non-distributed counterparts. In particular, the protocols should have low computation and bandwidth cost, and require minimal interaction.

General-purpose multi-party computation protocols can also be used to solve the same problems by computing standard symmetric-key encryption schemes inside an MPC (e.g. see [RSS17, dya]). While this approach has the benefit of preserving the original non-interactive

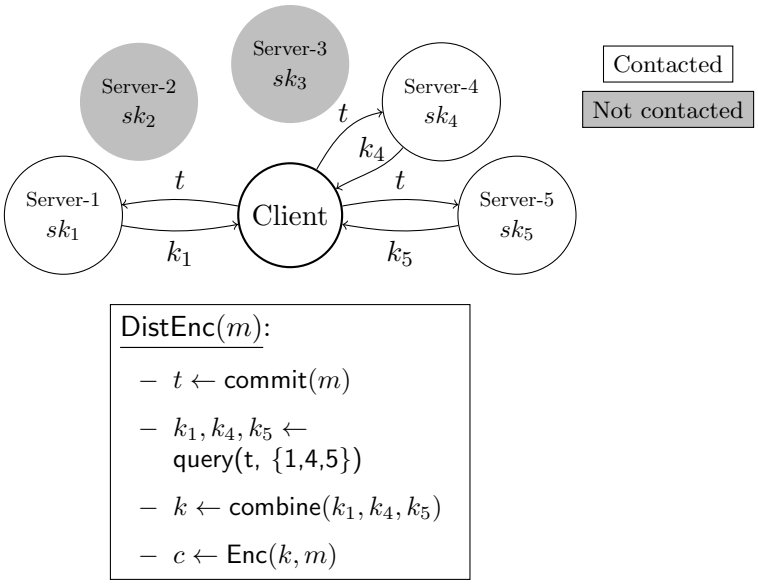

Figure 1: The flow of our distributed encryption protocol for $n = 5$ and $t = 3$. Client contacts servers 1, 4 and 5 to encrypt a message m. Servers do not communicate with each other. We show client separate from the servers for simplicity. A simplified outline of the encryption protocol is given in the box. See Figure 2 for the actual steps. The flow of decryption protocol is similar to encryption but the steps involved are different.

algorithm, the resulting protocols would be prohibitively interactive, bandwidth-intensive, and would become increasingly expensive for larger number of parties. In this paper, we aim for two-round protocols where one server sends a message to other servers and receives a response, while other servers need not exchange any messages. This minimal interaction model minimizes coordination between the servers and is ideal for low-latency applications.

We review the MPC-based solutions and other related work on protecting cryptographic secrets through splitting them among multiple parties (i.e. secret sharing, threshold PKE and threshold PRFs) in the related work section (Section 3).

1.2 Our Contribution

We formalize, design and implement new protocols for *distributed symmetric-key encryption*. Our contributions can be summarized as follows:

New security definitions. We initiate a formal study of authenticated encryption in the distributed setting. We propose novel message privacy and ciphertext integrity definitions for threshold authenticated encryption that captures the unique issues that arise in this setting, in presence of a *malicious* adversary that corrupts a subset of parties.

Simple and lightweight protocols. We put forward a *generic construction* based on any *distributed pseudorandom function* (DPRF) in the standard model. The construction only assumes one-way functions.

– When we instantiate with multiple efficient DPRF constructions from Naor et al. [NPR99] and our enhanced variants, we derive a number of threshold authenticated encryption

protocols with different security guarantees and efficiency levels (see Figure 8). All our protocols are light-weight: they require only two rounds of communication and incur minimal computation and bandwidth cost. Specifically, the party interested in encryption or decryption sends one request message to other parties and receives one response message in return (see Figure 1 for a visual depiction).[1] In the most efficient instantiation, there are no public-key operations as parties only make PRF calls and hashing.

– We provide the first formal analysis for both the PRF-based and the DDH-based instantiations of the DPRF constructions given in Naor et al. [NPR99] by proposing a strong pseudo-randomness property. We also formalize correctness of DPRFs in presence of malicious corruption and extend their DDH-based construction to satisfy this notion.

– Our protocols allow for an arbitrary threshold t such that only $t-1$ other parties need to be contacted to encrypt or decrypt. At the same time, the protocols are resilient to the corruption of $t-1$ parties (clearly, this is the best one could hope for).

Implementation and Evaluation. We implement several variants of our protocols in C++ and perform extensive experiments to evaluate their performance for applications with high-throughput and low-latency requirements. Our most efficient instantiation achieves a throughput of upto *1 million* encryptions/decryptions per seconds, or alternatively a *sub-millisecond* latency with upto 18 participating parties. We achieve this high level of performance through a variety of cryptographic optimization and system level engineering such as full use of hardware accelerated AES and instruction vectorization. The result is a lightweight challenge-response protocol where only one message is sent and received by the participating parties.

2 Technical Overview

2.1 Security Requirements

A primary contribution of this work is to present a formal treatment of symmetric-key authenticated encryption in the distributed setting.

Our definitions are inspired by the traditional game-based notions of message privacy and ciphertext integrity for standard (i.e. non-interactive) symmetric-key encryption [BN00, KY01, RS06]. We intentionally avoid the Universal Composability framework [Can01] because such definitions, proposed in prior work for standard symmetric-key encryption, are cumbersome to work with (e.g. see [KT09]).

We remark that over the past two decades, a large body of work has considered various notions of security for standard authenticated encryption [BN00, Rog02, RS06, Rog13, RS06, GL15, HRRV15, FFL12, BHT18, HKR15, BT16, BK11, PW12] to address many practical issues such as concrete security, nonce-misuse resistance, online security, and multi-user security. As the first work to formalize distributed authenticated encryption, we choose to focus on the traditional notion of AE security (i.e. message privacy + ciphertext integrity) as even extending this important notion to the threshold setting raises many new subtleties (as we will see shortly) that do not exist in the non-interactive setting. We leave it for future work to extend threshold AE to the more advanced notions mentioned above.

[1] This is in contrast with two-round MPC protocols (e.g. [MW16]) where typically in each round every participant broadcasts messages to everyone else.

The Attack Model. In the distributed setting, we consider an attacker who controls a subset of parties and behaves *arbitrarily malicious* while the honest parties are connected via point-to-point secure channels. Moreover, to capture a more realistic scenario, we let the adversary choose its corruption set after receiving the public parameters of the scheme. As we will see shortly, this requires additional care in both the constructions and the security proof.

Threshold Symmetric-key Encryption. Analogous to its non-interactive counterpart, we define a *threshold symmetric-key encryption* (TSE) scheme consisting of a setup algorithm Setup and two protocols, DistEnc and DistDec, for distributed encryption and decryption, respectively. The scheme is parameterized by two positive integers n and t, with $n \geq t \geq 2$ where n denotes the total number of parties and t the threshold. We allow at most $t-1$ corruptions which is clearly optimal in this setting. Setup generates n private keys $sk_1, sk_2, \ldots, sk_n$, one for each party, and some public parameters pp. In DistEnc, one of the parties, called the encryptor, who holds a message, sends a request message to *any* $t-1$ other parties in the protocol. The participating parties use their respective secret-keys to compute their individual responses. At the end of the protocol, only the encryptor learns a ciphertext. Analogously, in DistDec, one of the parties (decryptor) with a ciphertext performs a similar process and learns the corresponding message. Note that we do not assume that the same party plays the role of encryptor and decryptor. Our *consistency* property requires that any subset of t parties should be able to encrypt or decrypt.

Correctness. The natural correctness requirement in the non-interactive setting is that a ciphertext c generated by running an encryption algorithm on a message m must decrypt to m. But in the threshold setting where the adversary is malicious, defining correctness becomes more subtle. Informally, correctness requires that a ciphertext that is generated by an honest encryptor but may involve corrupt parties in the encryption protocol can only be decrypted (by an honest decryptor) to the correct message or results in an abort (i.e. $\perp$) even if the decryption involves corrupted parties. This notion may already be sufficient for many applications. We also formalize a stronger notion wherein any execution of an encryption protocol that potentially involves malicious parties either produces a correct ciphertext (by correct we mean that an honest decryption produces the original message) or results in an abort. In other words, a valid ciphertext carries an implicit guarantee that an honest decryption/verification will always be successful. Looking ahead, if we do not impose the stronger correctness requirement, our instantiation is significantly faster—since to achieve the stronger form of correctness we need non-interactive zero-knowledge proofs (NIZK) that require more expensive public-key operations.

Message Privacy. As discussed earlier, our definition has two components, message privacy and ciphertext integrity (also called *authenticity*). In the non-interactive case, message privacy is defined via a chosen plaintext attack (CPA) game where the adversary can engage in encryption queries before and after the challenge phase where the challenge stage consists of guessing between the ciphertexts for two adversarially chosen messages.

In the threshold setting, we allow for two types of encryption queries in the message privacy game. First, the adversary can initiate its own encryption queries using messages of its choice and obtain both the final ciphertext as well as the transcripts of the parties it corrupts (and influence their actions during encryption). Second, we allow the adversary to perform

indirect encryption queries where it invokes an honest party to initiate an encryption query using an adversary-chosen message and let the adversary learn the ciphertext (despite the fact that the TSE encryption would not necessarily leak the ciphertext to the adversary). This captures scenarios where the application using the service may unintentionally leak ciphertexts to the adversary (e.g. a cloud storage compromise or authentication token leakage). We then observe that this is not sufficient to capture full message privacy in the distributed setting. In particular, even decryption queries initiated by honest parties should preserve message privacy in presence of a malicious adversary who corrupts a subset of parties. Note that this issue does not arise in the non-interactive case where decryption queries always reveal the message. Hence, we allow these indirect decryption queries in our message privacy game and do not reveal the decrypted message to the adversary. In particular, an adversary could provide its challenge ciphertext to such an indirect decryption query and still should not be able to win the message privacy game.

Ciphertext Integrity. In the ciphertext integrity game, the adversary engages in both encryption and decryption queries, and then needs to create a new valid ciphertext (forgery). Several subtleties arise when defining a valid forgery. Let us start with the different types of encryption/decryption queries.

Similar to the message privacy game, both standard and indirect encryption queries are allowed. The ciphertexts resulting from the former are naturally not considered forgeries since the corrupt party is intended to learn it. However, in the indirect case where an honest party initiates the encryption, the security game does not provide the adversary with the resulting ciphertext. As such, the adversary is allowed to output the ciphertext of an indirect encryption query as a valid forgery if it manages to acquire one. Therefore the TSE scheme is required to prevent such attacks by making them unpredictable to him even while actively participating in the protocols.

Interestingly, we allow three types of decryption queries in the ciphertext integrity game. The adversary (i) either makes a standard decryption query where it initiates the decryption using a ciphertext of its choice and learns the decryption and transcripts of all corrupted parties; or (ii) it makes an indirect decryption query where an honest party initiates the decryption query *using a ciphertext provided by the adversary*; or (iii) makes an indirect decryption query *using a ciphertext it does not know* but that was previously generated via an indirect encryption protocol initiated by an honest party. The purpose of the third type (called *targeted decryption* queries) is to ensure that the decryption protocol initiated by an honest party does not leak the computed ciphertext to the adversary if it is the result of an earlier encryption by an honest party. To capture this, we do not count these ciphertexts towards adversary's forgery budget; in particular, the adversary wins the game if it outputs one of them as a forgery. In fact, the only decryption queries that we count towards adversary's forgery budget are of the first type, i.e. those initiated by the adversary itself. See Remark 6.11 for a more detailed discussion and how even this can be avoided at the cost of more expensive constructions.

One-More Ciphertext Integrity. To define a successful forgery in the usual non-interactive setting, one could just say that the adversary must produce a ciphertext *that is different from the ones it receives from the encryption oracle* [BN00, KY01]. Alternatively, in the case of unified definitions [RS06], the adversary is restricted from querying the decryption oracle

with a ciphertext it received from the encryption oracle[2]. Unfortunately, one cannot take a similar approach in the distributed setting. If the adversary initiates an encryption session that involves malicious parties, the output of the session (a ciphertext) may not be available to the honest parties even if they are involved. Thus, it is not clear how to explicitly define the ciphertext learned by the adversary and therefore no straightforward way to *prevent* the adversary from claiming such ciphertext as a valid forgery.

To circumvent the problem while keeping the definition simple, we keep track of the *maximum number* of ciphertexts, say k, the adversary could learn (in an ideal sense) by interacting with honest parties and require that as his forgery, he outputs $k + 1$ distinct ciphertexts that successfully decrypt. This implies that at least one of the ciphertexts he outputs is a new and valid ciphertext.

Two Notions of Authenticity. Our definition also needs to specify how the forged ciphertexts are decrypted to check their validity. The first option is to use an honest decryption (where all parties behave honestly). This is sufficient in applications where an external service would perform the decryption using the whole key (e.g. in case of single-sign-on access to a service). We refer to this variant as *authenticity* as it resembles the standard authenticity notions studied in the non-interactive setting. A second (and stronger) option is to continue using the distributed decryption protocol (where adversary actively controls a subset of parties) to decrypt the forged ciphertexts too. We refer to this variant as *strong authenticity*. We design and implement protocols meeting both notions trading off efficiency for higher security.

2.2 Our Generic Construction

We provide a brief overview of our main construction but before doing so, we discuss a few attempts that fail to meet our efficiency or security requirements. A more detailed discussion on the failed attempts can be found in Appendix B.

DPRF. All the constructions we discuss in this section use a Distributed Pseudorandom Function (DPRF) as a building block. A DPRF is a distributed analog of a standard PRF. It involves a setup where each party obtains their secret-key and the public parameters. Evaluation on an input is performed collectively by any t parties where t ($\leq n$) is a threshold. Importantly, at the end of the protocol, only one special party (evaluator) learns the output. A DPRF should meet two main requirements: (i) *consistency*: the evaluation should be independent of the participating set, and (ii) *pseudorandomness*: the evaluation's output should be pseudorandom to everyone else but the evaluator even if the adversary corrupts all other $t - 1$ parties and behaves maliciously.

In the malicious case, one can think of a slightly stronger property, called (iii) *correctness*, where after an evaluation involving up to $t - 1$ malicious corruptions, an honest evaluator either receives the correct output or can detect the malicious behavior.[3] Naor et al. [NPR99]

[2]Under the unified definition, the adversary is supposed to distinguish between two worlds, a 'real' world where access to both encryption and decryption oracle is provided, and an 'ideal' world where the encryption oracle is replaced with one that just returns random bits and the decryption oracle is replaced with one that just returns $\perp$.

[3]Looking ahead, our TSE protocol achieves strong authenticity, in which the adversary is involved in the decryption of the forgery, only if the underlying DPRF achieves correctness.

propose two very efficient (two-round) instantiations of DPRF, one based only on symmetric-key cryptography and another based on the DDH assumption. We provide the first formal proof of security for these constructions under a strong pseudo-randomness requirement. These constructions, however, do not satisfy the correctness definition (against malicious adversaries). Interestingly, we note that the recommended approach of obtaining correctness by applying a NIZK to each message of the protocol runs into a subtle technical issue, and show how to circumvent it by modifying the construction such that the public parameters provide a trapdoor commitment to the secret keys of the parties.

Attempt-0: A four-round protocol. As discussed earlier, our goal is to obtain a two-round protocol where one party sends a message to others and receives a response. But it is helpful to review a first attempt that requires four rounds of interaction and meets all our security requirements. We assume a DPRF scheme is already setup. To encrypt a message m, parties evaluate the DPRF on a random message r generated by the encryptor to obtain the output w. The encryptor then encrypts the message m using a CPA-secure symmetric-key encryption with w as the secret-key to obtain a ciphertext c. Parties then run the DPRF protocol one more time on $H(c)$ for a collision-resistance hash function H, such that the encryptor obtains the tag t. Encryptor outputs (r, c, t) as the output of the encryption protocol. The decryption protocol works as expected by first recomputing and checking t and then recovering w to decrypt c.

It is worth noting that this construction is reminiscent of the standard *encrypt-then-MAC* approach for obtaining an authenticated encryption scheme, where in one invocation the DPRF is used to generate a fresh random key for encryption and in the second invocation it is used to compute a MAC on the ciphertext. Note that the encryption protocol requires two sequential calls to the DPRF protocol, hence yielding four rounds of interaction. Interestingly, to obtain a two-round protocol, we need to deviate from this and design a protocol that roughly follows the *MAC-then-encrypt* paradigm but nevertheless meets our strong notions of security. Next we review two 2-round proposals that fail to achieve our notions.

Failed Attempt-1 [NPR99]. The first (to the best of our knowledge) proposal for a distributed encryption is due to Naor et al. [NPR99] (NPR in short). They propose to (i) first encrypt the message m locally to produce a ciphertext $e = \mathsf{SE}_w(m)$ by a "standard" symmetric-key encryption scheme SE where the key w is chosen freshly at random; (ii) then invoke the DPRF on the input $(j\|e)$ for the encryptor to obtain y, where j is the encryptor's identity; (iii) finally mask the key w with y. The final ciphertext is of the form $(j, y \oplus w, e)$. Although this achieves message privacy, it fails to achieve authenticity since the adversary, after obtaining a valid ciphertext as above, can change the key by mauling w to w' (and hence maul the ciphertext) and decrypt e with w' to produce a valid message m'. The crux of the problem is in giving the adversary the flexibility to choose the encryption key w without any checks or restrictions.

Failed Attempt-2. Another natural approach to construct distributed threshold encryption is to (i) choose a random nonce r, (ii) compute a DPRF value w on (j, r) and (iii) use w as a key for a standard authenticated encryption scheme AE to compute $e = \mathsf{AE}_w(m)$. The final ciphertext is (j, r, e). One can easily observe that, although message private, this approach does not suffice for authenticity since an attacker can make a single encryption query to obtain w and use it to encrypt more valid messages without violating the security

of the AE scheme.

Note that both attacks discussed above work even in the semi-honest setting since the corrupt parties behave honestly in all distributed protocols. In fact, the above attempts fail to achieve even a much weaker notion of authenticity which does not allow decryption queries. See Appendix B for more details.

Our Construction. At a high level, we use a DPRF scheme to generate a pseudorandom key w that is used to encrypt the message m. But to avoid the recurring problem in the failed attempts above, we need to ensure that an adversary cannot use the same w to generate any other valid ciphertext. To do so, we bind w to the message m (and the identity of party j). One way to achieve that is to use $(j\|m)$ as an input to the DPRF. First, note that it is necessary to put j inside the DPRF, otherwise a malicious attacker can easily obtain w by replaying the input of the DPRF in a new encryption query and thereby recovering any message encrypted by an honest encryptor. In the protocol we make sure each party checks if a message of the form $(j, *)$ is indeed coming from party j. Second, this does not suffice as it reveals m to all other parties during the encryption protocols originated by honest parties and as a result fails to achieve even message privacy. To overcome this, we instead input a commitment to m to the DPRF. The hiding property of the commitment ensures that m remains secret, and the binding property of the commitment binds w to this particular message. To enable the verification of the decommitment during the decryption, we need to also encrypt the commitment randomness along with m.

This almost works[4] except that the attacker can still generate valid new ciphertexts by keeping m, j and w the same and using new randomness to encrypt m. We prevent this by making the ciphertext deterministic given m and w: we input w to a pseudorandom number generator to produce a pseudorandom string serving as a "one-time pad" that is used to encrypt m just by XOR'ing (this can be thought of as applying a standard stream-cipher using w as the "random" nonce).

To summarize, our final construction can be informally described as follows: (i) the encryptor with identity j chooses a random ρ to compute $\alpha := \mathsf{Com}(m; \rho)$ where Com is a commitment and sends (j, α) to the participating parties, (ii) the participating parties then first check if the message (j, α) is indeed sent by j (otherwise they abort) and then evaluate the DPRF on $(j\|\alpha)$ for the encryptor to obtain the output w, (iii) finally, the encryptor computes $e = \mathsf{PRG}(w) \oplus (m\|\rho)$ and outputs the ciphertext (j, α, e).

In Section 7 we show that the above construction achieves consistency, message privacy and authenticity (ciphertext integrity) against a malicious adversary who corrupts up to $t-1$ parties if the underlying DPRF is consistent and pseudorandom. Moreover, if the underlying DPRF satisfies our correctness definition, then our TSE achieves strong authenticity. Note that given a DPRF, the only assumption required for the transformation is one-way functions.

3 Related Work

We briefly discuss several related research directions with similar motivations.

Secret-sharing. Secret-sharing schemes can be used to share the key for symmetric-key encryption among multiple parties, say n. They guarantee that even if up to $n-1$ parties are

[4] In fact this already satisfies a weaker notion of plaintext integrity (see Remark 6.10) since the adversary cannot forge a ciphertext for a new message.

compromised, no information about the key is leaked. A popular key management tool called Vault [vaub] takes this approach. It uses Shamir's secret sharing [Sha79] to split the master secret key into *shards*. According to the documentation [vauc], "This allows each shard of the master key to be on a distinct machine for better security." In practice, however, the master secret key is reconstructed from the shards when the Vault server is started, and remains in the memory of several—potentially, very weakly protected—parties for extended periods of time[5]. Certainly, Vault makes it easy for multiple applications or services to share the same key material but, at the same time, does not reduce key exposure in a significant way. Effectively, instead of being stored in a permanent way on multiple parties, the key material lives in memory.

Threshold PKE. Threshold public-key encryption is a well-studied problem in cryptography [Fra90, DF90, DDFY94, SG98, CG99, NPR99, DP08]. Here, the decryption key is shared among a set of parties such that at least a threshold of them are needed to decrypt any ciphertext. In some sense, threshold PKE is an analog of the problem we study here. But as discussed earlier, being a public-key notion, neither the security notions nor the efficiency requirements meet those of symmetric-key applications.

Threshold Pseudorandom Functions. To the best of our knowledge, the only threshold constructions designed for symmetric-key primitives are for pseudorandom functions [MS95, NPR99, Nie02, Dod03, DY05, DYY06, BLMR13, ECS+15]. This line of work is primarily focused on distributed PRFs (DPRF) with security in the standard model or additional properties such as verifiability or key-rotation, but does not provide definitions or constructions for the more general case of symmetric-key encryption. The only exception is the work of Naor et al. [NPR99], which also proposes a mechanism for encrypting messages using their DPRF construction. But as we have discussed (c.f. Sec 2.1, Appendix B), their proposal fails to meet our definition of threshold authenticated encryption. Nevertheless, we use Naor et al.'s DPRF constructions as the main building block in our constructions and implementations.

General-purpose MPC. Secure multi-party computation (MPC) allows multiple parties to evaluate a function over their private inputs without revealing anything about their inputs beyond the function's output. Since its introduction in early 80s, MPC has grown into a rich area with a number of different solutions of various flavors. In the last decade or so, the performance of general-purpose MPC protocols (which allow arbitrary functions to be computed) has improved substantially in both the two-party and multi-party setting [mpca, mpcb, mpcc].

However, all general-purpose MPC protocols work with a circuit representation of the function which seems to be an overkill to solve our specific problem. Furthermore, the communication complexity of these protocols typically scales linearly with the size of the circuit and the number of parties. Finally, the number of rounds of interactions is often more than two[6] for all practical MPC instantiations; and the protocols require all pairs of

[5] If not the master secret key itself, then at least the *encryption key* remains in memory. The encryption key encrypts the actual data and the master key encrypts the encryption key. We refer to the documentation for details.

[6] A recent surge of results [GGHR14, MW16, GMPP16, GS18, GS17, EC:] construct two round MPC protocols. However, these constructions focus mainly on generic feasibility and minimizing assumptions and are far from being practical.

parties to interact. Thus, a general-purpose MPC protocol for evaluating symmetric ciphers such as AES in any encryption mode [DK10, GRR+16, RSS17, dya] is too expensive of a solution for many applications of distributed symmetric-key encryption. On the other hand, MPC-based solutions are advantageous in scenarios where the desired encryption scheme is fixed and cannot be changed by the application (due to compatibility with other components or a compliance requirement to use standardized schemes such as AES) since MPC can be used to securely compute arbitrary cryptographic functions.

4 Preliminaries

In this paper, unless mentioned otherwise, we focus on challenge-response style two-round protocols: a party sends messages to some other parties and gets a response from each one of them. In particular, the parties contacted need not communicate with each other.

Common notation. Let $\mathbb{N}$ denote the set of positive integers. We use $[n]$ for $n \in \mathbb{N}$ to denote the set $\{1, 2, \ldots, n\}$. A function $f : \mathbb{N} \to \mathbb{N}$ is negligible, denoted by negl, if for every polynomial p, $f(n) < 1/p(n)$ for all large enough values of n. We use $\mathsf{D}(x) =: y$ or $y := \mathsf{D}(x)$ to denote that y is the output of the *deterministic* algorithm D on input x. Also, $\mathsf{R}(x) \to y$ or $y \leftarrow \mathsf{R}(x)$ denotes that y is the output of the *randomized* algorithm R on input x. R can be derandomized as $\mathsf{R}(x; r) =: y$, where r is the explicit random tape used by the algorithm. Finally, we write $X \sim D_S$ to denote a random variables X that follows a distribution D over a set S. For two random variables X and Y we write $X \approx_{comp} Y$ to denote that they are computationally indistinguishable and $X \approx_{stat} Y$ to denote that they are statistically close. Concatenation of two strings a and b is either denoted by $(a\|b)$ or (a, b). Throughout the paper, we use n to denote the total number of parties, t to denote the threshold, and κ to denote the security parameter. We make the natural identification between players and elements of $\{1, \ldots, n\}$.

We will use *Lagrange interpolation* for evaluating a polynomial. For any polynomial P, the i-th Lagrange coefficient for a set S to compute $P(j)$ is denoted by $\lambda_{j,i,S}$. Matching the threshold, we will mostly consider $(t-1)$-degree polynomials, unless otherwise mentioned. In this case, at least t points on P are needed to compute any $P(j)$.

Inputs and outputs. We write $[j : x]$ to denote that the value x is private to party j. For a protocol π, we write $[j : z'] \leftarrow \pi([i : (x, y)], [j : z], c)$ to denote that party i has two private inputs x and y; party j has one private input z; all the other parties have no private input; c is a common public input; and, after the execution, only j receives an output z'. We write $[i : x_i]_{\forall i \in S}$ or more compactly $[\![\mathbf{x}]\!]_S$ to denote that each party $i \in S$ has a private value x_i.

Network model. We assume that all the parties are connected by point-to-point secure and authenticated channels. We also assume that there is a known upper-bound on the time it takes to deliver a message over these channels.

Adversary model. We allow an adversary to take control of up to $t-1$ parties and make them behave in an arbitrary manner (active/malicious corruption). The set of corrupt parties is not known in advance, but we assume that it does not change during protocol execution (static corruption). We use C to denote the set of parties under the control of an adversary $\mathcal{A}$.

Cryptographic primitives. We need some standard cryptographic primitives to design our protocols like commitments, secret-sharing, non-interactive zero-knowledge proofs, etc. For completeness we define them formally in Appendix A.

5 Distributed Pseudo-random Functions: Definitions

Micali and Sydney introduced the notion of distributed pseudo-random functions in the mid 90s [MS95]. A DPRF distributes between n parties the evaluation of a function f which is an approximation of a random function, such that only authorized subsets of parties are able to compute f. A party who wants to compute $f(x)$ sends x to the parties in an authorized subset and receives information which enables her to find $f(x)$. A DPRF must be consistent in the sense that for all inputs x, all authorized subsets should lead to the same value $f(x)$.

A number of constructions and variants have been proposed over the course of more than two decades but they either involve multiple rounds of communication [Dod03], extensive interaction [Nie02, DY05], consider only passive corruption [NPR99, BLMR13], or achieve stronger properties which makes them more expensive [DYY06]. Several pseudo-randomness definitions have also been put forward in the literature, but they are not very formal or general in most cases. There are several attacks that are not explicitly captured by these definitions (though the proposed constructions may be secure against them). First, the adversary is not allowed to choose the set of parties to corrupt based on the public parameters (the only exception we know of is the definition proposed by Boneh et al. [BLMR13]). Second, it cannot obtain DPRF partial evaluations from honest parties on the challenge input (up to the threshold). Third, it is not allowed to participate in computing the DPRF on the challenge input, which may help it in distinguishing the true DPRF value from random. (Note that this last attack makes sense only under an active corruption.)

We allow the adversary to do all of the above in the pseudo-randomness game, thus obtaining a much stronger security guarantee. Apart from consistency and pseudo-randomness, we also propose a correctness property which ensures that even if corrupt parties are involved in a DPRF computation, they cannot make an honest party output a wrong value.[7] We build on the constructions of Naor et al. [NPR99] to obtain these properties from our DPRF instantiations.

Naor et al. [NPR99], however, were mainly concerned with DPRF security against semi-honest adversaries. They provide a security definition and two different constructions for such adversaries. They mention briefly that using non-interactive zero-knowledge (NIZK) proofs, one could make their DPRF constructions actively secure. However, they do not give a formal security definition for active security. It turns out that a naive application of NIZK proofs is in fact not sufficient to obtain security against malicious participants. We additionally need trapdoor commitments to satisfy the stronger pseudo-randomness requirement proposed here. Further, the fact that adversaries can obtain DPRF partial outputs on the challenge input and participate in computing the challenge DPRF value makes the proof more intricate.

We now present a formal treatment of DPRF. Similar to NPR [NPR99], we use a threshold t to capture the authorized subsets, i.e., any set of at least t parties can compute the function f. Security is provided against any set of up to $t-1$ corrupt parties.

[7] This is a weaker requirement than robustness for DPRFs which guarantees that an honest party *will* receive the correct DPRF value. However, Dodis [Dod03], for instance, assumes that the set of parties contacted by the honest party includes at least t honest parties to achieve robustness (and the proposed protocol involves several rounds of communication). We do not make any such assumption. In fact, when the threshold is close to the total number of parties, there may not be enough honest parties to fulfill the condition.

Definition 5.1 (Distributed Pseudo-random Function) A distributed pseudo-random function (DPRF) DP is a tuple of three algorithms $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Combine})$ who satisfy a consistency property.

- $\mathsf{Setup}(1^\kappa, n, t) \to ((sk_1, \ldots, sk_n), pp)$. The setup algorithm generates n secret keys (sk_1, sk_2, ..., sk_n) and public parameters pp. The i-th secret key sk_i is given to party i.
- $\mathsf{Eval}(sk_i, x, pp) \to z_i$. The Eval algorithm generates pseudo-random shares for a given value. Party i computes the i-th share z_i for a value x by running Eval with sk_i, x and pp.
- $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp) =: z/\perp$. The $\mathsf{Combine}$ algorithm combines the partial shares $\{z_i\}_{i \in S}$ from parties in the set S to generate a value z. If the algorithm fails, its output is denoted by $\perp$.

Consistency. For any $n, t \in \mathbb{N}$ such that $t \leq n$, all $((sk_1, \ldots, sk_n), pp)$ generated by $\mathsf{Setup}(1^\kappa, n, t)$, any input x, any two sets $S, S' \subset [n]$ of size at least t, there exists a negligible function negl such that

$$\Pr[\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp) = \mathsf{Combine}(\{(j, z'_j)\}_{j \in S'}, pp) \neq \perp] \geq 1 - \mathsf{negl}(\kappa),$$

where $z_i \leftarrow \mathsf{Eval}(sk_i, x, pp)$ for $i \in S$, $z'_j \leftarrow \mathsf{Eval}(sk_j, x, pp)$ for $j \in S'$, and the probability is over the randomness used by Eval.

Definition 5.2 (Security of DPRF) *Let* DP *be a distributed pseudo-random function. We say that* DP *is* secure *against malicious adversaries if it satisfies the pseudorandomness requirement (Def. 5.3). Also, we say that* DP *is* strongly-secure *against malicious adversaries if it satisfies both the pseudorandomness and correctness (Def. 5.4) requirements.*

A DPRF is *pseudorandom* if no adversary can guess the PRF value on an input for which it hasn't obtained shares from at least t parties. It is *correct* if no adversary can generate shares which lead to an incorrect PRF value. We define these properties formally below.

Definition 5.3 (Pseudorandomness) A DPRF $\mathsf{DP} := (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Combine})$ is *pseudorandom* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that

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

where $\mathsf{PseudoRand}$ is defined below.

$\mathsf{PseudoRand}_{\mathsf{DP},\mathcal{A}}(1^\kappa, b)$:

- *Initialization.* Run $\mathsf{Setup}(1^\kappa, n, t)$ to get $((sk_1, \ldots, sk_n), pp)$. Give pp to $\mathcal{A}$. Initialize a list $L := \emptyset$ to record the set of values for which $\mathcal{A}$ may know the PRF outputs.
- *Corruption.* Receive the set of corrupt parties C from $\mathcal{A}$, where $|C| < t$. Give the secret keys $\{sk_i\}_{i \in C}$ of these parties to $\mathcal{A}$. Define the corruption *gap* as $g := t - |C|$.
- *Pre-challenge evaluation queries.* In response to $\mathcal{A}$'s evaluation query $(\texttt{Eval}, x, i)$ for some $i \in [n] \setminus C$, return $\mathsf{Eval}(sk_i, x, pp)$ to $\mathcal{A}$. Repeat this step as many times as $\mathcal{A}$ desires.

– *Build the list.* Add an x to L if $|\{i \mid \mathcal{A}$ made a $(\texttt{Eval}, x, i)$ query$\}| \geq g$. In other words, if $\mathcal{A}$ contacts at least g honest parties on a value x, it has enough information to compute the PRF output on x.

– *Challenge.* $\mathcal{A}$ outputs $(\texttt{Challenge}, x^\star, S, \{(i, z_i^\star)\}_{i\in U})$ such that $|S| \geq t$ and $U \subseteq S \cap C$. If $x^\star \in L$, output 0 and stop. Let $z_i \leftarrow \mathsf{Eval}(sk_i, x, pp)$ for $i \in S \setminus U$ and $z^\star := \mathsf{Combine}(\{(i, z_i)\}_{i\in S\setminus U} \cup \{(i, z_i^\star)\}_{i\in U}, pp)$. If $z^\star = \perp$, return $\perp$. Else, if $b = 0$, return $z^\star$; otherwise, return a uniformly random value.

– *Post-challenge evaluation queries.* Same as the pre-challenge phase except that if $\mathcal{A}$ makes a query of the form $(\texttt{Eval}, x^\star, i)$ for some $i \in [n] \setminus C$ and i is the g-th party it contacted, then output 0 and stop.

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

Definition 5.4 (Correctness) A DPRF $\mathsf{DP} := (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Combine})$ is *correct* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that the following game outputs 1 with probability at least $1 - \mathsf{negl}(\kappa)$.

– *Initialization.* Run $\mathsf{Setup}(1^\kappa, n, t)$ to get $((sk_1, \ldots, sk_n), pp)$. Give pp to $\mathcal{A}$.

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

– *Evaluation* In response to $\mathcal{A}$'s evaluation query $(\texttt{Eval}, x, i)$ for some $i \in [n] \setminus C$, return $\mathsf{Eval}(sk_i, x, pp)$ to $\mathcal{A}$. Repeat this step as many times as $\mathcal{A}$ desires.

– *Computation.* Receive a set S of size at least t, an input $x^\star$, and shares $\{(i, z_i^\star)\}_{i\in S\cap C}$ from $\mathcal{A}$. Let $z_j \leftarrow \mathsf{Eval}(sk_j, x^\star, pp)$ for $j \in S$ and $z_i' \leftarrow \mathsf{Eval}(sk_i, x^\star, pp)$ for $i \in S\setminus C$. Also, let $z := \mathsf{Combine}(\{(j, z_j)\}_{j\in S}, pp)$ and $z^\star := \mathsf{Combine}(\{(i, z_i')\}_{i\in S\setminus C} \cup \{(i, z_i^\star)\}_{i\in S\cap C}, pp)$. Output 1 if $z^\star \in \{z, \perp\}$; else, output 0.

6 Threshold Symmetric-key Encryption: Definitions

In this section, we introduce threshold symmetric-key encryption (TSE) and formalize notions of correctness, message privacy, and authenticity for such schemes. We start by specifying the algorithms that constitute a TSE scheme.

Definition 6.1 (Threshold Symmetric-key Encryption) A threshold symmetric-key encryption scheme TSE is given by a tuple $(\mathsf{Setup}, \mathsf{DistEnc}, \mathsf{DistDec})$ that satisfies the consistency property below.

– $\mathsf{Setup}(1^\kappa, n, t) \to (\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$: Setup is a randomized algorithm that takes the security parameter as input, and outputs n secret keys $sk_1, \ldots, sk_n$ and public parameters pp. The i-th secret key sk_i is given to party i.

– $\mathsf{DistEnc}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : m, S], pp) \to [j : c/\perp]$: $\mathsf{DistEnc}$ is a distributed protocol through which a party j encrypts a message m with the help of parties in a set S. At the end of the protocol, j outputs a ciphertext c (or $\perp$ to denote failure). All the other parties have no output.

– $\mathsf{DistDec}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : c, S], pp) \to [j : m/\perp]$: $\mathsf{DistDec}$ is a distributed protocol through which a party j decrypts a ciphertext c with the help of parties in a set S. At the end of the protocol, j outputs a message m (or $\perp$ to denote failure). All the other parties have no output.

Consistency. For any $n, t \in \mathbb{N}$ such that $t \leq n$, all $(\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$ output by $\mathsf{Setup}(1^\kappa)$, for any message m, any two sets $S, S' \subset [n]$ such that $|S|, |S'| \geq t$, and any two parties $j \in S, j' \in S'$, if all the parties behave honestly, then there exists a negligible function negl such that

$$\Pr\Big[[j' : m] \leftarrow \mathsf{DistDec}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j' : c, S'], pp) \;\; | \;\; [j : c] \leftarrow \mathsf{DistEnc}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : m, S], pp)\Big] \geq 1 - \mathsf{negl}(\kappa),$$

where the probability is over the random coin tosses of the parties involved in $\mathsf{DistEnc}$ and $\mathsf{DistDec}$.

Definition 6.2 (Security of TSE) *Let* TSE *be a threshold symmetric-key encryption scheme. We say that* TSE *is* (strongly)-secure *against malicious adversaries if it satisfies the (strong)-correctness (Def. 6.4), message privacy (Def. 6.6) and (strong)-authenticity (Def. 6.8) requirements.*

In the security requirements that follow, the adversary is allowed to make encryption and decryption queries. In a query, it will specify a special party j who will initiate the protocol, a set of parties whom j will contact, and the input of j (message or ciphertext). The protocol will be executed as one would expect: challenger will play the role of all parties not in the control of adversary and exchange messages with it on their behalf. If j is honest, then challenger will initiate the protocol, otherwise, the adversary will initiate it. For 2-round protocols, the interaction between the challenger and adversary will be quite simple. If j is honest, then the challenger will send every message intended for a corrupt party to the adversary on behalf of j and wait to get a response from it. Challenger will then combine the response together with the response of honest parties (which it generates itself) to get the final output. On the other hand, when j is corrupt, the challenger is just supposed to respond to the messages that adversary sends to the honest parties.

From here on, we will not be explicit about the details of a protocol execution. We will just state that an instance of encryption or decryption protocol is run when adversary requests for it. Also note that although all the games below have separate encryption and decryption phases, this is only to make the definitions easy to read. The adversary is not restricted in this sense and can alternate between encryption and decryption queries.

Remark 6.3 (Relation with standard definitions) *Note that our security notion can also be thought of as a generalization of standard (non-interactive) authenticated encryption. In particular, setting* $n = 1$ *and* $t = 0$ *one gets standard CPA-security from our message privacy definition (Def. 6.6) and standard ciphertext integrity from our authenticity definition (Def. 6.8).*

6.1 Correctness

A TSE scheme is correct if whenever $\mathsf{DistEnc}$ outputs a ciphertext c for an input message m (i.e., it does not fail), then $\mathsf{DistDec}$ outputs either m or $\perp$ when run with c as input.

An adversary should not be able to influence the decryption protocol to produce a message different from m. We also consider *strong-correctness* which additionally requires that c should only decrypt to m (not even $\perp$) when decryption is performed honestly.

Definition 6.4 (Correctness) A TSE scheme $\mathsf{TSE} := (\mathsf{Setup}, \mathsf{DistEnc}, \mathsf{DistDec})$ is *correct* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that the following game outputs 1 with probability at least $1 - \mathsf{negl}(\kappa)$.

- *Initialization.* Run $\mathsf{Setup}(1^\kappa)$ to get $(\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$. Give pp to $\mathcal{A}$.
- *Corruption.* Receive the set of corrupt parties C from $\mathcal{A}$, where $|C| < t$. Give the secret-keys $\{sk_i\}_{i \in C}$ of these parties to $\mathcal{A}$.
- *Encryption.* Receive $(\texttt{Encrypt}, j, m, S)$ from $\mathcal{A}$ where $j \in S \setminus C$ and $|S| \geq t$. Initiate the protocol $\mathsf{DistEnc}$ from party j with inputs m and S. If j outputs $\perp$ at the end, then output 1 and stop. Else, let c be the output ciphertext.
- *Decryption.* Receive $(\texttt{Decrypt}, j', S')$ from $\mathcal{A}$ where $j' \in S' \setminus C$ and $|S'| \geq t$. Initiate the protocol $\mathsf{DistDec}$ from party j' with inputs c, S' and pp.
- *Output.* Output 1 if and only if j' outputs m or $\perp$.

A *strongly-correct* TSE scheme is a correct TSE scheme but with a different output step. Specifically, output 1 if and only if:

- If all parties in S' behave honestly, then j' outputs m; or,
- If corrupt parties in S' deviate from the protocol, then j' outputs m or $\perp$.

Remark 6.5 (Correctness for different applications) *In applications like key management, ciphertexts generated at some point may be decrypted much later when the plaintext is no longer available. In such cases, malformed ciphertexts must be immediately detected, hence strong correctness is needed. In applications like network authentication (Kerberos) or IoT-based payments where ciphertexts are typically decrypted shortly after encryption, the weaker notion of TSE suffices. In such cases, the outcome of decryption is known immediately and, if it is a failure, one can run another encryption session with a different set of parties.*

6.2 Message privacy

We allow for two types of encryption queries in the message privacy game: 1) the adversary can initiate an encryption session to obtain both the final ciphertext as well as the transcripts of the parties it corrupts. 2) it can make an *indirect* encryption query where it invokes an honest party to initiate an encryption session using a message of its choice. To make the definition stronger, we provide the ciphertext output by the honest party to the adversary.

However, this alone is not sufficient to capture full message privacy in the distributed setting. A *decryption session* initiated by an honest party on any ciphertext of adversary's choice (including the challenge) should not reveal what the decrypted message is either. Thus, we must allow the adversary to make such queries as well.

Definition 6.6 (Message privacy) A TSE scheme $\mathsf{TSE} := (\mathsf{Setup}, \mathsf{DistEnc}, \mathsf{DistDec})$ satisfies *message privacy* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that

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

where $\mathsf{MsgPriv}$ is defined below.

$\mathsf{MsgPriv}_{\mathsf{TSE},\mathcal{A}}(1^\kappa, b)$:

- *Initialization.* Run $\mathsf{Setup}(1^\kappa, n, t)$ to get $(\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$. Give pp to $\mathcal{A}$.
- *Corruption.* Receive the set of corrupt parties C from $\mathcal{A}$, where $|C| < t$. Give the secret keys $\{sk_i\}_{i \in C}$ of these parties to $\mathcal{A}$.
- *Pre-challenge encryption queries.* In response to $\mathcal{A}$'s encryption query $(\texttt{Encrypt}, j, m, S)$, where $j \in S$ and $|S| \geq t$, run an instance of the protocol $\mathsf{DistEnc}$ with $\mathcal{A}$[8]. If $j \notin C$, then party j initiates the protocol with inputs m and S, and the output of j is given to $\mathcal{A}$. Repeat this step as many times as $\mathcal{A}$ desires.
- *Pre-challenge indirect decryption queries.* In response to $\mathcal{A}$'s decryption query $(\texttt{Decrypt}, j, c, S)$, where $j \in S \backslash C$ and $|S| \geq t$, party j initiates $\mathsf{DistDec}$ with inputs c and S. Repeat this step as many times as $\mathcal{A}$ desires.
- *Challenge.* $\mathcal{A}$ outputs $(\texttt{Challenge}, j^\star, m_0, m_1, S^\star)$ where $|m_0| = |m_1|$, $j^\star \in S^\star \setminus C$ and $|S^\star| \geq t$. Initiate the protocol $\mathsf{DistEnc}$ from party $j^\star$ with inputs m_b and $S^\star$. Give $c^\star$ (or $\perp$) output by $j^\star$ as the challenge to $\mathcal{A}$.
- *Post-challenge encryption queries.* Repeat pre-challenge encryption phase.
- *Post-challenge indirect decryption queries.* Repeat pre-challenge decryption phase.
- *Guess.* Finally, $\mathcal{A}$ returns a guess b'. Output b'.

Remark 6.7 *When* $\mathsf{DistEnc}$ *is run in the challenge phase with* $S^\star \cap C \neq \emptyset$*, corrupt parties can easily cause the protocol to fail, leading* $j^\star$ *to output* $\perp$*. The definition above ensures that the probability that this happens cannot depend on the message* m_b*.*

6.3 Authenticity

As discussed in the overview section (Section 2.1), we cannot directly generalize the standard (non-interactive) authenticity definition to our setting for multiple reasons. First, the ability to make decryption queries gives additional power to the adversary. Second, ciphertexts generated in indirect encryption and decryption queries should remain unpredictable to the adversary or else they would enable successful forgeries. Thus, the definition we present below departs significantly from the non-interactive version.

In the definition, the variable g captures the minimum number of honest parties an adversary must contact in order to get enough information to generate one ciphertext. The variable ct counts the total number of times honest parties are contacted in encryption/decryption protocols initiated by corrupt parties. Thus, the definition requires that an efficient adversary should only be able to produce $\lfloor \mathsf{ct}/g \rfloor$ ciphertexts at the end of the game.

[8]Note that j can be either honest or corrupt here. So both types of encryption queries are captured.

We present two variants of the definition. In the first notion, the forged ciphertexts output by an adversary at the end of the game are decrypted in an honest manner, i.e., all the parties involved in decryption follow the protocol. On the other hand, our stronger authenticity notion allows the adversary to influence the decryption process. A forged ciphertext that may otherwise not decrypt successfully, could be decryptable if corrupt parties manipulate their responses. Thus, there could be ciphertexts that are valid forgeries in the strong authenticity game but not in the standard one.

Recall that a targeted decryption query provides a way for an adversary to ask an honest party to initiate a decryption session on a ciphertext that was previously generated by some honest party, since such a ciphertext may not be available to the adversary. Just as in regular encryption/decryption sessions initiated by honest parties, the counter ct is not updated in a targeted decryption session because we want to capture that the adversary does not get any useful information towards generating new ciphertexts in such a session.

Definition 6.8 (Authenticity) A TSE scheme $\mathsf{TSE} := (\mathsf{Setup}, \mathsf{DistEnc}, \mathsf{DistDec})$ satisfies *authenticity* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that

$$\Pr\left[\mathsf{AUTH}_{\mathsf{TSE},\mathcal{A}}(1^\kappa) = 1\right] \leq \mathsf{negl}(\kappa),$$

where AUTH is defined below.

$\underline{\mathsf{AUTH}_{\mathsf{TSE},\mathcal{A}}(1^\kappa)}$:

- *Initialization.* Run $\mathsf{Setup}(1^\kappa, n, t)$ to get $(\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$. Give pp to $\mathcal{A}$. Initialize a counter $\mathsf{ct} := 0$ and an *ordered* list $L_{\mathsf{ctxt}} := \emptyset$. Below, we assume that for every query, the (j, S) output by $\mathcal{A}$ are such that $j \in S$ and $|S| \geq t$.

- *Corruption.* Receive the set of corrupt parties C from $\mathcal{A}$, where $|C| < t$. Give the secret keys $\{sk_i\}_{i \in C}$ of these parties to $\mathcal{A}$. Define the gap between the threshold and the number of corrupt parties as $g := t - |C|$.

- *Encryption queries.* On receiving $(\texttt{Encrypt}, j, m, S)$ from $\mathcal{A}$, run the protocol $\mathsf{DistEnc}$ with m, S as the inputs of j. If $j \in C$, increment ct by $|S \setminus C|$ (number of honest parties in S). Else, append the ciphertext output by j to L_{ctxt}.

- *Decryption queries.* On receiving $(\texttt{Decrypt}, j, c, S)$ from $\mathcal{A}$, run the protocol $\mathsf{DistDec}$ with c, S as the inputs of j. If $j \in C$, increment ct by $|S \setminus C|$.

- *Targeted decryption queries.* On receiving $(\texttt{TargetDecrypt}, j, \ell, S)$ from $\mathcal{A}$ for some $j \in S \setminus C$, run $\mathsf{DistDec}$ with c, S as the inputs of j, where c is the ℓ-th ciphertext in L_{ctxt}.

- *Forgery.* Let $k := \lfloor \mathsf{ct}/g \rfloor$. $\mathcal{A}$ outputs $((j_1, S_1, c_1), (j_2, S_2, c_2), \ldots, (j_{k+1}, S_{k+1}, c_{k+1}))$ such that $j_1, \ldots, j_{k+1} \notin C$ and $c_u \neq c_v$ for any $u \neq v \in [k+1]$ (ciphertexts are not repeated). For every $i \in [k+1]$, run an instance of $\mathsf{DistDec}$ with c_i, S_i as the input of party j_i. In that instance, all parties in S_i behave honestly. Output 0 if any j_i outputs $\perp$; else output 1.

A TSE scheme satisfies *strong-authenticity* if it satisfies authenticity but with a slightly modified AUTH: In the forgery phase, the restriction on corrupt parties in S_i to behave honestly is removed (for all $i \in [k+1]$).

Remark 6.9 (Authenticity for different applications) *When protecting data at rest, an application may require that both encryption and decryption are distributed. If adversary can also interfere with decryption, the stronger version of authenticity should be used. In case of authentication tokens generated for an external service, the decryption is likely to be performed by a third party who holds the full key in a secure environment. Hence, the weaker notion of authenticity may suffice.*

As we will see later, our TSE construction requires a stronger property from the underlying DPRF to achieve the stronger form of authenticity and as a result require the use of zero-knowledge proofs, but the normal form of authenticity can be achieved without it.

Remark 6.10 (Integrity of plaintexts) *In the non-interactive setting for authenticated encryption, a weaker form of INT-CTXT, called integrity of plaintexts (INT-PTXT) [BN00], has also been studied. If a forged ciphertext decrypts to a message encrypted earlier by the adversary, then it is not considered a valid forgery in the INT-PTXT game. One can also weaken our authenticity definition in a similar fashion: a sequence of ℓ forgeries would be accepted only if they decrypt to ℓ unique messages. See Lemma 7.5 for how this notion comes up in the distributed setting.*

Remark 6.11 (Not updating counter for decryption) *In our current authenticity definitions, we increment the counter* ct *only for decryption queries initiated by the adversary (not for indirect or targeted queries), implying that a ciphertext the adversary could deduce from such an interaction is not considered a successful forgery. Though it may seem at first that we are increasing the attack surface (note that direct encryption queries are already counted towards this), the extra information leakage may not make a significant difference in practice, especially when applications restrict and/or log who initiates decryption and what can be decrypted by whom.*

One can modify our construction to satisfy an even stronger notion where even decryption queries initiated by the adversary are not counted towards its forgery budget. For example, in parallel to evaluating the DPRF on $j\|\alpha$, a threshold signature on the same input can be computed. Then, during decryption, parties first check the validity of the signature before responding with their partial share of the DPRF value. However, adding an invocation of a threshold signature scheme to DiSE would be a significant overhead and would eliminate the possibility of a symmetric-key only solution.

7 Our Construction: DiSE

In this section, we put forward our main construction DiSE, based on any DPRF. A full description of the construction is provided in Figure 2. (See Section 2.2 for an overview.) We prove that if the DPRF is (strongly) secure, then DiSE is (strongly) secure too. We provide concrete DPRF instantiations in Section 8.

Theorem 7.1 *The TSE scheme DiSE of Figure 2 is (strongly)-secure if the underlying DPRF* DP *is (strongly)-secure.*

Proof. We show each property of DiSE separately.

Consistency. Recall that consistency is required only when all the parties behave honestly. Thus, consistency of DiSE follows easily from the consistency of DP.

Ingredients:

- An (n,t)-DPRF protocol $\mathsf{DP} := (\mathsf{DP.Setup}, \mathsf{Eval}, \mathsf{Combine})$ (Def. 5.1).
- A pseudorandom generator PRG of polynomial stretch.
- A commitment scheme $\Sigma := (\Sigma.\mathsf{Setup}, \mathsf{Com})$ (Def. A.2).

$\underline{\mathsf{Setup}(1^\kappa, n, t) \to \left([\![\mathbf{sk}]\!]_{[n]}, pp\right)}$: Run $\mathsf{DP.Setup}(1^\kappa, n, t)$ to get $((rk_1, \ldots, rk_n), pp_{\mathsf{DP}})$ and $\Sigma.\mathsf{Setup}(1^\kappa)$ to get pp_{com}. Set $sk_i := rk_i$ for $i \in [n]$ and $pp := (pp_{\mathsf{DP}}, pp_{\mathsf{com}})$.

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

- Party j computes $\alpha := \mathsf{Com}(m, pp_{\mathsf{com}}; \rho)$ for a randomly chosen ρ and sends α to all parties in S.
- For every $i \in S$, party i runs $\mathsf{Eval}(sk_i, j\|\alpha, pp)$ to get z_i, and sends it to party j.
- Party j runs $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp)$ to get w or $\perp$. In the latter case, it outputs $\perp$. Otherwise, it computes $e := \mathsf{PRG}(w) \oplus (m\|\rho)$ and then outputs $c := (j, \alpha, e)$.

$\underline{\mathsf{DistDec}([\![\mathbf{sk}]\!]_{[n]}, [j' : c, S], pp) \to [j' : m/\perp]}$: To decrypt a ciphertext c with the help of parties in S:

- Party j' first parses c into (j, α, e). Then it sends $j\|\alpha$ to all the parties in S.
- For $i \in S$, party i receives x and checks if it is of the form $j^\star\|\alpha^\star$ for some $j^\star \in [n]$. If not, then it sends $\perp$ to party j'. Else, it runs $\mathsf{Eval}(sk_i, x, pp)$ to get z_i, and sends it to party j'.
- Party j' runs $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp)$ to get w or $\perp$. In the latter case, it outputs $\perp$. Otherwise, it computes $m\|\rho := \mathsf{PRG}(w) \oplus e$ and checks if $\alpha = \mathsf{Com}(m, pp_{\mathsf{com}}; \rho)$. If the check succeeds, it outputs m; otherwise, it outputs $\perp$.

Figure 2: DiSE: our threshold symmetric-key encryption protocol.

Lemma 7.2 (Correctness) *DiSE is a correct* TSE *scheme.*

Proof. A TSE scheme is correct if whenever an honest party j initiates $\mathsf{DistEnc}$ on a message m to obtain a ciphertext c (i.e., $\mathsf{DistEnc}$ does not fail), any honest party j' recovers m itself (or $\perp$) when it runs $\mathsf{DistDec}$ with c as input (except with negligible probability). In the protocol DiSE, c is of the form (j, α, e) where $\alpha := \mathsf{Com}(m, pp_{\mathsf{com}}; \rho)$ is generated locally. Any decryptor must verify that the message m' and randomness ρ' that it recovers (if $\mathsf{Combine}$ does not fail) satisfy $\alpha = \mathsf{Com}(m', pp_{\mathsf{com}}; \rho')$ or not. If Σ is a binding commitment scheme, then this verification succeeds only if $m = m'$, except with negligible probability. ■

Lemma 7.3 (Strong-correctness) *If* DP *satisfies the correctness property, then DiSE is a strongly-correct* TSE *scheme.*

Proof. For a TSE scheme to be strongly-correct, we also need that if all the parties involved in decryption behave honestly, then a ciphertext $c := (j, \alpha, e)$, where $e := \mathsf{PRG}(w) \oplus (m\|\rho)$, generated by an honest party (possibly involving some corrupt parties) should decrypt to the right message with high probability. Now the correctness property of DP guarantees that if all the parties involved in decryption are honest w' obtained through $\mathsf{Combine}$ during decryption will be the same as the w obtained during encryption except with negligible probability (as the input to the DPRF is the same $j\|\alpha$). Therefore, $\mathsf{PRG}(w') \oplus e$ in the last step of decryption would give $\mathsf{PRG}(w') \oplus \mathsf{PRG}(w) \oplus (m\|\rho) = m\|\rho$. ■

For the following three lemma, we provide a sketch here and defer formal proofs to Appendix C.

Lemma 7.4 (Message privacy) *If* DP *is a secure DPRF, then DiSE is a message-private* TSE *scheme.*

Proof sketch. The challenge ciphertext $c^\star$ has the form $(j^\star, \alpha^\star, e^\star)$ where $e^\star = \mathsf{PRG}(w^\star) \oplus (m_b \| \rho^\star)$, $\alpha^\star = \mathsf{Com}(m_b, pp_{\mathsf{com}}; \rho^\star)$ and $w^\star$ is the output of DPRF DP on $j^\star \| \alpha^\star$. One can think about the masking with PRG as a symmetric-key encryption using a stream cipher. So, an adversary $\mathcal{A}$ will find it computationally hard to guess b if $w^\star$ is indistinguishable from random. The pseudorandomness property of DP ensures this as long as $\mathcal{A}$ has no way of evaluating the DPRF on $j^\star \| \alpha^\star$ itself. (Note that $\alpha^\star$ does not reveal information about m_b due to the hiding property of Σ.)

If a corrupt party initiates an encryption protocol, then $\mathcal{A}$ can learn $j \| \alpha^\star$ for any j because $\alpha^\star$ is not hidden from it, but j would never be equal to $j^\star$ since $j^\star$ is an honest party. On the other hand, even if $\mathcal{A}$ asks party $j^\star$ to initiate encryption, $j^\star$ would compute DPRF on a value $\alpha \neq \alpha^\star$ due to the binding property of Σ. As a result, no matter how an encryption query is crafted, $\mathcal{A}$ cannot compute the DPRF on $j^\star \| \alpha^\star$. See Appendix C.1 for a detailed proof. ■

Lemma 7.5 (Authenticity) *If* DP *is a secure DPRF, then DiSE is a* TSE *scheme that satisfies authenticity.*

Proof sketch. Among the forged ciphertexts output by adversary, suppose there are two ciphertexts c_1, c_2 ($c_1 \neq c_2$) with the same j and commitment α. When these two are decrypted with possibly different sets of parties, the DPRF value recovered would be the same due to the consistency property of DP (it is assumed that all parties involved in decryption behave honestly). As a result, (m_1, ρ_1) and (m_2, ρ_2) recovered from c_1 and c_2, respectively, would be different. Due to the binding property of Σ, α cannot be a commitment to both. Hence, decryption of at least one of c_1, c_2 fails, and AUTH outputs 0. Therefore, if an adversary must succeed, each of the $k+1$ ciphertexts must have unique (j, α).

Recall that a valid adversary is allowed to contact honest parties strictly less than $k \cdot g$ number of times. So one can find at least one (j, α) among the forged ciphertexts for which adversary has not contacted g parties. Due to the pseudorandomness property of DP, the adversary does not know the value of DPRF on (j, α). Hence, it can not produce a valid ciphertext with it.

A detailed proof can be found in Appendix C.2. Note that if parties involved in the decryption of forged ciphertexts are allowed to act maliciously, we cannot invoke DPRF's consistency property. However, the adversary would still not be able to make sure that c_1, c_2 decrypt successfully to two distinct messages because the commitment is binding. Thus, DiSE can be shown to satisfy a strong notion of an INT-PTXT-style definition in the distributed setting (see Remark 6.10). ■

Lemma 7.6 (Strong-authenticity) *If* DP *is a strongly-secure DPRF, then DiSE is a* TSE *scheme that satisfies strong-authenticity.*

Proof sketch. Strong authenticity gives additional power to the adversary. In the decryption of forged ciphertexts, corrupt parties can deviate from the protocol arbitrarily. Thus, unlike

above, consistency of DP alone would not suffice. Using both consistency and correctness though, one can argue that even if c_1, c_2 are decrypted with different sets of parties, the recovered DPRF values w_1, w_2 are either the same or $\perp$. In the latter case, AUTH clearly outputs 0, and, in the former, it outputs 0 for the same reason as above.

The rest of the proof is similar to the one for weak-authenticity with some minor changes in how the pseudorandomness guarantee is reduced to authenticity. See Appendix C.3 for more details. ■ ■

Remark 7.7 (Key-management application) *As discussed in the introduction (c.f. Section 1), a main motivation of this work is to strengthen the security of key-management applications like Hashicorp Vault [vaub]. For such applications, DiSE should be viewed as distributing the role of the key-manager itself. Multiple servers would keep shares of the master secret key (which is used to encrypt various types of secrets) and know about each other's identity. Clients of the key-management application would need to authenticate via a separate mechanism.*

Remark 7.8 (Other definitions of security) *Through Theorem 7.1, we study two forms of security for TSE in this paper. The stronger form combines strong correctness with strong authenticity and the normal form combines their normal versions. One could consider other possibilities too like combining strong correctness with normal authenticity. The exact requirements would depend on the application for which TSE is being used (see remarks 6.5 and 6.9).*

8 Instantiations of Distributed Pseudorandom Functions

In this section, we revisit the distributed pseudo-random function (DPRF) constructions of Naor, Pinkas, and Reingold [NPR99] (henceforth NPR) and study the properties defined in Section 5.

NPR proposed two different instantiations of DPRF, one based on the decisional Diffie-Hellman assumption (DDH) and another based on any PRF. They showed that their constructions are secure against semi-honest adversaries, and briefly discussed how the first construction (DDH-based) could be extended to the malicious setting. Below, we present the two instantiations in their original form, and show that both achieve our pseudorandomness requirement against malicious adversaries (Def. 5.3). As discussed in Section 5, our definition captures several attacks that were not considered before. Thus, the proofs require significantly more care. Further, building on the idea mentioned in NPR, we strengthen the DDH-based construction with a NIZK proof (specifically, Schnorr's proof [Sch90, CV90] via the Fiat-Shamir transform [FS87]) to obtain *strong security*. However, it turns out that in addition to the application of NIZKs, we need to use trapdoor commitments to commit to secret key shares of parties in order to achieve our stronger pseudorandomness property. We also briefly discuss how to strengthen the PRF-based construction to make it strongly secure using only symmetric-key primitives.

8.1 DDH-based construction

NPR's first DPRF is based on any multiplicative group G of prime order p in which DDH holds. The PRF functionality being computed collectively can be written as $f_s(x) = H(x)^s$,

where $H : \{0,1\}^* \to G$ is a hash function (modeled as a random oracle) and the key is $s \in \mathbb{Z}_p$. To distribute the evaluation of f, the secret key s must be secret shared between the parties.

In the setup phase, a trusted party samples a master key $s \leftarrow_\$ \mathbb{Z}_p$ and uses Shamir's secret sharing scheme with a threshold t to create n shares $s_1, \ldots, s_n$ of s. Share s_i is given privately to the party i. We know that for any set S of $\ell \geq t$ parties $S := \{i_1, ..., i_\ell\} \subseteq [n]$, there exists integers (i.e. Lagrange coefficients) $\lambda_{0,1,S}, \ldots, \lambda_{0,\ell,S} \in \mathbb{Z}_p$ such that $\sum_{j \in S} s_{i_j} \lambda_{0,j,S} = s$. Therefore, it holds that

$$f_s(x) = H(x)^s = H(x)^{\sum_{j=1}^{\ell} \lambda_{0,j,S} s_{i_j}} = \prod_{i=1}^{\ell} \left(H(x)^{s_{i_j}}\right)^{\lambda_{0,j,S}},$$

which can be computed in a distributed manner running the protocol $\Pi_{\mathsf{DDH\text{-}DP}}$ as shown in Figure 3. This protocol satisfies the pseudorandomness definition (Def. 5.3), but *not* the correctness definition (Def. 5.4). Formally we show that:

Parameters: Let $G = \langle g \rangle$ be a multiplicative cyclic group of prime order p in which the DDH assumption holds and $\mathcal{H} : \{0,1\}^* \to G$ be a hash function modeled as a random oracle. Let SSS be Shamir's secret sharing scheme (Def. A.5).

- $\mathsf{Setup}(1^\kappa, n, t) \to (\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$: Sample $s \leftarrow_\$ \mathbb{Z}_p$ and get $(s_1, \ldots, s_n) \leftarrow \mathsf{SSS}(n, t, p, s)$. Set $pp := (p, g, G)$ and $sk_i := s_i$ and give (sk_i, pp) to party i, for $i \in [n]$.
- $\mathsf{Eval}(sk_i, x, pp) \to z_i$: Compute $w := \mathcal{H}(x)$, $h_i := w^{sk_i}$ and output h_i.
- $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp) =: z/\perp$: If $|S| < t$, output $\perp$. Else, parse z_i as (h_i) for $i \in S$. Output $\prod_{i \in S} h_i^{\lambda_{0,i,S}}$.

Figure 3: A secure DPRF protocol $\Pi_{\mathsf{DDH\text{-}DP}}$ based on DDH.

Theorem 8.1 *Protocol $\Pi_{\mathsf{DDH\text{-}DP}}$ in Figure 3 is a secure DPRF under the DDH assumption in the programmable random oracle model.*

We provide a detailed formal analysis in Appendix C.4. Here we provide a brief sketch. Consistency follows from Shamir's secret sharing in a straightforward way. The pseudorandomness property can be reduced from the hardness of DDH assumption. Intuitively, since the attacker is restricted to get at most $t-1$ evaluations of the secret polynomial for the challenge $x^\star$, it does not have enough information whether the returned value in the challenge phase lies on the secret polynomial or not. However, subtleties arise due to the fact that the adversary may obtain more than $t-1$ evaluations through queries on $x \neq x^\star$, and hence the above argument must hold conditioned on those values. Fortunately, via a sequence of hybrids, we can gradually move to a game in which a fresh random polynomial is selected for each different x that is correlated with the secret polynomial on up to $\ell \leq t-1$ points, where ℓ is the number of corrupt parties (as those many secrets can be obtained by the adversary via corruption). Clearly, $t-1$ evaluations point give no information about the secret $(t-1)$-degree polynomial. Therefore, the response on challenge $x^\star$ is indistinguishable from random.

Strong security Adding trapdoor commitments and NIZK proofs in the RO model (for a statement slightly different from the one suggested by NPR) appropriately to $\Pi_{\mathsf{DDH\text{-}DP}}$, we obtain the protocol $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$, described in detail in Figure 4. This protocol also satisfies correctness and hence achieves *strong security*. Formally:

Theorem 8.2 *Protocol* $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$ *in Figure 4 is a* strongly *secure DPRF under the DDH assumption in the programmable random oracle model.*

A detailed proof is provided in Appendix C.5. The proof for pseudorandomness follows the same structure as before. However, to accommodate the changes (trapdoor commitments and NIZK proofs), some additional effort is needed. Note that NIZK proofs are used by each party i to show that they use the correct secret-share s_i which is committed as γ_i in the public parameters. However, since the adversary is allowed to corrupt the parties after obtaining the public parameters, we need trapdoor commitments to make sure that the commitments can be opened to some different values later by the simulator with a trapdoor. The correctness property follows from the extractability of NIZK and binding of commitments.

Parameters: Let $G = \langle g \rangle$ be a multiplicative cyclic group of prime order p in which the DDH assumption holds, $\mathcal{H} : \{0,1\}^* \to G$ and $\mathcal{H}' : \{0,1\}^* \to \{0,1\}^{\mathsf{poly}(\kappa)}$ be two hash functions modeled as random oracles. Let SSS be Shamir's secret sharing scheme (Def. A.5), $\mathsf{TDC} := (\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ be a trapdoor commitment scheme (Def. A.3) and $\mathsf{NIZK} := (\mathsf{Prove}^{\mathcal{H}'}, \mathsf{Verify}^{\mathcal{H}'})$ be a simulation-sound NIZK proof system (Def. A.6).

- $\mathsf{Setup}(1^\kappa, n, t) \to (\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$. Sample $s \leftarrow_\$ \mathbb{Z}_p$ and get $(s_1, \ldots, s_n) \leftarrow \mathsf{SSS}(n, t, p, s)$. Run $\mathsf{Setup}_{\mathsf{com}}(1^\kappa)$ to get pp_{com}. Compute a commitment $\gamma_i := \mathsf{Com}(s_i, pp_{\mathsf{com}}; r_i)$ by picking r_i at random. Set $pp = (p, g, G, \gamma_1, \ldots, \gamma_n, pp_{\mathsf{com}})$, $sk_i := (s_i, r_i)$ and give sk_i to party i, for $i \in [n]$.
- $\mathsf{Eval}(sk_i, x, pp) \to z_i$. Compute $w := H(x)$ and $h_i := w^{s_i}$. Run $\mathsf{Prove}^{\mathcal{H}'}$ with the statement stmt_i: $\{\exists s, r \text{ s.t. } h_i = w^s \wedge \gamma_i = \mathsf{Com}(s, pp_{\mathsf{com}}; r)\}$ and witness (s_i, r_i) to obtain a proof π_i. Output $((w, h_i), \pi_i)$.
- $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp) =: z/\perp$. If $|S| < t$, output $\perp$. Else, parse z_i as $((w, h_i), \pi_i)$ and check if $\mathsf{Verify}^{\mathcal{H}'}(\mathsf{stmt}_i, \pi_i) = 1$ for all $i \in S$. If check fails for any i, output $\perp$. Else, output $\prod_{i \in S} h_i^{\lambda_{0,i,S}}$.

Figure 4: A *strongly* secure DPRF protocol $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$ based on DDH. Differences from $\Pi_{\mathsf{DDH\text{-}DP}}$ are highlighted in blue.

Parameters: Let $G = \langle g \rangle$ be a multiplicative cyclic group of prime order p in which the DDH assumption holds, $\mathcal{H} : \{0,1\}^* \to G$ and $\mathcal{H}' : \{0,1\}^* \to \mathbb{Z}_p$ be hash functions. Let SSS be Shamir's secret sharing scheme (Def. A.5).

- $\mathsf{Setup}(1^\kappa, n, t) \to (\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$. Sample $s \leftarrow_\$ \mathbb{Z}_p$ and get $(s_1, \ldots, s_n) \leftarrow \mathsf{SSS}(n, t, p, s)$. Sample a generator h of G at random. Compute a commitment $\gamma_i := g^{s_i} \cdot h^{r_i}$ to s_i by picking $r_i \leftarrow_\$ \mathbb{Z}_p$. Set $pp = (p, g, G, \mathcal{H}, \mathcal{H}', \gamma_1, \ldots, \gamma_n, h)$, $sk_i := (s_i, r_i)$ and give sk_i to party i, for $i \in [n]$.
- $\mathsf{Eval}(sk_i, x, pp) \to z_i$. Compute $w := \mathcal{H}(x)$ and $h_i := w^{s_i}$. Pick $v_i, v_i' \leftarrow_\$ \mathbb{Z}_p$ and set $t_i := w^{v_i}$, $t_i' := g^{v_i} \cdot h^{v_i'}$. Compute a hash $c_i := \mathcal{H}'(h_i, w, \gamma_i, g, h, t_i, t_i')$, $u_i := v_i - c_i \cdot s_i$ and $u_i' := v_i' - c_i \cdot r_i$. Define π_i to be (c_i, u_i, u_i') and output $((w, h_i), \pi_i)$.
- $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp) =: z/\perp$. If $|S| < t$, output $\perp$. Else, parse z_i as $((w, h_i), (c_i, u_i, u_i'))$ for $i \in S$. Compute $t_i := w^{u_i} \cdot h_i^{c_i}$, $t_i' := g^{u_i} \cdot h^{u_i'} \cdot \gamma_i^{c_i}$ and check if $c_i = \mathcal{H}'(h_i, w, \gamma_i, g, h, t_i, t_i')$. If check fails for any $i \in S$, output $\perp$. Else, output $\prod_{i \in S} h_i^{\lambda_{0,i,S}}$.

Figure 5: A concrete instantiation of the protocol $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$ from Figure 4 using Pedersen commitment and Schnorr-style proof (via the Fiat-Shamir transform).

Parameters: Let $G = \langle g \rangle$ be a multiplicative cyclic group of prime order p in which the DDH assumption holds, $\mathcal{H} : \{0,1\}^* \to G$ and $\mathcal{H}' : \{0,1\}^* \to \mathbb{Z}_p$ be hash functions. Let SSS be Shamir's secret sharing scheme (Def. A.5).

- $\mathsf{Setup}(1^\kappa, n, t) \to (\llbracket \mathbf{sk} \rrbracket_{[n]}, pp)$. Sample $s \leftarrow_\$ \mathbb{Z}_p$ and get $(s_1, \ldots, s_n) \leftarrow \mathsf{SSS}(n, t, p, s)$. Set $pp = (p, g, G, \mathcal{H}, \mathcal{H}', g^{s_1}, \ldots, g^{s_n})$, $sk_i := s_i$ and give sk_i to party i, for $i \in [n]$.
- $\mathsf{Eval}(sk_i, x, pp) \to z_i$. Compute $w := \mathcal{H}(x)$ and $h_i := w^{sk_i}$. Pick $v_i \leftarrow_\$ \mathbb{Z}_p$ and set $t_i := g^{v_i}$. Compute a hash $c_i := \mathcal{H}'(h_i, w, g^{sk_i}, g, t_i)$ and $u_i := v_i - c_i \cdot sk_i$. Define π_i to be (c_i, u_i) and output $((w, h_i), \pi_i)$.
- $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp) =: z/\perp$. If $|S| < t$, output $\perp$. Else, parse z_i as $((w, h_i), (c_i, u_i))$ for $i \in S$. Compute $t_i := w^{u_i} \cdot h_i^{c_i}$ and check if $c_i = \mathcal{H}'(h_i, w, g^{sk_i}, g, t_i)$. If check fails for any $i \in S$, output $\perp$. Else, output $\prod_{i \in S} h_i^{\lambda_{0,i,S}}$.

Figure 6: A privately verifiable version of the protocol from Figure 5.

An efficient way to instantiate trapdoor commitments and NIZK arguments of knowledge (in the random oracle model) is via Pedersen commitments and Fiat-Shamir transformation on Schnorr-style proofs. We give this concrete version of $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$ in Figure 5 and use it for our experiments in the following section. The concrete protocol remains secure under DDH (in random oracle model).

Remark 8.3 *Besides correctness, protocol of Figure 4 has the additional property that each party's proof can be publicly verified, i.e. the* Combine *algorithm only takes public inputs and the public messages sent/received. In particular, even an external party who does not hold any secrets, given the partial DPRF values and the NIZK proofs, can publicly verify that the DPRF was computed correctly. This may be useful in applications where an external party wants to verify the correctness of a token. But if we settle for strong correctness with only private verifiability, we can obtain a more efficient protocol. In particular, instead of publicly committing to the secret keys, each party can be given g^{s_i} for all i as part of its secret key in the setup, and the Schnorr-based NIZK can be simplified to reduce the number of required exponentiations. In the experiment section we implement both variants and show that the privately verifiable version is 25% faster than the publicly verifiable version. A concrete construction is provided in Figure 6.*

8.2 PRF-based construction

NPR also presented a DPRF construction based on any PRF, e.g. AES. [9] To obtain an t-out-of-n threshold, this protocol incurs an exponential overhead of $O(n^{\min(t,n-t)})$. However, for $n < 20$ or $t \approx n$ it can significantly outperform the previously described DDH based construction (see Section 9).

In the setup phase of the protocol, $d := \binom{n}{n-t+1}$ random numbers $k_1, ..., k_d$ are chosen. We assume that d is polynomial in the security parameter so that all the DPRF algorithms are polynomial time. Let $D_1, \ldots, D_d$ be the d distinct $(n-t+1)$-sized subsets of $[n]$. Then, the i-th random number is given to all parties in the set D_i. The DPRF is defined as $F_k(x) = \bigoplus_{i=1}^{d} f_{k_i}(x)$, where f can be any PRF. Since all the d keys are needed to compute F_k, no set S of parties of size less than t can compute F_k by itself (at least one of the

[9] Micali and Sydney provided a similar construction but for more general access structures [MS95].

Parameters: Let $f : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^*$ be a pseudo-random function.

- $\mathsf{Setup}(1^\kappa, n, t) \to (\llbracket \mathbf{SK} \rrbracket_{[n]}, pp)$: Pick $k_1, ..., k_d \leftarrow_\$ \{0,1\}^\kappa$ where $d := \binom{n}{n-t+1}$. Let $D_1, \ldots, D_d$ be the d distinct $(n-t+1)$-sized subsets of $[n]$. For $i \in [n]$, let $SK_i := \{k_j \mid i \in D_j \text{ for } j \in [d]\}$. Set $pp := (f)$ and give (SK_i, pp) to party i, for $i \in [n]$.
- $\mathsf{Eval}(SK_i, x, pp) \to z_i$: Compute $h_{i,k} := f_k(x)$ for all $k \in SK_i$ and output $\{h_{i,k}\}_{k \in SK_i}$.
- $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp) =: z/\perp$: If $|S| < t$, output $\perp$. Else, parse z_i as $\{h_{i,k}\}_{k \in SK_i}$ for $i \in S$. Let $\{SK'_i\}_{i \in S}$ be mutually disjoint sets such that $\cup_{i \in S} SK'_i = \{k_1, \ldots, k_d\}$ and $SK'_i \subseteq SK_i$ for every i. Output $\oplus_{k \in SK'_i, i \in S} h_{i,k}$.

Figure 7: A secure DPRF protocol $\Pi_{\mathsf{f\text{-}DP}}$ based on any PRF.

$D_1, \ldots, D_d$ subsets, say D_j, does not intersect with S; thus parties in S do not have k_j). See Figure 7 for a formal description.

Theorem 8.4 *If f is a PRF, then $\Pi_{\mathsf{f\text{-}DP}}$ in Figure 7 is a secure DPRF.*

Proof of the above theorem can be found in Appendix C.6. We also note that it is possible to augment this PRF-based construction into one that satisfies strong correctness (hence strong security) using only symmetric-key primitives. In particular, one could commit to the PRF secrets during the setup, and require that each party provides a symmetric-key NIZK of correctness of its evaluation with respect to its committed secret keys using recent techniques [GMO16, CDG+17]. We do not present such an instantiation since it would be quite inefficient.

9 Experimental Evaluation

When we combine the constructions of Section 7 and the DPRF instantiations of Section 8, we obtain four variants (two with strong security) of a threshold authenticated encryption scheme as depicted in Figure 8. We remark that although our implementation uses a hash function modeled as a random oracle to implement the commitment scheme used in DiSE the construction itself is proven secure using any commitment scheme in the standard model.

DPRF Instantiation	Resulting TSE	Assumption	Model
$\Pi_{\mathsf{f\text{-}DP}}$ (Fig. 7)	Γ_{AES}	OWF	Standard
$\Pi_{\mathsf{DDH\text{-}DP}}$ (Fig. 3)	Γ_{DDH}	DDH	Standard
$\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$ (Fig. 5)	$\Gamma^{\mathsf{S}}_{\mathsf{DDH}}$ (Strong)	DDH	ROM
$\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$ (Fig. 6)	$\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$ (Strong)	DDH	ROM

Figure 8: The four TSE schemes we implemented by instantiating DiSE. There are two concrete instantiations of $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$, depending on the verifiability feature (see Remark 8.3).

We implement all four variants of our protocol in C++. We implement the random oracle as Blake2 [bla] and the PRF/PRGs are constructed from AES-NI. The DDH-based DPRF [NPR99] uses the Miracl library [mir] with Curve p256k1. Benchmarks were performed on a single server equipped with two 18-core Intel Xeon CPUs at 2.3Ghz and 256GB of RAM. Parties communicate through a kernel loopback device simulating two settings: LAN - 10 Gbps and 0.1ms (RTT) latency, WAN: *shared* 40 Mbps and 80ms latency.

t	n	Throughput (enc/s)				Latency (ms/enc)				Bandwidth (Throughput $Mbps$)			
		Γ_{AES}	Γ_{DDH}	$\Gamma^{\mathsf{S}}_{\mathsf{DDH}}$	$\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$	Γ_{AES}	Γ_{DDH}	$\Gamma^{\mathsf{S}}_{\mathsf{DDH}}$	$\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$	Γ_{AES}	Γ_{DDH}	$\Gamma^{\mathsf{S}}_{\mathsf{DDH}}$	$\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$
$n/3$	6	1,095,770	556	232	189	0.1	4.6	9.3	10.5	268	0.28	0.29	0.28
	12	656,728	382	99	77	0.3	4.4	15.9	18.8	481	0.53	0.37	0.35
	18	45,434	297	64	46	0.6	5.4	21.5	27.6	55	0.77	0.40	0.35
	24	902	173	34	31	7.5	11.2	36.4	43.1	2	0.69	0.30	0.34
$n/2$	4	1,113,090	555	235	190	0.1	4.7	9.2	10.1	272	0.13	0.30	0.29
	6	510,152	527	146	112	0.2	4.0	11.9	14.3	249	0.26	0.44	0.34
	12	198,020	300	64	48	0.7	5.2	21.2	26.1	242	0.37	0.47	0.36
	18	10,194	231	42	31	1.1	8.0	31.3	38.8	20	0.45	0.50	0.37
	24	165	125	22	22	38.0	15.9	54.5	69.6	0	0.33	0.38	0.36
$2n/3$	3	1,100,413	561	239	190	0.1	3.9	10.7	18.6	269	0.14	0.30	0.29
	6	1,033,592	399	101	75	0.4	4.2	15.0	18.6	757	0.29	0.38	0.34
	12	438,957	245	47	35	1.1	6.4	27.6	34.5	750	0.42	0.49	0.36
	18	21,139	176	31	21	1.6	8.9	41.6	51.5	57	0.47	0.43	0.35
	24	445	100	17	16	16.5	21.5	72.4	85.0	2	0.37	0.32	0.36
$n-2$	12	727,273	203	37	34	1.4	7.37	33.1	44.7	1598	0.45	0.42	0.36
	18	524,109	135	23	17	2.2	12.6	55.2	66.2	1919	0.49	0.43	0.38
	24	283,822	75	12	10	5.6	28.0	98.9	116.4	1455	0.38	0.32	0.31
2	12	1,058,574	556	235	189	0.1	4.6	9.56	10.0	258	0.14	0.30	0.29
	18	1,037,703	553	226	188	0.1	4.6	9.6	10.3	253	0.14	0.28	0.28
	24	735,294	404	176	151	2.2	4.6	9.56	10.4	180	0.10	0.22	0.23

Figure 9: Encryption performance metrics for 10 second trials of 32 bytes messages in the LAN setting with various number of parties n and threshold t. Throughput is computed by performing many encryptions concurrently (single thread per party). Latency is computed by performing sequential encryptions. Bandwidth is total (send + receive) bandwidth consumed at peak throughput.

t	n	Throughput (enc/s)				Latency (ms/enc)				Bandwidth (Throughput $Mbps$)			
		Γ_{AES}	Γ_{DDH}	$\Gamma^{\mathsf{S}}_{\mathsf{DDH}}$	$\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$	Γ_{AES}	Γ_{DDH}	$\Gamma^{\mathsf{S}}_{\mathsf{DDH}}$	$\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$	Γ_{AES}	Γ_{DDH}	$\Gamma^{\mathsf{S}}_{\mathsf{DDH}}$	$\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$
$\frac{n}{3}$	6	153,332	570	238	190	81	86	96	101	37	0.14	0.30	0.29
	12	51,745	399	103	76	81	88	111	117	38	0.29	0.39	0.34
	18	31,096	303	65	46	81	90	125	139	38	0.37	0.41	0.35
	24	775	191	36	26	86	90	132	146	1	0.33	0.31	0.27
$\frac{n}{2}$	4	150,783	571	239	188	81	86	96	104	37	0.14	0.30	0.28
	6	76,957	536	150	112	81	86	103	111	38	0.26	0.38	0.34
	12	30,937	297	65	48	82	90	125	131	38	0.36	0.41	0.36
	18	11,776	235	42	31	82	92	141	145	23	0.46	0.43	0.37
	24	166	132	24	18	102	96	146	149	0	0.36	0.33	0.30
$\frac{2n}{3}$	3	150,965	555	238	189	81	86	97	105	37	0.14	0.30	0.28
	6	51,535	396	103	77	81	88	112	122	38	0.29	0.39	0.34
	12	21,484	244	45	35	81	93	123	152	37	0.42	0.40	0.37
	18	14,029	174	31	22	82	97	156	169	38	0.47	0.43	0.37
	24	446	101	17	13	92	98	164	172	2	0.37	0.33	0.28

Figure 10: Encryption performance metrics for 10 second trials of 32 bytes messages in the WAN setting (shared send+receive 40Mbps, 80ms RTT) with various number of parties n and threshold t.

Throughput. Figure 9 shows the throughput and latency of our protocols under a variety of configurations in the LAN setting. Throughput measures the maximum number of operations that can be performed given that each party has a *single core*. Throughput is an important metric for many tasks such as a key/token server or per row database decryption.

The Γ_{AES} protocol is the fastest by a large margin for all $n \leq 24$ despite having exponential overhead in the number of parties. For instance, encrypting 32 bytes with $n = 6$ and $t = 4$, Γ_{AES} achieves 1 million encryption per second while Γ_{DDH}, the next fastest, is $2000\times$ slower with 556 encryptions. Increasing the parameters to $n = 24, t = 16$, Γ_{AES} achieves 902

encryptions per second while Γ_{DDH} is still 5× slower with 173 encryptions. The protocol $\Gamma^{\mathsf{S}}_{\mathsf{DDH}}$ which achieves strong correctness incurs a 2 to 5× overhead compared to the weaker Γ_{DDH} while the publicly verifiable variant $\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$ has, on average, 25% lower throughput.

Latency. Another important metric is latency. That is, the time from the start of an encryption/decryption until the result is ready. Due to various system level optimization for improved latency, the throughput and latency results shown in Figure 9 are for different configurations of the protocol, e.g. less vectorization which improves latency at the cost of a smaller throughput. Γ_{AES} achieves sub-millisecond latency for most configurations. On the other hand, $\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$ with its strong security guarantees achieves a latency between 10 and 100ms.

Communication. In addition to achieving the best throughput and latency, the Γ_{AES} protocol has the smallest communication overhead of $32(t-1)$ bytes per encryption. The Γ_{DDH} incurs slightly more communication with $49(t-1)$ bytes per encryption while $\Gamma^{\mathsf{S}}_{\mathsf{DDH}}$ and $\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$ have the most communication with $148(t-1)$ bytes. However, despite having comparable communication overheads, the pure symmetric-key Γ_{AES} protocol is significantly faster for small n due to the use of much more efficient AES operations (in contrast to exponentiations).

Key Size. The primary advantage of the DDH-based protocols Γ_{DDH}, $\Gamma^{\mathsf{S}}_{\mathsf{DDH}}$ and $\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$ is that the key size is either constant (33 bytes) or linear in the threshold ($33t$ bytes). The Γ_{AES} protocol, on the other hand, requires that each party hold roughly $\binom{n}{t} \approx O(n^{\min(t,n-t)})$ keys. As such, the single benchmark machine sharing 256GB of RAM was not able to handle significantly more than 24 parties. For instance, with $n = 6, t = 4$ each party must hold 80 bytes of key while the case of $n = 24, t = 16$ requires each party to hold a 8MB key. In the worst case of $t = n/2$ with $n = 24$, the key size increases to 22MB per party. However, despite this exponential blowup, the Γ_{AES} can gracefully handle cases where n is small or the threshold t is near 2 or n as shown in the bottom half of Figure 9.

WAN Performance. To measure the performance of the protocols over the Internet, we benchmark on a (simulated) network with a *shared* bandwidth of 40 Mbps and an 80ms round-trip time.

As shown in Figure 10, the bandwidth restriction limits the throughput of the Γ_{AES} protocol due to it easily saturating the network. With $n = 6, t = 2$, we observe that the throughput drops 7× to 153,332 encryption per second. However, this is near optimal given that simply communicating κ bits requires 37 out of the 40Mbps bandwidth limit. Additionally, the latency of the Γ_{AES} is near the optimal of 80ms in most cases. The Γ_{DDH} protocol require slightly more time of roughly 90ms in most cases while the strongly-correct $\Gamma^{\mathsf{PV}}_{\mathsf{DDH}}$ and $\Gamma^{\mathsf{S}}_{\mathsf{DDH}}$ protocols require between 95 and 170ms.

10 Acknowledgment

We thank Saikrishna Badrinarayanan, Dan Boneh, Atul Luykx and anonymous CCS 2018 reviewers for helpful comments on earlier drafts of this paper.

References

[AMN01] Michel Abdalla, Sara Miner, and Chanathip Namprempre. Forward-secure threshold signature schemes. In *Cryptographers Track at the RSA Conference*, pages 441–456. Springer, 2001.

[BBH06] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 226–243. Springer, Heidelberg, February 2006.

[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.

[BGM04] Mihir Bellare, Oded Goldreich, and Anton Mityagin. The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, Report 2004/309, 2004. http://eprint.iacr.org/2004/309.

[BHT18] Priyanka Bose, Viet Tung Hoang, and Stefano Tessaro. Revisiting AES-GCM-SIV: multi-user security, faster key derivation, and better bounds. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, pages 468–499, 2018.

[BK11] Mihir Bellare and Sriram Keelveedhi. Authenticated and misuse-resistant encryption of key-dependent data. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 610–629. Springer, Heidelberg, August 2011.

[bla] MIRACL Cryptographic SDK. https://www.miracl.com/.

[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.

[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.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Heidelberg, January 2003.

[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.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1825–1842, 2017.

[CG99] Ran Canetti and Shafi Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 90–106. Springer, Heidelberg, May 1999.

[CV90] David Chaum and Hans Van Antwerpen. Undeniable signatures. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 212–216. Springer, Heidelberg, August 1990.

[DDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *26th ACM STOC*, pages 522–533. ACM Press, May 1994.

[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.

[DK01] Ivan Damgård and Maciej Koprowski. Practical threshold RSA signatures without a trusted dealer. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 152–165. Springer, Heidelberg, May 2001.

[DK10] Ivan Damgård and Marcel Keller. Secure multiparty AES. In Radu Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 367–374. Springer, Heidelberg, January 2010.

[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.

[DP08] Cécile Delerablée and David Pointcheval. Dynamic threshold public-key encryption. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 317–334. Springer, Heidelberg, August 2008.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, January 2005.

[dya] Dyadic Security. https://www.dyadicsec.com.

[DYY06] Yevgeniy Dodis, Aleksandr Yampolskiy, and Moti Yung. Threshold and proactive pseudo-random permutations. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 542–560. Springer, Heidelberg, March 2006.

[EC:]

[ECS+15] Adam Everspaugh, Rahul Chaterjee, Samuel Scott, Ari Juels, and Thomas Ristenpart. The pythia PRF service. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 547–562, 2015.

[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.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Heidelberg, December 2012.

[Fra90] Yair Frankel. A practical protocol for large group oriented networks. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 56–61. Springer, Heidelberg, April 1990.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 74–94. Springer, Heidelberg, February 2014.

[GHKR08] Rosario Gennaro, Shai Halevi, Hugo Krawczyk, and Tal Rabin. Threshold RSA for dynamic and ad-hoc groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 88–107. Springer, Heidelberg, April 2008.

[GJKR96] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 354–371. Springer, Heidelberg, May 1996.

[GL15] Shay Gueron and Yehuda Lindell. GCM-SIV: Full nonce misuse-resistant authenticated encryption at under one cycle per byte. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 109–119. ACM Press, October 2015.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security Symposium*, pages 1069–1083, 2016.

[GMPP16] Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 448–476. Springer, Heidelberg, May 2016.

[GRR+16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 430–443. ACM Press, October 2016.

[GS17] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 588–599. IEEE Computer Society, 2017.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 468–499. Springer, 2018.

[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.

[HRRV15] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. Online authenticated-encryption and its nonce-reuse misuse-resistance. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 493–517. Springer, Heidelberg, August 2015.

[kera] Kerberos: Encryption Types. http://web.mit.edu/kerberos/krb5-latest/doc/admin/enctypes.html.

[kerb] Kerberos: The Network Authentication Protocol. http://web.mit.edu/kerberos/.

[KT09] R. Kusters and M. Tuengerthal. Universally composable symmetric encryption. In *2009 22nd IEEE Computer Security Foundations Symposium*, pages 293–307, July 2009.

[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.

[mir] BLAKE2 - fast secure hashing. https://blake2.net/.

[mpca] ePrint Archive MPC Papers. http://users-cs.au.dk/psn/list/.

[mpcb] List of MPC Software. http://www.multipartycomputation.com/mpc-software.

[mpcc] List of resources on MPC. https://github.com/rdragos/awesome-mpc.

[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.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[Nie02] Jesper Buus Nielsen. A threshold pseudorandom function construction and its applications. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 401–416. Springer, Heidelberg, August 2002.

[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.

[por] Porticor Cloud Security. http://www.porticor.com/. Acquired by Intuit.

[PW12] Kenneth G. Paterson and Gaven J. Watson. Authenticated-encryption with padding: A formal security treatment. In *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, pages 83–107, 2012.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 02*, pages 98–107. ACM Press, November 2002.

[Rog13] Philip Rogaway. The Evolution of Authenticated Encryption. https://crypto.stanford.edu/RealWorldCrypto/slides/phil.pdf, 2013.

[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.

[RSS17] Dragos Rotaru, Nigel P. Smart, and Martijn Stam. Modes of operation suitable for computing on encrypted data. Cryptology ePrint Archive, Report 2017/496, 2017. http://eprint.iacr.org/2017/496.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

[sec] Infrastructure Secret Management Software Overview. https://gist.github.com/maxvt/bb49a6c7243163b8120625fc8ae3f3cd.

[sep] Sepior. https://sepior.com.

[SG98] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 1–16. Springer, Heidelberg, May / June 1998.

[SG02] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, 2002.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[vaua] Vault Architecture. https://www.vaultproject.io/docs/internals/architecture.html.

[vaub] Vault by HashiCorp. https://www.vaultproject.io/.

[vauc] Vault Docs, Basic Concepts, Seal-Unseal. https://www.vaultproject.io/docs/concepts/seal.html.

A Cryptographic Primitives

We include definitions of some well-known primitives for completeness.

A.1 Authenticated Encryption

Definition A.1 (Symmetric encryption) *A symmetric encryption scheme is a triple of polynomial-time algorithms* $(\mathsf{Kgen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *that satisfy a correctness requirement.*

- $\mathsf{Kgen}(1^\kappa) \to sk$: *On input the security parameter,* Kgen *outputs a secret key* sk.
- $\mathsf{Encrypt}(sk, m) \to c$: *On input the secret key* sk *and a message* m, Encrypt *outputs a ciphertext* c.
- $\mathsf{Decrypt}(sk, c) =: m/\perp$: *On input the secret key* sk *and a ciphertext* c, Decrypt *outputs a message* m *or a failure symbol* $\perp$.

Correctness. For all $\kappa \in \mathbb{N}$, sk *output by* $\mathsf{Kgen}(1^\kappa)$, *and any message* m, *there exists a negligible function* negl *such that*

$$\Pr\left[m := \mathsf{Decrypt}(sk, c) \ : \ c \leftarrow \mathsf{Encrypt}(sk, m)\right] \geq 1 - \mathsf{negl}(\kappa),$$

where the probability is over the randomness of Encrypt.

Chosen-plaintext attack. A symmetric encryption scheme $\Pi = (\mathsf{Kgen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is secure against chosen-plaintext attacks (CPA) if for all PPT adversary $\mathcal{A}$, there exists a negligible function negl such that

$$\left|\Pr\left[\mathsf{SymCPA}_{\Pi,\mathcal{A}}(1^\kappa) = 1\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\kappa),$$

where SymCPA is defined as follows:

1. *Initialize.* Run Kgen to get a key sk.
2. *Pre-challenge encryption queries.* On receiving $(\texttt{Encrypt}, m)$ from $\mathcal{A}$, return $c \leftarrow \mathsf{Encrypt}(sk, m)$. This step can be repeated any number of times.
3. *Challenge.* When $\mathcal{A}$ sends $(\texttt{Challenge}, (m_0, m_1))$ such that $|m_0| = |m_1|$, choose a random bit $b \leftarrow_\$ \{0, 1\}$ and return $c^\star \leftarrow \mathsf{Encrypt}(sk, m_b)$.
4. *Post-challenge encryption queries.* Same as Step 2.
5. *Guess.* Finally, receive a guess b' from $\mathcal{A}$ and output 1 if and only if $b' = b$.

Authenticity [BN00, KY01, RS06]. A symmetric encryption scheme $\Pi = (\mathsf{Kgen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ satisfies authenticity if for all PPT adversary $\mathcal{A}$, there exists a negligible function negl such that

$$\Pr\left[\mathsf{SymAUTH}_{\Pi,\mathcal{A}}(1^\kappa) = 1\right] \leq \mathsf{negl}(\kappa),$$

where $\mathsf{SymAUTH}$ is defined as follows:

1. *Initialize.* Run Kgen to get a key sk.
2. *Encryption queries.* On receiving $(\mathtt{Encrypt}, m)$ from $\mathcal{A}$, return $c \leftarrow \mathsf{Encrypt}(sk, m)$. This step can be repeated any number of times.
3. *Forgery.* $\mathcal{A}$ produces a ciphertext $c^\star$. Output 1 if and only if $\mathsf{Decrypt}(sk, c^\star) \neq \perp$.

In the following, we use $\mathsf{Encrypt}_{sk}(\cdot)$ and $\mathsf{Decrypt}_{sk}(\cdot)$ to mean $\mathsf{Encrypt}(sk, \cdot)$ and $\mathsf{Decrypt}(sk, \cdot)$, respectively.

A.2 Commitment

Definition A.2 *A (non-interactive) commitment scheme Σ consists of two PPT algorithms $(\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ which satisfy hiding and binding properties:*

- $\mathsf{Setup}_{\mathsf{com}}(1^\kappa) \to pp_{\mathsf{com}}$: *It takes the security parameter as input, and outputs some public parameters.*
- $\mathsf{Com}(m, pp_{\mathsf{com}}; r) =: \alpha$: *It takes a message m, public parameters pp_{com} and randomness r as inputs, and outputs a commitment α.*

Hiding. A commitment scheme $\Sigma = (\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ is hiding if for all PPT adversaries $\mathcal{A}$, all messages m_0, m_1, there exists a negligible function negl such that for $pp_{\mathsf{com}} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^\kappa)$,

$$|\Pr[\mathcal{A}(pp_{\mathsf{com}}, \mathsf{Com}(m_0, pp_{\mathsf{com}}; r_0)) = 1] - \Pr[\mathcal{A}(pp_{\mathsf{com}}, \mathsf{Com}(m_1, pp_{\mathsf{com}}; r_1)) = 1]| \leq \mathsf{negl}(\kappa),$$

where the probability is over the randomness of $\mathsf{Setup}_{\mathsf{com}}$, random choice of r_0 and r_1, and the coin tosses of $\mathcal{A}$.

Binding. A commitment scheme $\Sigma = (\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ is binding if for all PPT adversaries $\mathcal{A}$, if $\mathcal{A}$ outputs m_0, m_1, r_0 and r_1 $((m_0, r_0) \neq (m_1, r_1))$ given $pp_{\mathsf{com}} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^\kappa)$, then there exists a negligible function negl such that

$$\Pr[\mathsf{Com}(m_0, pp_{\mathsf{com}}; r_0) = \mathsf{Com}(m_1, pp_{\mathsf{com}}; r_1)] \leq \mathsf{negl}(\kappa),$$

where the probability is over the randomness of $\mathsf{Setup}_{\mathsf{com}}$ and the coin tosses of $\mathcal{A}$.

Definition A.3 (Trapdoor (Non-interactive) Commitments.) *Let $\mathsf{Com} = (\mathsf{Setup}_{\mathsf{com}}, \mathsf{Com})$ be a (non-interactive) commitment scheme. A trapdoor commitment scheme has two more PPT algorithms $\mathsf{SimSetup}$ and $\mathsf{SimOpen}$:*

- $\mathsf{SimSetup}(1^\kappa) \to (pp_{\mathsf{com}}, \tau_{\mathsf{com}})$: *It takes the security parameter as input, and outputs public parameters pp_{com} and a trapdoor τ_{com}.*

– $\mathsf{SimOpen}(pp_{\mathsf{com}}, \tau_{\mathsf{com}}, m', (m, r)) =: r'$: *It takes the public parameters* pp_{com}*, the trapdoor* τ_{com}*, a message* m' *and a message-randomness pair* (m, r)*, and outputs a randomness* r'*.*

For every (m, r) *and* m'*, there exists a negligible function* negl *such that*

$$pp_{\mathsf{com}} \approx_{stat} pp'_{\mathsf{com}} \qquad \textit{where } pp_{\mathsf{com}} \leftarrow \mathsf{Setup}_{\mathsf{com}}(1^\kappa) \textit{ and } (pp'_{\mathsf{com}}, \tau_{\mathsf{com}}) \leftarrow \mathsf{SimSetup}(1^\kappa)$$

and

$$\Pr\left[\mathsf{Com}(m, pp'_{\mathsf{com}}; r) = \mathsf{Com}(m', pp'_{\mathsf{com}}; r') \mid \begin{array}{c}(pp'_{\mathsf{com}}, \tau_{\mathsf{com}}) \leftarrow \mathsf{SimSetup}(1^\kappa);\\ r' := \mathsf{SimOpen}(pp'_{\mathsf{com}}, \tau_{\mathsf{com}}, m', (m, r))\end{array}\right] \geq 1 - \mathsf{negl}(\kappa).$$

Remark A.4 *Clearly, a trapdoor commitment can be binding against PPT adversaries only.*

A.2.1 Concrete instantiations.

Practical commitment schemes can be instantiated under various settings:

Random oracle. In the random oracle model, a commitment to a message m is simply the hash of m together with a randomly chosen string of length r of an appropriate length.

DLOG assumption. A popular commitment scheme secure under DLOG is Pedersen commitment. Here, $\mathsf{Setup}_{\mathsf{com}}(1^\kappa)$ outputs the description of a (multiplicative) group G of prime order $p = \Theta(\kappa)$ (in which DLOG holds) and two randomly and independently chosen generators g, h. If $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$ is a collision-resistant hash function, then a commitment to a message m is given by $g^{\mathcal{H}(m)} \cdot h^r$, where $r \leftarrow_\$ \mathbb{Z}_p$. A trapdoor is simply the discrete log of h with respect to g. In other words, $\mathsf{SimSetup}$ picks a random generator g, a random integer a in $\mathbb{Z}_p^\star$ and sets h to be g^a. Given (m, r), m' and a, $\mathsf{SimOpen}$ outputs $[(\mathcal{H}(m) - \mathcal{H}(m'))/a] + r$. It is easy to check that commitment to m with randomness r is equal to the commitment to m' with randomness r'.

Pseudo-random generators. Naor proposed a simple and efficient commitment scheme based on the existence of PRGs [Nao91]. We briefly describe here a non-interactive variant of the scheme for commitment to n-bit strings. $\mathsf{Setup}_{\mathsf{com}}$ outputs a randomly chosen string crs of length $4n$. Let $pad : \{0,1\}^n \to \{0,1\}^{4n}$ be the function that prepends $3n$ zeroes to its argument, $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ be a collision-resistant hash function, and $G : \{0,1\}^n \to \{0,1\}^{4n}$ be a pseudo-random generator. Then, a commitment to a message m is given by $G(r) + crs \cdot pad(\mathcal{H}(x))$ with arithmetic in $GF(2^{4n})$, where $r \leftarrow_\$ \{0,1\}^n$. We skip rest of the details.

A.3 Secret Sharing

Definition A.5 (Shamir's Secret Sharing) *Let* p *be a prime. An* (n, t, p, s)*-Shamir's secret sharing scheme is a randomized algorithm* SSS *that on input four integers* n, t, p, s*, where* $0 < t \leq n < p$ *and* $s \in \mathbb{Z}_p$*, outputs* n *shares* $s_1, \ldots, s_n \in \mathbb{Z}_p$ *such that the following two conditions hold for any set* $\{i_1, \ldots, i_\ell\}$*:*

– *if* $\ell \geq t$*, there exists fixed (i.e., independent of* s*) integers* $\lambda_1, \ldots, \lambda_\ell \in \mathbb{Z}_p$ *(a.k.a. Lagrange coefficients) such that* $\sum_{j=1}^{\ell} \lambda_j s_{i_j} = s \mod p$*;*

– *if $\ell < t$, the distribution of $(s_{i_1}, \ldots, s_{i_\ell})$ is uniformly random.*

Concretely, Shamir's secret sharing works as follows. Pick $a_1, \ldots, a_{t-1} \leftarrow_\$ \mathbb{Z}_p$. Let $f(x)$ be the polynomial $s + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_{t-1} \cdot x^{t-1}$. Then s_i is set to be $f(i)$ for all $i \in [n]$.

A.4 Non-interactive Zero-knowledge

Let R be an efficiently computable binary relation. For pairs $(s, w) \in R$, we refer to s as the statement and w as the witness. Let L be the language of statements in R, i.e. $L = \{s : \exists w \text{ such that } R(s, w) = 1\}$. We define non-interactive zero-knowledge arguments of knowledge in the random oracle model based on the work of Faust et al. [FKMV12].

Definition A.6 (Non-interactive Zero-knowledge Argument of Knowledge) *Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{\mathsf{poly}(\kappa)}$ be a hash function modeled as a random oracle. A* NIZK *for a binary relation R consists of two PPT algorithms* Prove *and* Verify *with oracle access to $\mathcal{H}$ defined as follows:*

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

– $\mathsf{Verify}^{\mathcal{H}}(s, \pi)$ *takes as input a statement s and a candidate proof π, and outputs a bit $b \in \{0, 1\}$ denoting acceptance or rejection.*

These two algorithms must satisfy the following properties:

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

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

– ***Zero-knowledge**: There must exist a pair of PPT simulators $(\mathcal{S}_1, \mathcal{S}_2)$ such that for all PPT adversary $\mathcal{A}$,*

$$\left|\Pr[\mathcal{A}^{\mathcal{H}, \mathsf{Prove}^{\mathcal{H}}}(1^\kappa) = 1] - \Pr[\mathcal{A}^{\mathcal{S}_1(\cdot), \mathcal{S}_2'(\cdot,\cdot)}(1^\kappa) = 1]\right| \leq \mathsf{negl}(\kappa)$$

for some negligible function negl*, where*

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

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

$$\Pr\Big[(s, w) \notin R \text{ and } \mathsf{Verify}^{\mathcal{H}}(s, \pi) = 1 \mid (s, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_1(\cdot)}(1^\kappa); w \leftarrow \mathcal{E}^{\mathcal{A}}(s, \pi, Q)\Big] \leq \mathsf{negl}(\kappa)$$

for some negligible function negl*, where*

  – *$\mathcal{S}_1$ is like above;*
  – *Q is the list of (query, response) pairs obtained from $\mathcal{S}_1$.*

Fiat-Shamir transform. Let $(\mathsf{Prove}, \mathsf{Verify})$ be a three-round public-coin honest-verifier zero-knowledge interactive proof system (a sigma protocol) with unique responses. Let $\mathcal{H}$ be a function with range equal to the space of the verifier's coins. In the random oracle model, the proof system $(\mathsf{Prove}^{\mathcal{H}}, \mathsf{Verify}^{\mathcal{H}})$ derived from $(\mathsf{Prove}, \mathsf{Verify})$ by applying the Fiat-Shamir transform satisfies the zero-knowledge and argument of knowledge properties defined above. See Definition 1, 2 and Theorem 1, 3 in Faust et al. [FKMV12] for more details. (They actually show that these properties hold even when adversary can ask for proofs of false statements.)

B A few failed attempts in detail

B.1 Attempt 1: Distributed Encryption Scheme proposed by Naor et al.

The work of Naor et al. [NPR99], which puts forward the DPRF constructions we use in this paper, also proposes the only distributed symmetric-key encryption proposal we are aware of. Their proposal appeared before any formal treatment of threshold symmetric-key existed, and in fact even prior to the introduction of authenticated encryption in the non-distributed setting. Hence, it is only natural that their scheme does not meet the strong security notions we introduce here. We review their protocol in Figure 11 and argue why it fails to meet the security definitions introduced in this paper.[10]

(In)-security of Π_{NPR}. The protocol Π_{NPR} is CPA-secure against malicious adversaries if the underlying DPRF satisfies weak-malicious security. The formal proof is similar to the proof of the CPA-security of our protocol (c.f. Theorem 7.4) and hence we omit it. However, we argue that Π_{NPR} does not satisfy authenticity (c.f. Definition 6.8) even when the corrupt parties follow the protocol. To see this, first observe that a ciphertext in this protocol is a tuple $c := (j, k, e)$ where $k = y \oplus w$, $y := \mathsf{DP}(j\|e)$ and $e \leftarrow \mathsf{sENC.Encrypt}_w(m)$ for some message m. After obtaining this ciphertext, an adversary can go "off-line" and compute another ciphertext $c' = (j, k', e)$ which now decrypts to $m' := \mathsf{sENC.Decrypt}_{w'}(e)$ where $w' = k' \oplus y$. This is possible since CPA-security does not guarantee that a ciphertext c can not be decrypted with another secret-key w' (although the message m' will not reveal any information about m by CPA-security). So the adversary could produce many valid ciphertexts by running just one encryption session, and thus win the authenticity game. (If we consider adversaries that choose their own w, then it becomes even harder to make the scheme secure.)

The authors also mention that "in order [to] combat changes to the stored information one should use parts of y as an authentication key to e and w" (symbols in the quote have been replaced with the equivalent ones here). As we interpret, one can additionally use a message-authentication code (MAC) so that the ciphertext would look like $c := (j, k, e, t)$ where $k := w \oplus y_1$, $t = \mathsf{MAC}_{y_2}(e||w)$ and $(y_1||y_2) := \mathsf{DP}(j\|e)$. However, the modified construction still does not satisfy the authenticity requirement (c.f. Def. 6.8) because once an adversary gets the MAC key y_2 through an encryption session, it can re-launch the same attack, this time attaching a correct MAC computed with y_2.

[10] We present a slightly different version here that does not use a collision-resistant hash function or a decryption policy, but incorporates the identity of the initiating party in computing the ciphertext. This does not interfere in any way with the security analysis we carry out.

Ingredients:

- An (n,t)-DPRF protocol $\mathsf{DP} := (\mathsf{DP.Setup}, \mathsf{DP.Eval}, \mathsf{DP.Combine})$.
- A CPA secure symmetric-key encryption scheme: $\mathsf{sENC} := (\mathsf{sENC.Kgen}, \mathsf{sENC.Encrypt}, \mathsf{sENC.Decrypt})$.

$\mathsf{Setup}(1^\kappa, n, t) \to \left([\![\mathbf{sk}]\!]_{[n]}, pp\right)$: Run $\mathsf{DP.Setup}(1^\kappa, n, t)$ to get $((rk_1, \ldots, rk_n), pp')$. Set $sk_i := rk_i$ for $i \in [n]$ and $pp := pp'$.

$\mathsf{DistEnc}([\![\mathbf{sk}]\!]_{[n]}, [j : m, S], pp) \to [j : c/\perp]$: To encrypt a message m with the help of parties in S:

- Party j samples $w \leftarrow \mathsf{sENC.Keygen}(1^\kappa)$ and computes $e \leftarrow \mathsf{sENC.Encrypt}_w(m)$. Then it sends e to all parties in S.
- For every $i \in S$, party i runs $\mathsf{DP.Eval}(sk_i, j\|e, pp)$ to get y_i, and sends it to party j.
- Party j runs $\mathsf{Combine}(\{(i, y_i)\}_{i \in S}, pp)$ to get y or $\perp$. In the latter case, it outputs $\perp$. Otherwise, it outputs $c := (j, k, e)$ where $k := y \oplus w$.

$\mathsf{DistDec}([\![\mathbf{sk}]\!]_{[n]}, [j' : c, S], pp) \to [j' : m/\perp]$: To decrypt a ciphertext c with the help of parties in S:

- Party j' first parses c into (j, k, e). Then it sends $j\|e$ to all the parties in S.
- For $i \in S$, party i receives x and checks if it is of the form $j^\star\|e^\star$ for some $j^\star \in [n]$. If not, then it sends $\perp$ to party j'. Else, it runs $\mathsf{DP.Eval}(sk_i, x, pp)$ to get y_i, and sends it to party j'.
- Party j' runs $\mathsf{Combine}(\{(i, y_i)\}_{i \in S}, pp)$ to get y or $\perp$. In the latter case, it outputs $\perp$. Otherwise, it computes $w := k \oplus y$ and outputs $m := \mathsf{sENC.Decrypt}_w(e)$.

Figure 11: Description of the protocol Π_{NPR}

B.2 Attempt 2: DPRF + Authenticated Encryption

Another natural proposal for encryption is to generate a fresh pseudorandom key using a DPRF and use it to encrypt the message via a symmetric-key authenticated encryption scheme. This was our first attempt for a secure construction. It is helpful to review this construction (Fig. 12) and show why it fails to meet our notion of authenticity, even when the corrupt parties do not deviate from the protocol.

To see why the above scheme does not meet our authenticity notion, consider an attacker who runs a single distributed encryption session to learn an encryption key w and uses it to encrypt many messages "off-line", thereby generating many new valid ciphertexts.

C Missing Proofs

C.1 Proof of Theorem 7.4

We use a sequence of hybrids to prove security, with the first hybrid being the message privacy game $\mathsf{MsgPriv}$. We first write down the challenge phase of $\mathsf{MsgPriv}$ in detail:

1. $\mathcal{A}$ outputs $(\texttt{Challenge}, j^\star, m_0, m_1, S^\star)$ where $j \in S^\star \setminus C$ and $|S^\star| \geq t$.
2. Compute $\alpha^\star := \mathsf{Com}(m_b, pp_{\mathsf{com}}; \rho^\star)$ by picking $\rho^\star$ at random, and $z_i \leftarrow \mathsf{DP.Eval}(sk_i, (j^\star\|\alpha^\star), pp)$ for every $i \in S^\star \setminus C$.
3. Send $\alpha^\star$ to $\mathcal{A}$ and get back $\hat{z}_i$ for every $i \in S^\star \cap C$.

Ingredients:

- An (n,t)-DPRF protocol $\mathsf{DP} := (\mathsf{DP.Setup}, \mathsf{DP.Eval}, \mathsf{DP.Combine})$.
- An authenticated encryption scheme $\mathsf{auENC} := (\mathsf{auENC.Kgen}, \mathsf{auENC.Encrypt}, \mathsf{auENC.Decrypt})$.

$\mathsf{Setup}(1^\kappa, n, t) \to \left(\llbracket \mathbf{sk} \rrbracket_{[n]}, pp\right)$: Run $\mathsf{DP.Setup}(1^\kappa, n, t)$ to get $((rk_1, \ldots, rk_n), pp')$. Set $sk_i := rk_i$ for $i \in [n]$ and $pp := pp'$.

$\mathsf{DistEnc}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j : m, S], pp) \to [j : c/\perp]$: To encrypt a message m with the help of parties in S:

- Party j samples a random r and sends it to all parties in S.
- For every $i \in S$, party i runs $\mathsf{DP.Eval}(sk_i, j\|r, pp)$ to get w_i, and sends it to party j.
- Party j runs $\mathsf{Combine}(\{(i, w_i)\}_{i \in S}, pp)$ to get w or $\perp$. In the latter case, it outputs $\perp$. Otherwise, it computes $e := \mathsf{auENC.Encrypt}_w(m)$ and outputs $c := (j, r, e)$.

$\mathsf{DistDec}(\llbracket \mathbf{sk} \rrbracket_{[n]}, [j' : c, S], pp) \to [j' : m/\perp]$: To decrypt a ciphertext c with the help of parties in S:

- Party j' first parses c into (j, r, e). Then it sends $j\|r$ to all the parties in S.
- For $i \in S$, party i receives x and checks if it is of the form $j^\star\|\alpha^\star$ for some $j^\star \in [n]$. If not, then it sends $\perp$ to party j'. Else, it runs $\mathsf{DP.Eval}(sk_i, x, pp)$ to get w_i, and sends it to party j'.
- Party j' runs $\mathsf{Combine}(\{(i, w_i)\}_{i \in S}, pp)$ to get w or $\perp$. If it obtains w, it outputs $m := \mathsf{auENC.Decrypt}_w(e)$; else, it outputs $\perp$.

Figure 12: Description of the protocol $\Pi_{\mathsf{DP+AE}}$

4. Run $\mathsf{Combine}(\{(i, z_i)\}_{i \in S^\star \setminus C} \cup \{(i, \hat{z}_i)\}_{i \in S^\star \cap C}, pp)$ to get $w^\star$ or $\perp$. In the latter case, give $\perp$ to $\mathcal{A}$. Otherwise, compute $e^\star := \mathsf{PRG}(w^\star) \oplus (m_b\|\rho^\star)$ and give $c^\star := (j^\star, \alpha^\star, e^\star)$ to $\mathcal{A}$.

We define five hybrids. With every hybrid, we only mention the difference from the previous hybrid (consider $\mathsf{MsgPriv}$ to be Hyb_0).

- Hyb_1: Whenever an honest party initiates an encryption session (including the challenge phase) and generates a commitment that is not unique (among all the commitments generated so far by honest parties), notify the adversary and stop.
- Hyb_2: In Step 4 (see breakdown of challenge phase above), use a randomly chosen $w^\star$ to compute $e^\star$ (instead of the one obtained through $\mathsf{Combine}$).
- Hyb_3: Replace $\mathsf{PRG}(w^\star)$ with a randomly chosen string of the same length.
- Hyb_4: Set $e^\star$ to be a random string of length $|m_b| + \kappa$.
- Hyb_5: Set $\alpha^\star$ to be a commitment to a random message.

Except with negligible probability, the commitments generated by honest parties are all unique due to the binding property of Σ (a fresh ρ is chosen every time). Thus $\mathsf{MsgPriv}$ is indistinguishable from Hyb_1. Indistinguishability of Hyb_2, Hyb_3 and Hyb_4 is easy to see. Further, Hyb_4 and Hyb_5 are indistinguishable due to the hiding property of Σ.

We are only left to show that Hyb_1 and Hyb_2 are indistinguishable. On the contrary, suppose there exists a PPT adversary $\mathcal{A}$ who can distinguish between Hyb_1 and Hyb_2 with

a non-negligible probability. We use $\mathcal{A}$ to build another PPT adversary $\mathcal{B}$ who succeeds in $\mathsf{PseudoRand}$ for the DPRF DP (Definition 5.3) with the same probability.

Let Chal denote the challenger in $\mathsf{PseudoRand}$. $\mathcal{B}$ acts as follows:

- *Initialization.* Get pp_{DP} from Chal and run $\Sigma.\mathsf{Setup}(1^\kappa)$ to get pp_{com}. Give $(pp_{\mathsf{DP}}, pp_{\mathsf{com}})$ to $\mathcal{A}$.
- *Corruption.* Receive the set of corrupt parties C from $\mathcal{A}$, where $|C| < t$. Pass it on to Chal. When Chal returns $\{sk_i\}_{i \in C}$, forward them to $\mathcal{A}$.
- *Pre-challenge encryption queries.* Suppose $\mathcal{A}$ outputs $(\texttt{Encrypt}, j, m, S)$, where $j \in S$ and $|S| \geq t$. There are two possibilities:
  - If j is honest, compute $\alpha := \mathsf{Com}(m, pp_{\mathsf{com}}; \rho)$ for a randomly chosen ρ. If α is not unique among all the commitments generated so far, notify $\mathcal{A}$ and stop; else, send α to it. $\mathcal{A}$ responds with a z_i for every $i \in S \cap C$. At the same time, send $(\texttt{Eval}, j \| \alpha, i)$ to Chal and get back a response z_i for every $i \in S \setminus C$. Now run $\mathsf{Combine}(\{(i, z_i)\}_{i \in S}, pp)$ to get w or $\perp$. In the latter case, send $\perp$ to $\mathcal{A}$. Otherwise, compute $e := \mathsf{PRG}(w) \oplus (m \| \rho)$ and send $c := (j, \alpha, e)$ to $\mathcal{A}$.
  - If j is corrupt, $\mathcal{A}$ expects the honest parties in S to help compute a DPRF value. For a message x sent to an honest party j' in S, send $(\texttt{Eval}, j \| x, j')$ to Chal. Forward Chal's response to $\mathcal{A}$.
- *Pre-challenge indirect decryption queries.* Suppose $\mathcal{A}$ outputs $(\texttt{Decrypt}, j, c, S)$, where $j \in S \setminus C$ and $|S| \geq t$. If c parses to (j', α, e), send $j' \| \alpha$ to $\mathcal{A}$. Ignore any response from $\mathcal{A}$.
- *Challenge.* $\mathcal{A}$ outputs $(\texttt{Challenge}, j^\star, m_0, m_1, S^\star)$ where $|m_0| = |m_1|$, $j^\star \in S^\star \setminus C$ and $|S^\star| \geq t$. Compute $\alpha^\star := \mathsf{Com}(m_b, pp_{\mathsf{com}}; \rho^\star)$ by picking $\rho^\star$ at random and, based on the uniqueness of $\alpha^\star$, do the same thing as before. For every $i \in S^\star \cap C$, $\mathcal{A}$ provides a value $\hat{z}_i$. Send $(\texttt{Challenge}, j^\star \| \alpha^\star, S, \{(i, \hat{z}_i)\}_{i \in S^\star \cap C})$ to Chal. If Chal returns $\perp$, forward the same to $\mathcal{A}$. Otherwise, use Chal's response $w^\star$ to compute $e^\star := \mathsf{PRG}(w^\star) \oplus (m_b \| \rho^\star)$, and give $c^\star := (j^\star, \alpha^\star, e^\star)$ to $\mathcal{A}$.
- *Post-challenge encryption queries.* This is same as the pre-challenge encryption phase.
- *Post-challenge indirect decryption queries.* This is same as the pre-challenge decryption phase.
- *Guess.* Finally, $\mathcal{A}$ returns a guess b'. Output b'.

$\mathcal{B}$ makes an $\texttt{Eval}$ request in the (pre- or post-challenge) encryption phase only. When j is honest, a request is made only when α is unique. When j is corrupt, $j \| x$ can not be equal to $j^\star \| \alpha^\star$ because $j^\star$ is honest. As a result, $\mathcal{B}$ never requests any evaluation on $j^\star \| \alpha^\star$. Therefore, the challenge message it sends to Chal is valid. Now it is easy to see that when $w^\star := \mathsf{Combine}(\{(i, z_i)\}_{i \in S^\star \setminus C} \cup \{(i, \hat{z}_i)\}_{i \in S^\star \cap C}, pp)$, $\mathcal{B}$ perfectly simulates Hyb_1, and when $w^\star$ is random, it perfectly simulates Hyb_2.

C.2 Proof of Theorem 7.5

We first write down the last step of the authenticity game AUTH, forgery, in detail. Recall that ct counts the total number of times honest parties are contacted in encryption/decryption protocols initiated by corrupt parties and g captures the minimum number of honest parties an adversary must contact in order to get enough information to generate one ciphertext. Let $k := \lfloor \mathsf{ct}/g \rfloor$.

1. $\mathcal{A}$ outputs $((j_1, S_1, c_1), (j_2, S_2, c_2), \ldots, (j_{k+1}, S_{k+1}, c_{k+1}))$ s.t. $j_\ell \in S \setminus C$, $|S_\ell| \geq t$ for $\ell \in [k+1]$ and $c_u \neq c_v$ for any $u \neq v \in [k+1]$. Let $c_\ell := (j_\ell, \alpha_\ell, e_\ell)$ for $\ell \in [k+1]$.

2. For $\ell \in [k+1]$, compute $z_{\ell,i} \leftarrow \mathsf{DP.Eval}(sk_i, j_\ell \| \alpha_\ell, pp)$ for all $i \in S_\ell$ and $w_\ell := \mathsf{Combine}(\{(i, z_{\ell,i}\}_{i \in S_\ell}, pp)$. Output 0 if any $w_\ell = \perp$. Else, compute $m_\ell \| \rho_\ell := \mathsf{PRG}(w_\ell) \oplus e_\ell$ and check if $\alpha_\ell = \mathsf{Com}(m_\ell, pp_{\mathsf{com}}; \rho_\ell)$. Output 0 if the check fails for any ℓ. Else, output 1.

We define a new authenticity game called $\mathsf{AUTH\text{-}U}$ (U stands for unique). It differs from AUTH in the first step of forgery:

- $\mathcal{A}$ outputs $((j_1, S_1, c_1), (j_2, S_2, c_2), \ldots, (j_{k+1}, S_{k+1}, c_{k+1}))$ s.t. $j_\ell \in S \setminus C$, $|S_\ell| \geq t$ for $\ell \in [k+1]$ and $c_u \neq c_v$ for any $u \neq v \in [k+1]$. Let $c_\ell := (j_\ell, \alpha_\ell, e_\ell)$ for $\ell \in [k+1]$. Output 0 if for any $u \neq v$, $j_u = j_v$ and $\alpha_u = \alpha_v$.

We show that AUTH is indistinguishable from $\mathsf{AUTH\text{-}U}$. Suppose an adversary outputs $k+1$ distinct ciphertexts s.t. there are two ciphertexts c_u and c_v for which $j_u = j_v$ and $\alpha_u = \alpha_v$ (thus, $e_u \neq e_v$). We show that in this case even AUTH will output 0. Since all the parties involved in decrypting forged ciphertexts are assumed to behave honestly, $w_u = w_v$ with high probability due to the consistency property of DP. Together with $e_u \neq e_v$, this implies that $m_u \| \rho_u \neq m_v \| \rho_v$. As a result, $\mathsf{Com}(m_u, pp_{\mathsf{com}}; \rho_u) \neq \mathsf{Com}(m_v, pp_{\mathsf{com}}; \rho_v)$ with all but negligible probability (binding property of Σ).

We now show that if there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{AUTH\text{-}U}$ outputs 1 with probability at least ε, then one can use it to build another PPT adversary $\mathcal{B}$ for the pseudorandomness game $\mathsf{PseudoRand}$ (Definition 5.3) whose advantage is at least $\varepsilon - \mathsf{negl}$ for some negligible function negl. Let Chal denote the challenger for the pseudorandomness game. $\mathcal{B}$ acts as follows:

- *Initialization.* Get pp_{DP} from Chal and run $\Sigma.\mathsf{Setup}(1^\kappa)$ to get pp_{com}. Give $(pp_{\mathsf{DP}}, pp_{\mathsf{com}})$ to $\mathcal{A}$. Initialize an ordered list $L_{\mathsf{p\text{-}ctxt}} := \emptyset$ and a counter $\mathsf{ct}_{j,\alpha} := 0$ for every (j, α). (Here, $\mathsf{p\text{-}ctxt}$ stands for partial ciphertext.)

- *Corruption.* Receive the set of corrupt parties C from $\mathcal{A}$, where $|C| < t$. Pass it on to Chal. When Chal returns $\{sk_i\}_{i \in C}$, forward them to $\mathcal{A}$. Set $g := t - |C|$.

- *Encryption queries.* Suppose $\mathcal{A}$ outputs $(\texttt{Encrypt}, j, m, S)$, where $j \in S$ and $|S| \geq t$. There are two possibilities:

  - If j is honest, send $\alpha \leftarrow \mathsf{Com}(m, pp_{\mathsf{com}}; \rho)$ to $\mathcal{A}$, where ρ is picked at random. Ignore any response from $\mathcal{A}$. Append (j, α) to $L_{\mathsf{p\text{-}ctxt}}$.
  - If j is corrupt, $\mathcal{A}$ expects the honest parties in S to help compute a DPRF value. For a message x sent to an honest party j' in S, send $(\texttt{Eval}, j \| x, j')$ to Chal. Forward Chal's response to $\mathcal{A}$ and increment $\mathsf{ct}_{j,x}$ by 1.

– *Decryption queries.* Suppose $\mathcal{A}$ outputs $(\texttt{Decrypt}, j', c, S)$, where $j' \in S$ and $|S| \geq t$. Parse c into (j, α, e). Now:

  – If j' is honest, send $j \| \alpha$ to $\mathcal{A}$. Ignore any response from $\mathcal{A}$.
  – If j' is corrupt, for a message x sent to an honest party $j'' \in S$ by $\mathcal{A}$, check if x is of the form $j^\star \| \alpha^\star$ for some $j^\star \in [n]$. If not, return $\perp$ to $\mathcal{A}$. Else, send $(\texttt{Eval}, x, j'')$ to Chal and forward its response to $\mathcal{A}$. Increment $\mathsf{ct}_{j^\star, \alpha^\star}$ by 1.

– *Targeted decryption queries.* Suppose $\mathcal{A}$ outputs $(\texttt{TargetDecrypt}, j', \ell, S)$, where $j' \in S \setminus C$ and $|S| \geq t$. Let (j, α) be the ℓ-th entry of $L_{\mathsf{p\text{-}ctxt}}$. Send $j \| \alpha$ to $\mathcal{A}$. Ignore any response from $\mathcal{A}$.

– *Forgery.* $\mathcal{A}$ outputs $((j_1, S_1, c_1), (j_2, S_2, c_2), \ldots, (j_\tau, S_\tau, c_\tau))$ s.t. $j_\ell \in S \setminus C$, $|S_\ell| \geq t$ for $\ell \in [\tau]$ and $c_u \neq c_v$ for any $u \neq v \in [\tau]$. Let $c_\ell := (j_\ell, \alpha_\ell, e_\ell)$ for $\ell \in [\tau]$. Now:

  – If for any $u \neq v$, $j_u = j_v$ and $\alpha_u = \alpha_v$, output 1 as the guess and stop.
  – Pick a $c_{\ell^\star}$ such that $\mathsf{ct}_{j_{\ell^\star}, \alpha_{\ell^\star}} < g$. Send $(\texttt{Challenge}, j_{\ell^\star} \| \alpha_{\ell^\star}, S_{\ell^\star}, \emptyset)$ to Chal. Let $w^\star$ be Chal's response. If $w^\star = \perp$, output 1; else, compute $m \| \rho := \mathsf{PRG}(w^\star) \oplus e_{\ell^\star}$ and check if $\alpha_{\ell^\star} = \mathsf{Com}(m, pp_{\mathsf{com}}; \rho)$. If the check fails, output 1; else, output 0.

Note that the sum of $\mathsf{ct}_{j,\alpha}$ for all (j, α) is at most ct, the variable in $\mathsf{AUTH\text{-}U}$ that keeps track of the number of times honest parties are contacted. In turn, ct must be less than $\tau \cdot g$ as required by $\mathsf{AUTH\text{-}U}$. Thus, there must exist a $c_{\ell^\star}$ such that $\mathsf{ct}_{j_{\ell^\star}, \alpha_{\ell^\star}} < g$. This ensures that $j_{\ell^\star} \| \alpha_{\ell^\star}$ is not in the list L maintained by Chal.

$\mathcal{B}$ simulates $\mathsf{AUTH\text{-}U}$ perfectly for $\mathcal{A}$. Let b' denote the bit output by $\mathcal{B}$ at the end. Suppose the bit b in $\mathsf{PseudoRand}$ is 0. From the third item in the first paragraph of this section, we can see that $\mathsf{AUTH\text{-}U}$ outputs 1 only if $w_\ell \neq \perp$ and $\alpha_\ell = \mathsf{Com}(m_\ell, pp_{\mathsf{com}}; \rho_\ell)$ succeeds for all ℓ—and, in particular, for $\ell = \ell^\star$. We have assumed that $\mathsf{AUTH\text{-}U}$ outputs 1 with probability at least ε. Thus, $\Pr[b' = 1 \mid b = 0] \leq 1 - \varepsilon$.

On the other hand, when $b = 1$, $w^\star$ is picked at random, so $m \| \rho$ is a pseudo-random value. The probability that $\alpha_{\ell^\star} = \mathsf{Com}(m, pp_{\mathsf{com}}; \rho)$ is at most negl for some negligible function negl due to the binding property of Σ. Therefore, $\Pr[b' = 1 \mid b = 1] \geq 1 - \mathsf{negl}$. So,

$$\left|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]\right| \geq \varepsilon - \mathsf{negl}.$$

C.3 Proof of Theorem 7.6

In the case of strong authenticity, we cannot assume anymore that the corrupt parties involved in decrypting forged ciphertexts behave honestly. As a result, first of all, we cannot use the consistency property of DPRF alone to argue the indistinguishability of AUTH and $\mathsf{AUTH\text{-}U}$. We will need the correctness property too.

We first define a notion called *validity* for DPRFs which combines both consistency and correctness. Formally, an (n, t)-DPRF $\mathsf{DP} := (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Combine})$ is *valid* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that the following game outputs 1 with probability at least $1 - \mathsf{negl}(\kappa)$.

– *Initialization.* Run $\mathsf{Setup}(1^\kappa, n, t)$ to get $((sk_1, \ldots, sk_n), pp)$. Give pp to $\mathcal{A}$.

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

– *Evaluation.* In response to $\mathcal{A}$'s evaluation query $(\texttt{Eval}, x, i)$ for some $i \in [n] \setminus C$, return $\mathsf{Eval}(sk_i, x, pp)$ to $\mathcal{A}$. Repeat this step as many times as $\mathcal{A}$ desires.

– *Computation.* Receive two sets S_0, S_1 of size at least t, an input $x^\star$, and shares $\{(i, z^\star_{0,i})\}_{i \in S_0 \cap C}$, $\{(i, z^\star_{1,i})\}_{i \in S_1 \cap C}$ from $\mathcal{A}$. Let $z_{0,j} \leftarrow \mathsf{Eval}(sk_j, x^\star, pp)$ for $j \in S_0 \setminus C$ and $z_{1,j} \leftarrow \mathsf{Eval}(sk_j, x^\star, pp)$ for $j \in S_1 \setminus C$. Let $z^\star_b := \mathsf{Combine}(\{j, z_{b,j}\}_{j \in S_b \setminus C} \cup \{(i, z^\star_{b,i})\}_{i \in S_b \cap C}, pp)$ for $b \in \{0, 1\}$. Output 1 if $z^\star_0 = \perp$ or $z^\star_1 = \perp$ or $z^\star_0 = z^\star_1$; else, output 0.

Notice that validity is different from consistency in the sense that corrupt parties also contribute to DPRF evaluation, and it is different from correctness in the sense that both DPRF evaluations involve corrupt parties (instead of just one).

Fix a PPT adversary $\mathcal{A}$ for the validity game defined above. Let $Z^\star_0$ and $Z^\star_1$ be random variables that capture the distribution of $z^\star_0$ and $z^\star_1$ in the game, respectively. Define two adversaries $\mathcal{A}_0$ and $\mathcal{A}_1$ for the correctness game based on $\mathcal{A}$ as follows: for $b \in \{0, 1\}$, $\mathcal{A}_b$ runs $\mathcal{A}$ up to the end of the evaluation phase and, when $\mathcal{A}$ produces its output for the computation phase, $\mathcal{A}_b$ outputs the b-th part of it (i.e., S_b, $x^\star$ and $\{(i, z^\star_{b,i})\}_{i \in S_b \cap C}$). Let $\overline{Z}_b$ and $\overline{Z}^\star_b$ be random variables that capture the distribution of z (the *true* DPRF value) and $z^\star$ in the correctness game with $\mathcal{A}_b$, for $b \in \{0, 1\}$.

On account of DPRF DP being correct, we have that

$$\Pr[\overline{Z}^\star_b \in \{\overline{Z}_b, \perp\}] \geq 1 - \varepsilon_b(\kappa)$$

for $b \in \{0, 1\}$ and a negligible function ε_b. Further, the consistency of DP gives us that

$$\Pr[\overline{Z}_0 = \overline{Z}_1 \neq \perp] \geq 1 - \varepsilon$$

for some negligible function ε. Now, observe that $Z^\star_b$ is identically distributed to $\overline{Z}^\star_b$ for $b \in \{0, 1\}$ due to the definition of the validity and correctness games as well as the construction of $\mathcal{A}_0$, $\mathcal{A}_1$ from $\mathcal{A}$. Therefore,

$$\begin{aligned}
\Pr[\text{Validity game outputs 1}] &= \Pr[Z^\star_0 = \perp \vee Z^\star_1 = \perp \vee Z^\star_0 = Z^\star_1] \\
&= \Pr[\overline{Z}^\star_0 = \perp \vee \overline{Z}^\star_1 = \perp \vee \overline{Z}^\star_0 = \overline{Z}^\star_1] \\
&\geq \Pr\left[\begin{matrix} \overline{Z}^\star_0 \in \{\overline{Z}_0, \perp\} \wedge \overline{Z}^\star_1 \in \{\overline{Z}_1, \perp\} \\ \wedge \overline{Z}_0 = \overline{Z}_1 \end{matrix}\right] \\
&\geq 1 - (\varepsilon_0 + \varepsilon_1 + \varepsilon).
\end{aligned}$$

Therefore, validity game outputs 1 with all but negligible probability. Since the above analysis holds for any PPT adversary, DP is a valid DPRF.

We now turn to arguing the indistinguishability of AUTH and $\mathsf{AUTH\text{-}U}$. The forgery step of AUTH is now different from the previous case. We first write it down in detail here:

1. $\mathcal{A}$ outputs $((j_1, S_1, c_1), (j_2, S_2, c_2), \ldots, (j_{k+1}, S_{k+1}, c_{k+1}))$ s.t. $j_\ell \in S \setminus C$, $|S_\ell| \geq t$ for $\ell \in [k+1]$ and $c_u \neq c_v$ for any $u \neq v \in [k+1]$. Let $c_\ell := (j_\ell, \alpha_\ell, e_\ell)$ for $\ell \in [k+1]$.

2. For $\ell \in [k+1]$,

   – for $i \in S_\ell \cap C$, send $j_\ell \| \alpha_\ell$ to $\mathcal{A}$ and get back $z^\star_{\ell,i}$;

   – for $i \in S_\ell \setminus C$, compute $z_{\ell,i} \leftarrow \mathsf{DP.Eval}(sk_i, j_\ell \| \alpha_\ell, pp)$.

3. Compute $w_\ell := \mathsf{Combine}(\{(i, z^\star_{\ell,i})\}_{i\in S_\ell\cap C} \cup \{(i, z_{\ell,i}\}_{i\in S_\ell\setminus C}, pp)$ for $\ell \in [k+1]$. Output 0 if any $w_\ell = \perp$. Else, compute $m_\ell \| \rho_\ell := \mathsf{PRG}(w_\ell) \oplus e_\ell$ and check if $\alpha_\ell = \mathsf{Com}(m_\ell, pp_{\mathsf{com}}; \rho_\ell)$. Output 0 if the check fails for any ℓ. Else, output 1.

Note that corrupt parties can now provide arbitrary shares of DPRF values during the decryption of forged ciphertexts (step 2).

AUTH-U is different from AUTH in the same way as before. Specifically, the difference is in the first step of forgery:

– $\mathcal{A}$ outputs $((j_1, S_1, c_1), (j_2, S_2, c_2), \ldots, (j_{k+1}, S_{k+1}, c_{k+1}))$ s.t. $j_\ell \in S \setminus C$, $|S_\ell| \geq t$ for $\ell \in [k+1]$ and $c_u \neq c_v$ for any $u \neq v \in [k+1]$. Let $c_\ell := (j_\ell, \alpha_\ell, e_\ell)$ for $\ell \in [k+1]$. Output 0 if for any $u \neq v$, $j_u = j_v$ and $\alpha_u = \alpha_v$.

Let E denote the event that among the ciphertexts output by the adversary, there exists $u \neq v$ such that $j_u = j_v$ and $\alpha_u = \alpha_v$. Clearly, the probability that E happens is the same in AUTH and AUTH-U.

Fix any PPT adversary $\mathcal{A}$. We now informally describe an adversary $\mathcal{B}'$ who simulates AUTH for $\mathcal{A}$ with the help of the challenger, say Chal', of the validity game. $\mathcal{B}'$ will be able to handle all the encryption and decryption queries of $\mathcal{A}$ by making evaluation queries to Chal'. When $\mathcal{A}$ outputs $k+1$ forgeries, $\mathcal{B}'$ will check if the event E occurs or not. If not, then it aborts. Otherwise, $\mathcal{B}'$ will pick any $u \neq v$ s.t. $j_u = j_v$ and $\alpha_u = \alpha_v$. For all $\ell \in [k+1] \setminus \{u, v\}$, $\mathcal{B}'$ will evaluate w_ℓ by making evaluation queries. Then, it will send $(S_u, S_v, j_u \| \alpha_u, \{(i, z^\star_{u,i})\}_{i\in S_u\cap C}, \{(i, z^\star_{v,i})\}_{i\in S_v\cap C})$ to Chal'. Assume, for simplicity, that Chal' returns $z^\star_0$ and $z^\star_1$ to $\mathcal{B}'$, which we refer to as w_u and w_v, respectively. Now, given $w_1, \ldots, w_{k+1}$, $\mathcal{B}'$ outputs the final bit in the same way as AUTH.

To show that AUTH and AUTH-U are indistinguishable, all we need to do is argue that when E occurs, AUTH also outputs 0 with high probability. When E occurs, $\mathcal{B}'$ is a valid adversary for the validity game. Hence, $w_u = \perp$, $w_v = \perp$ or $w_u = w_v$ with all but negligible probability. In the first two cases, $\mathcal{B}'$ clearly outputs 0. Further, in the last case, by the same argument as in the previous proof, $\mathcal{B}'$ outputs 0 with high probability.

Finally, when using an advantage in AUTH-U to break the pseudorandomness of DP like in the previous proof, the construction of $\mathcal{B}$ needs to change slightly. Since corrupt parties are involved in the final decryption, they will provide partial shares of DPRF too. Therefore, instead of sending $(\texttt{Challenge}, j_{\ell^\star} \| \alpha_{\ell^\star}, S_{\ell^\star}, \emptyset)$ to Chal in the forgery step, $\mathcal{B}$ will send $(\texttt{Challenge}, j_{\ell^\star} \| \alpha_{\ell^\star}, S_{\ell^\star}, \{(i, z^\star_{\ell^\star,i})_{i\in S_\ell\cap C}\})$ instead, where for $\ell \in [\tau]$ and $i \in S_\ell \cap C$, $z^\star_{\ell,i}$ is returned by $\mathcal{A}$ when $j_\ell \| \alpha_\ell$ is sent to it. This change does not affect the argument that if AUTH-U outputs 1 with probability ε then $\mathcal{B}$ will be able to get a similar advantage in the pseudorandomness game.

C.4 Proof of Theorem 8.1

Recall that a secure DPRF protocol is supposed to provide consistency and pseudo-randomness guarantees but not necessarily correctness. First, it is straightforward to see that $\Pi_{\mathsf{DDH\text{-}DP}}$ is consistent due to the properties of Shamir's secret sharing.

Pseudorandomness. We show that if the DDH assumption holds in group G, then $\Pi_{\mathsf{DDH\text{-}DP}}$ satisfies the pseudorandomness property in the random oracle model. We will go through several hybrids to establish this. For any PPT adversary $\mathcal{A}$, let us first consider the real game $\mathsf{PseudoRand}_{\Pi_{\mathsf{DDH\text{-}DP}},\mathcal{A}}(1^\kappa, b)$. For simplicity, let us denote the game just by $\mathsf{PseudoRand}_{\mathcal{A}}(b)$.

$\underline{\mathsf{PseudoRand}_{\mathcal{A}}(b)}$:

1. Give the public parameters $pp := (p, g, G)$ (G is a cyclic group of order p and g is a generator of G) to $\mathcal{A}$.
2. Program the random oracle $\mathcal{H}$ as follows: Initialize $\mathcal{L}_{\mathcal{H}} := \emptyset$. For random oracle call with input x:
   1. If there exists a tuple $(x, r, h) \in \mathcal{L}_H$, output h.
   2. Otherwise, choose $r \leftarrow_{\$} \mathbb{Z}_p$ and set $h := g^r$. Update $\mathcal{L}_{\mathcal{H}} := \mathcal{L}_{\mathcal{H}} \cup (x, r, h)$ and output h.

   Give random oracle access to $\mathcal{A}$.
3. Choose a $(t-1)$-degree random polynomial f. Define $s_i := f(i)$ for $i \in [n] \cup \{0\}$. Get the set of corrupt parties C from $\mathcal{A}$. Without loss of generality assume that $C = \{1, \ldots \ell\}$. Then send the corresponding secret keys $\{s_1, \ldots, s_\ell\}$ to $\mathcal{A}$.
4. On an evaluation query $(\texttt{Eval}, x, i)$ for an honest i, return $\mathcal{H}(x)^{s_i}$.
5. On the challenge query $(\texttt{Challenge}, x^\star, S, g_1^\star, \ldots, g_u^\star)$ for $u \leq \ell$ (without loss of generality assume that $S \cap C = [u]$):
   1. If $\mathcal{A}$ has already made at least $t - |C|$ queries of the form $(\texttt{Eval}, x^\star, *)$, then output 0 and stop.
   2. Otherwise do as follows:
      1. Set $g_i^\star := \mathcal{H}(x^\star)^{s_i}$ for $i \in S \setminus C$.
      2. Depending on b do as follows:
         1. If $b = 0$ then compute $z := \prod_{i \in S} g_i^{\star \lambda_{0,i,S}}$.
         2. Else, choose a random $z \leftarrow_{\$} G$.
   3. Send z to $\mathcal{A}$.
6. Continue answering evaluation queries as before, but if $\mathcal{A}$ makes a query of the form $(\texttt{Eval}, x^\star, i)$ for some $i \in [n] \setminus C$ and i is the g-th party it contacted, then output 0 and stop.
7. Receive a guess b' from $\mathcal{A}$; output b'.

For any adversary $\mathcal{A}$ that asks evaluation queries on q_E distinct x, we now define hybrid games $\mathsf{Hyb}_{\mathcal{A}}^{(k)}(b)$ for $k \in [q_E]$ and $b \in \{0, 1\}$. The only difference between $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ is the way the evaluation queries for the k-th distinct x are answered. Specifically, in the k-th hybrid these queries are answered using a randomly chosen $(t-1)$-degree polynomial $f^{(k)}$, where evaluations of exactly ℓ points match with that of "real polynomial" f, and $f^{(k)}$ is only used to reply to such queries. On the other hand, in the $(k-1)$-th hybrid the queries on k-th distinct x are answered according to f itself. However, in both the games the evaluation queries for the first $k-1$ distinct x are answered using randomly chosen polynomials and all the subsequent queries (for the $(k+1)$-th distinct x onwards) are answered using f.

We will formally specify hybrid $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ now. Differences with the game $\mathsf{PseudoRand}$ are highlighted in red.

$\underline{\mathsf{Hyb}_{\mathcal{A}}^{(k)}(b):}$

1. Give the public parameters $pp := (p, g, G)$ to $\mathcal{A}$.
2. Program the random oracle $\mathcal{H}$ as follows: Initialize $\mathcal{L}_{\mathcal{H}} := \emptyset$. For random oracle call with input x:
   1. If there exists a tuple $(x, r, h) \in \mathcal{L}_{\mathcal{H}}$, output h.
   2. Otherwise, choose $r \leftarrow_{\$} \mathbb{Z}_p$ and set $h := g^r$. Update $\mathcal{L}_{\mathcal{H}} := \mathcal{L}_{\mathcal{H}} \cup (x, r, h)$ and output h.

   Give random oracle access to $\mathcal{A}$.
3. Choose a $(t-1)$-degree random polynomial f. Define $s_i := f(i)$ for $i \in [n] \cup \{0\}$. Get the set of corrupt parties C from $\mathcal{A}$. Without loss of generality assume that $C = \{1, \ldots \ell\}$. Then send the corresponding secret keys $\{s_1, \ldots, s_\ell\}$ to $\mathcal{A}$.
4. Choose k $(t-1)$-degree random polynomials $f^{(1)}, \ldots, f^{(k)}$ with the constraint that for all $i \in [\ell]$ and all $j \in [k]$, $f^{(j)}(i) = s_i$. Define $\tilde{s}_i^{(j)} = f^{(j)}(i)$ for $i \in [n], j \in [k]$.
5. Define a function pc as follows:

$$\mathsf{pc}(x, j, i) := \begin{cases} \mathcal{H}(x)^{\tilde{s}_i^{(j)}} \text{ if } j \leq k \\ \mathcal{H}(x)^{s_i} \text{ otherwise,} \end{cases}$$

   for $i \in [n]$.
6. On an evaluation query $(\texttt{Eval}, x, i)$ for an honest i, if x is the j-th distinct value, then return $\mathsf{pc}(x, j, i)$. [11]
7. On the challenge query $(\texttt{Challenge}, x^\star, S, g_1^\star, \ldots, g_u^\star)$:
   1. If $x^\star$ was queried in the evaluation phase and it was the j-th distinct value, then let $j^\star := j$. Else, let $j^\star := q_E + 1$.
   2. If $\mathcal{A}$ has already made at least $t - |C|$ queries of the form $(\texttt{Eval}, x^\star, *)$, then output 0 and stop.
   3. Otherwise do as follows:
      1. Set $g_i^\star := \mathsf{pc}(x^\star, j^\star, i)$ for $i \in S \setminus C$.
      2. Depending on b do as follows:
         1. If $b = 0$ then compute $z := \prod_{i \in S} g_i^{\star \lambda_{0,i,S}}$.
         2. Else, choose a random $z \leftarrow_{\$} G$.
   4. Send z to $\mathcal{A}$.
8. Continue answering evaluation queries as before, but if $\mathcal{A}$ makes a query of the form $(\texttt{Eval}, x^\star, i)$ for some $i \in [n] \setminus C$ and i is the g-th party it contacted, then output 0 and stop.

[11] To clarify a bit more, suppose $\mathcal{A}$ makes three evaluation queries: $(\texttt{Eval}, x_1, i_1)$, $(\texttt{Eval}, x_2, i_2)$ and $(\texttt{Eval}, x_1, i_3)$ such that $x_1 \neq x_2$. For both the first and third queries, j will be 1, and for the second, it will be 2.

9. Receive a guess b' from $\mathcal{A}$; output b'.

It is easy to check that the view of $\mathcal{A}$ in $\mathsf{Hyb}_{\mathcal{A}}^{(0)}(b)$ is identical to that in $\mathsf{PseudoRand}_{\mathcal{A}}(b)$. We now prove the following lemma.

Lemma C.1 *For any $b \in \{0,1\}$ and $k \in [q_E]$, the outputs of hybrids $\mathsf{Hyb}_{\mathcal{A}}^{(k-1)}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{(k)}(b)$ are computationally indistinguishable.*

Proof. We show that if there exists a PPT adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ with non-negligible probability then we can construct a PPT adversary $\mathcal{A}'$ that can break an extended version of the DDH assumption with non-negligible probability using $\mathcal{A}$ as a sub-routine.

A DDH-tuple over a cyclic group G is given by $(g, g^\alpha, g^\beta, y)$ where $\alpha, \beta \leftarrow_\$ \mathbb{Z}_p$ and y is equal to $g^{\alpha\beta}$ or a random element in G. The extended version of DDH we consider here is given by $(g, g^{\alpha_0}, g^{\alpha_1}, g^{\alpha_w}, g^\beta, y_0, y_1, \ldots, y_w)$ where $y_i = g^{\alpha_i \beta}$ or random for all $i \in \{0, \ldots, w\}$. One can easily show that this extended version of DDH follows from DDH itself (with some polynomial security loss) as long as w is a polynomial.

We now construct $\mathcal{A}'$ as follows:

1. Forward the public parameters $pp := (p, g, G)$ from the DDH challenger to $\mathcal{A}$. Receive a DDH-tuple (g^α, g^β, y) from the DDH challenger.

2. Program the random oracle $\mathcal{H}$ as follows: Initialize $\mathcal{L}_{\mathcal{H}} := \emptyset$. Let $q_{\mathcal{H}}$ be the total number of random oracle queries asked in this game. Guess an index $\eta^\star \leftarrow_\$ [q_{\mathcal{H}}]$ randomly. For random oracle call with input x,

   1. If there exists a tuple $(x, r, h) \in \mathcal{L}_H$, output h.
   2. Otherwise,
      1. if this is the $\eta^\star$-th call, set $r := \perp$ and $h := g^\beta$, where g^β is from the DDH tuple;
      2. else, choose $r \leftarrow_\$ \mathbb{Z}_p$ and set $h := g^r$.

      Update $\mathcal{L}_{\mathcal{H}} := \mathcal{L}_{\mathcal{H}} \cup (x, r, h)$ and output h.

   Give random oracle access to $\mathcal{A}$.

3. Get the set of corrupt parties C from $\mathcal{A}$. Without loss of generality assume that $C = \{1, \ldots \ell\}$. Then proceed as follows:

   1. Choose random $s_i \leftarrow_\$ \mathbb{Z}_p$ and define $\hat{g}_i := g^{s_i}$ for $i \in [\ell]$.
   2. Then let $\hat{g}_0 := g^{\alpha_0}$ and $\hat{g}_i := g^{\alpha_i}$ for $i \in [\ell + 1, t-1]$, where $g^{\alpha_0}, g^{\alpha_{\ell+1}}, \ldots, g^{\alpha_{t-1}}$ comes from the DDH-tuple. The values $\alpha_0, s_1, \ldots, s_\ell, \alpha_{\ell+1}, \ldots, \alpha_{t-1}$ define a $(t-1)$-degree polynomial f. Setting $T := \{0\} \cup [t-1]$, compute $\hat{g}_i := \prod_{j \in T} \hat{g}_j^{\lambda_{i,j,T}}$ for all $i \in \{t, \ldots, n\}$.
   3. Send $\{s_1 \ldots, s_\ell\}$ to $\mathcal{A}$.

4. Choose $k-1$ $(t-1)$-degree random polynomials, $f^{(1)}, \ldots, f^{(k-1)}$ with the constraint that for all $i \in [\ell]$ and all $j \in [k-1]$, $f^{(j)}(i) = s_i$. Define $\tilde{s}_i^{(j)} := f^{(j)}(i)$ for $i \in [n], j \in [k-1]$.

5. Compute $\bar{g}_i := (g^\beta)^{s_i}$ for all $i \in [\ell]$. Set $\bar{g}_0 := y_0$ and $\bar{g}_i := y_i$ for $i \in [\ell+1, t-1]$, where $y_0, y_{\ell+1}, \ldots, y_{t-1}$ are from the DDH-tuple. Then compute $\bar{g}_i := \prod_{j \in T} \bar{g}_j^{\lambda_{i,j,T}}$ for all $i \in \{t, \ldots, n\}$.

6. Define a function pc as follows. On input (x, i, j) for $i \in [n]$, pc first invokes RO on x. This associates a tuple (x, r, h) with x in $\mathcal{L}_H$ if there wasn't one already. Now, pc returns
   - $\mathcal{H}(x)^{\tilde{s}_i^{(j)}}$ if $j < k$,
   - $\bar{g}_i$ if $j = k$,
   - $\hat{g}_i^r$ if $j > k$ (if $r = \perp$, return $\perp$).

7. On an evaluation query $(\texttt{Eval}, x, i)$ for an honest i, if x is the j-th distinct value, then return $\mathsf{pc}(x, j, i)$.

8. On the challenge query $(\texttt{Challenge}, x^\star, S, g_1^\star, \ldots, g_u^\star)$:
   1. If $x^\star$ was queried in the evaluation phase and it was the j-th distinct value, then let $j^\star := j$. Else, let $j^\star := q_E + 1$.
   2. If $\mathcal{A}$ has already made at least $t - |C|$ queries of the form $(\texttt{Eval}, x^\star, *)$, then output 0 and stop.
   3. Otherwise do as follows:
      1. Set $g_i^\star := \mathsf{pc}(x^\star, j^\star, i)$ for $i \in S \setminus C$.
      2. Depending on b do as follows:
         1. If $b = 0$ then compute $z := \prod_{i \in S} g_i^{\star \lambda_{0,i,S}}$
         2. Else, choose a random $z \leftarrow_\$ G$.
   4. Send z to $\mathcal{A}$.

9. Continue answering evaluation queries as before, but if $\mathcal{A}$ makes a query of the form $(\texttt{Eval}, x^\star, i)$ for some $i \in [n] \setminus C$ and i is the g-th party it contacted, then output 0 and stop.

10. Receive a guess b' from $\mathcal{A}$; output b'.

Let x_j denote the j-th distinct x on which an evaluation/challenge query is made. Suppose the first RO query made on x_k is the η-th one. Let us consider the case when $\eta^\star = \eta$. First of all, pc never returns $\perp$ in Step 6. because r is set to $\perp$ only for the $\eta^\star$-th call and we have assumed that this call is for the k-th distinct x. Secondly, note that RO returns g^β when queried on x_k and β is associated with x_k only.

Let's consider the two possibilities for $y_0, y_{\ell+1}, \ldots, y_{t-1}$. When they are equal to $g^{\alpha_0 \beta}$, $g^{\alpha_{\ell+1}\beta}$, …, $g^{\alpha_{t-1}\beta}$, respectively, then pc's return value $\bar{g}_i$ on $j = k$ is equal to $\mathcal{H}(x_k)^{f^{(k)}(i)}$ where $f^{(k)}$ is the $(t-1)$-degree polynomial that satisfies $f^{(k)}(0) = \alpha_0$, $f^{(k)}(1) = s_1$, …, $f^{(k)}(\ell) = s_\ell$, $f^{(k)}(\ell+1) = \alpha_{\ell+1}$, …, $f^{(k)}(t-1) = \alpha_{t-1}$. This is exactly the same polynomial f as defined in Step 3.. Thus $\mathcal{A}'$ perfectly simulates $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ in this case.

When $y_0, y_{\ell+1}, \ldots, y_{t-1}$ are random, $f^{(k)}$ is a completely random polynomial except that it "matches" with f on $s_1, \ldots, s_\ell$. Thus, in this case, $\mathcal{A}'$ simulates $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ perfectly.

As a result, when $\eta^\star = \eta$, which happens with probability $1/q_{\mathcal{H}}$, $\mathcal{A}'$ distinguishes between the two possibilities described above with non-negligible probability. Since $q_{\mathcal{H}}$ is polynomial in the security parameter, this breaks the (extended version of) DDH assumption. ■

Now, we claim that the views of any adversary in the games $\mathsf{Hyb}_{\mathcal{A}}^{(q_E)}(0)$ and $\mathsf{Hyb}_{\mathcal{A}}^{(q_E)}(1)$ are statistically close. Irrespective of whether $x^\star$ is queried in the evaluation phase or not, a unique $(t-1)$-degree polynomial, say f', is used for it. f' "matches" with any other polynomial only on $1,\ldots,\ell$, and is completely random otherwise. Adversary is allowed to make up to $g-1 = t-\ell-1$ evaluation queries on $x^\star$. Thus, it could learn $\mathcal{H}(x^\star)^{f'(i)}$ for $g-1$ additional values of i. In total, information-theoretically, adversary learns the value of f' on at most $t-1$ points. As a result, the product $\prod_{i\in S} g_i^{\star\lambda_{0,i,S}}$ $(|S| \geq t)$ computed in the $b=0$ case has at least one $g_i^\star$ for which adversary has no information. So the product appears random to the adversary, making the $b=0$ case indistinguishable from $b=1$.

C.5 Proof of Theorem 8.2

A strongly secure DPRF needs to be consistent, pseudorandom and correct. Consistency easily follows from the properties of Shamir's secret sharing and completeness of NIZK.

Pseudorandomness. The overall approach is the same as the proof of Theorem 8.1 in Appendix C.4 but due to the presence NIZK and commitments, we need to go through a couple of hybrids. We will use the zero-knowledge property of NIZKs to simulate proofs and the trapdoor property of commitments to produce fake commitments.

Let $\mathsf{PseudoRand}'_{\mathcal{A}}(b)$ be a shorthand for the game $\mathsf{PseudoRand}_{\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}},\mathcal{A}}(1^\kappa, b)$. We first describe this game in detail, highlighting differences from the game $\mathsf{PseudoRand}_{\mathcal{A}}(b)$ of Appendix C.4.

$\underline{\mathsf{PseudoRand}'_{\mathcal{A}}(b):}$

1. Let G be a cyclic group of order p and g a generator of G. Sample $s \leftarrow_\$ \mathbb{Z}_p$ and get $(s_1,\ldots,s_n) \leftarrow \mathsf{SSS}(n,t,p,s)$. Run $\mathsf{Setup}_{\mathsf{com}}(1^\kappa)$ to get pp_{com}. Compute a commitment $\gamma_i := \mathsf{Com}(s_i, pp_{\mathsf{com}}; r_i)$ by picking r_i at random. Send public parameters $pp = (p, g, G, \gamma_1, \ldots, \gamma_n, pp_{\mathsf{com}})$ to $\mathcal{A}$.

2. Get the set of corrupt parties C from $\mathcal{A}$. Without loss of generality assume that $C = \{1,\ldots\ell\}$. Send the corresponding secret keys $\{(s_1,r_1),\ldots,(s_\ell,r_\ell)\}$ to $\mathcal{A}$.

3. On an evaluation query $(\mathtt{Eval}, x, i)$ for an honest i, compute $w := H(x)$ and $h_i := w^{s_i}$. Run $\mathsf{Prove}^{\mathcal{H}'}$ with the statement stmt_i: $\{\exists s, r \text{ s.t. } h_i = w^s \wedge \gamma_i = \mathsf{Com}(s, pp_{\mathsf{com}}; r)\}$ and witness (s_i, r_i) to obtain a proof π_i. Return $((w,h_i),\pi_i)$ to $\mathcal{A}$.

4. On the challenge query $(\mathtt{Challenge}, x^\star, S, ((w,g_1^\star),\pi_1),\ldots,((w,g_u^\star),\pi_u))$ for $u \leq \ell$ (without loss of generality assume that $S \cap C = [u]$):

    1. If $\mathcal{A}$ has already made at least $t-|C|$ queries of the form $(\mathtt{Eval}, x^\star, *)$, then output 0 and stop.

    2. Otherwise do as follows:

        1. If $\mathsf{Verify}^{\mathcal{H}'}(\mathsf{stmt}_i, \pi_i) \neq 1$ for any $i \in [u]$, output 0 and stop.
        2. Set $g_i^\star := \mathcal{H}(x^\star)^{s_i}$ for $i \in S \setminus C$.
        3. Depending on b do as follows:
            1. If $b=0$ then compute $z := \prod_{i\in S} g_i^{\star\lambda_{0,i,S}}$.
            2. Else, choose a random $z \leftarrow_\$ G$.

3. Send z to $\mathcal{A}$.

5. Continue answering evaluation queries as before, but if $\mathcal{A}$ makes a query of the form $(\texttt{Eval}, x^\star, i)$ for some $i \in [n] \setminus C$ and i is the g-th party it contacted, then output 0 and stop.

6. Receive a guess b' from $\mathcal{A}$; output b'.

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

Second hybrid. Define a hybrid $\mathsf{Hyb}^{(com)}_{\mathcal{A}}(b)$ which is similar to the previous one except:

– In step 1., $\mathsf{SimSetup}$ of TDC is run to get $(pp'_{\mathsf{com}}, \tau_{\mathsf{com}})$, pp_{com} is replaced by pp'_{com} in the public parameters, and γ_i becomes a commitment to some fixed value $s^\star$ for all $i \in [n]$.

– In step 2., $\mathsf{SimOpen}(pp'_{\mathsf{com}}, \tau_{\mathsf{com}}, s_i, (s^\star, r_i))$ is run to get randomness r'_i, and $(s_1, r'_1), \ldots, (s_\ell, r'_\ell)$ is sent to $\mathcal{A}$.

One can see that $\mathsf{Hyb}^{(zk)}_{\mathcal{A}}(b)$ is indistinguishable from $\mathsf{Hyb}^{(com)}_{\mathcal{A}}(b)$ for any PPT $\mathcal{A}$ and $b \in \{0,1\}$ due to the trapdoor property of commitments.

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

– Get group parameters (p, g, G) from Chal. Run $\mathsf{SimSetup}$ of TDC to get pp'_{com} and τ_{com}. Generate n commitments $\gamma'_1, \ldots, \gamma'_n$ to the fixed value $s^\star$ using randomness $r_1, \ldots, r_n$. Send $(p, g, G, \gamma'_1, \ldots, \gamma'_n, pp'_{\mathsf{com}})$ to $\mathcal{A}$.

– Get the set of corrupt parties $C = \{1, \ldots, \ell\}$ from $\mathcal{A}$. Pass it along to Chal and get back shares $s_1, \ldots, s_\ell$. Run $\mathsf{SimOpen}(pp'_{\mathsf{com}}, \tau_{\mathsf{com}}, s_i, (\vec{0}, r_i))$ to get randomness r'_i. Send $(s_1, r'_1), \ldots, (s_\ell, r'_\ell)$ to $\mathcal{A}$.

– On an evaluation query $(\texttt{Eval}, x, i)$ for an honest i from $\mathcal{A}$, send the same query to Chal and get back h_i. Compute $w := \mathcal{H}(x)$ and a simulated proof π'_i. Return $((w, h_i), \pi'_i)$ to $\mathcal{A}$.

– On the challenge query $(\texttt{Challenge}, x^\star, S, ((w, g^\star_1), \pi_1), \ldots, ((w, g^\star_u), \pi_u))$ from $\mathcal{A}$, if $\mathcal{A}$ has already made at least $t - |C|$ queries of the form $(\texttt{Eval}, x^\star, *)$ or one of the proofs does not verify, then output 0 and stop. Query Chal with $(\texttt{Challenge}, x^\star, S, g^\star_1, \ldots, g^\star_u)$ and get back z. Return z to $\mathcal{A}$.

– Continue answering evaluation queries as before, but if $\mathcal{A}$ makes a query of the form $(\texttt{Eval}, x^\star, i)$ for some $i \in [n] \setminus C$ and i is the g-th party it contacted, then output 0 and stop.

– Receive a guess b' from $\mathcal{A}$; output b'.

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

Correctness. To prove correctness of $\Pi_{\mathsf{ZK\text{-}DDH\text{-}DP}}$, we will exploit the extractibility property of proofs and the binding property of commitments. We first describe the correctness game (Definition 5.4) in detail. (The first three steps are same as that of $\mathsf{PseudoRand}'_{\mathcal{A}}(b)$ in Appendix C.5.)

1. Let G be a cyclic group of order p and g a generator of G. Sample $s \leftarrow_{\$} \mathbb{Z}_p$ and get $(s_1, \ldots, s_n) \leftarrow \mathsf{SSS}(n, t, p, s)$. Run $\mathsf{Setup}_{\mathsf{com}}(1^\kappa)$ to get pp_{com}. Compute a commitment $\gamma_i := \mathsf{Com}(s_i, pp_{\mathsf{com}}; r_i)$ by picking r_i at random. Send public parameters $pp = (p, g, G, \gamma_1, \ldots, \gamma_n, pp_{\mathsf{com}})$ to $\mathcal{A}$.

2. Get the set of corrupt parties C from $\mathcal{A}$. Without loss of generality assume that $C = \{1, \ldots \ell\}$. Send the corresponding secret keys $\{(s_1, r_1), \ldots, (s_\ell, r_\ell)\}$ to $\mathcal{A}$.

3. On an evaluation query $(\mathtt{Eval}, x, i)$ for an honest i, compute $w := H(x)$ and $h_i := w^{s_i}$. Run $\mathsf{Prove}^{\mathcal{H}'}$ with the statement stmt_i: $\{\exists s, r \text{ s.t. } h_i = w^s \wedge \gamma_i = \mathsf{Com}(s, pp_{\mathsf{com}}; r)\}$ and witness (s_i, r_i) to obtain a proof π_i. Return $((w, h_i), \pi_i)$ to $\mathcal{A}$.

4. On the challenge query $(\mathtt{Challenge}, x^\star, S, ((w, g_1^\star), \pi_1), \ldots, ((w, g_u^\star), \pi_u))$ for $u \leq \ell$ (without loss of generality assume that $S \cap C = [u]$):

    1. If $|S| < t$ or any of $\pi_1, \ldots, \pi_u$ do not verify, output 1.
    2. Else, compute $g_j := \mathcal{H}(x^\star)^{s_j}$ for $j \in S$, $g_i^\star := \mathcal{H}(x^\star)^{s_i}$ for $i \in S \setminus C$, $z := \prod_{i \in S} {g_i}^{\lambda_{0,i,S}}$ and $z^\star := \prod_{i \in S} {g_i^\star}^{\lambda_{0,i,S}}$. If $z^\star = z$, output 1.
    3. Else, output 0.

Suppose there exists an adversary $\mathcal{A}$ s.t. the correctness game reaches the very last step (3.), leading the challenger to output 0, with non-negligible probability. We will show that this leads to a contradiction. Towards this, we define a few intermediate hybrid games. In the first hybrid game, the hash function $\mathcal{H}'$ is replaced with the simulator $\mathcal{S}_1$ guaranteed by the zero-knowledge property of NIZK.

In the second hybrid, instead of producing a zero in the very last step (3.), the challenger:

– finds an $i^\star \in [u]$ s.t. $g_{i^\star}^\star \neq g_{i^\star}$ (such an $i^\star$ exists because $z^\star \neq z$);

– invokes the extractor $\mathcal{E}$ guaranteed by the argument of knowledge property on the adversary with inputs $(\mathsf{stmt}_{i^\star}, \pi_{i^\star}, Q)$ (Q is the list of queries made to $\mathcal{S}_1$ and their responses); and,

– outputs whatever the extractor does.

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

In the last hybrid, the challenger outputs $(s_{i^\star}, r_{i^\star})$ along with the extracted witness. Since $g^\star_{i^\star} \neq g_{i^\star}$, $s'_{i^\star} \neq s_{i^\star}$. Therefore, the challenger finds two distinct pairs $(s_{i^\star}, r_{i^\star})$ and $(s'_{i^\star}, r'_{i^\star})$ that produce the same committed value (with non-negligible probability). This breaks the binding property of TDC.

C.6 Proof of Theorem 8.4

First observe that the consistency property is trivially true. Next, we construct a PPT adversary $\mathcal{B}$ that breaks the security of PRF f if there exists a PPT adversary $\mathcal{A}$ that gains a non-negligible advantage in the pseudo-randomness game PseudoRand.

$\mathcal{B}$ gets access to $f_{k^\star}$ for a randomly chosen $k^\star$ in the PRF game. It can make an arbitrary polynomial number of queries to $f_{k^\star}$, and output a value $x^\star$ at some point. Then, $\mathcal{B}$ is supposed to distinguish between $f_{k^\star}(x^\star)$ and a random value from the range of f, provided that $x^\star$ was never queried. Let γ denote the PRF challenge.

Note that in PseudoRand adversary is allowed to query on the challenge input in the evaluation phase, but it must make less than $t - |C|$ queries, where C denotes the set of corrupt parties. Therefore, there is a set of size $n - t + 1$ honest parties that the adversary never queries. We denote this set by $D^\star$. In the following reduction, $\mathcal{B}$ will guess this set at the beginning of the game, and (implicitly) set the key for it to be $k^\star$. Recall that we have assumed that $d := \binom{n}{n-t+1}$ is polynomial in the security parameter. So we only suffer a polynomial loss in the security reduction.

$\mathcal{B}$ plays the role of challenger in PseudoRand with $\mathcal{A}$ as follows:

- *Initialization and corruption.* Define d and $D_1, \ldots, D_d$ in the same manner as Setup. Pick a subset $D^\star$ of $[n] \setminus C$ of size $n - t + 1$ at random. Implicitly set the key for $D^\star$ to be $k^\star$. Pick all the other PRF keys at random, use them to define $\{SK_j\}_{j \in C}$, and give $\{SK_j\}_{j \in C}$ to $\mathcal{A}$. Initialize list L and gap g as in PseudoRand. Add elements to L in the same way as well.

- *Pre-challenge evaluation queries.* Let $(\texttt{Eval}, x, i)$ be an evaluation query for some $i \in [n] \setminus C$. Send $h_{i,k} := f_k(x)$ for all $k \in SK_i$ to $\mathcal{A}$. When $k = k^\star$, query the PRF to get $f_{k^\star}(x)$.

- *Challenge.* Let $(\texttt{Challenge}, x^\star, S, \{(i, z^\star_i)\}_{i \in U})$ such that $|S| \geq t$ and $U \subseteq S \cap C$ be the challenge query. If $x^\star \in L$, output 0 and stop. Let V be the set of parties queried on $x^\star$ in the previous phase. If $V \cap D^\star \neq \emptyset$, then abort. Else, output $x^\star$ to the PRF challenger and get γ in return. For $i \in S \setminus U$, compute z_i by evaluating $h_{i,k} := f_k(x)$ for all $k \in SK_i$. Whenever $k = k^\star$ in these evaluations, use γ as the value for $f_{k^\star}(x^\star)$. Send $z^\star := \mathsf{Combine}(\{(i, z_i)\}_{i \in S \setminus U} \cup \{(i, z^\star_i)\}_{i \in U}, pp)$ to $\mathcal{A}$.

- *Post-challenge evaluation queries.* Same as before. If at any point $\mathcal{A}$ makes a query of type $(\texttt{Eval}, x^\star, i)$ for some $i \in [n] \setminus C$ so that $i \in D^\star$, then abort.

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

As long as $\mathcal{B}$ does not abort, it perfectly simulates PseudoRand for $\mathcal{A}$. If $\gamma = f_{k^\star}(x^\star)$, then $\mathcal{A}$ gets the view with $b = 0$, and if γ is random, then $\mathcal{A}$ gets the view with $b = 1$.