

# DiSE: Distributed Symmetric-key Encryption

Shashank Agrawal  
Visa Research  
Palo Alto, California  
shaagraw@visa.com

Pratyay Mukherjee  
Visa Research  
Palo Alto, California  
pratmukh@visa.com

Payman Mohassel  
Visa Research  
Palo Alto, California  
pmohasse@visa.com

Peter Rindal\*  
Oregon State University  
Corvallis, Oregon  
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.

## KEYWORDS

threshold cryptography; authenticated encryption; secret management systems; distributed pseudo-random functions

\*Work done as an intern at Visa Research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CCS '18, October 15–19, 2018, Toronto, ON, Canada

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-5693-0/18/10...\$15.00

<https://doi.org/10.1145/3243734.3243774>

## ACM Reference Format:

Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. 2018. DiSE: Distributed Symmetric-key Encryption. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*, October 15–19, 2018, Toronto, ON, Canada. ACM, New York, NY, USA, 18 pages. <https://doi.org/10.1145/3243734.3243774>

## 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 [2, 10, 11]. 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 [15, 21, 23, 24, 28, 32, 33, 35, 48, 49, 70] 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 [4]) 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 [12]:

"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 [6] 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 [5]) 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 [21, 24, 28, 34, 69]), 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 [19, 55, 65].

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 [17]), 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.

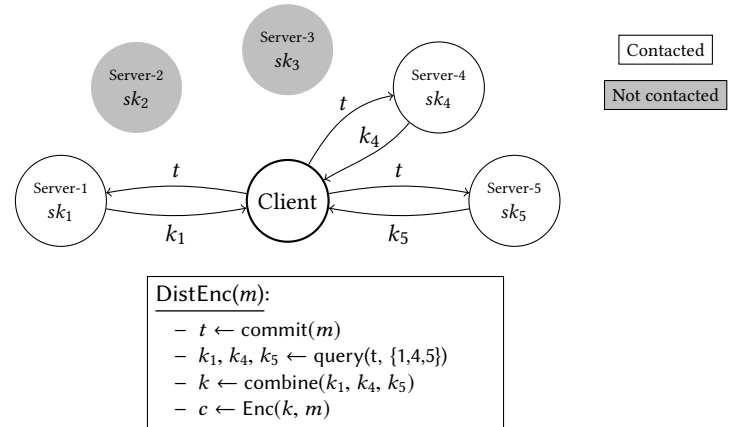
General-purpose multi-party computation protocols can also be used to solve the same problems by computing standard symmetric-key encryption schemes inside an MPC (e.g. see [2, 66]). While this approach has the benefit of preserving the original non-interactive algorithm, the resulting protocols would 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



**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.**

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. [60] 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).<sup>1</sup> 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. [60] 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

<sup>1</sup>This is in contrast with two-round MPC protocols (e.g. [58]) where typically in each round every participant broadcasts messages to everyone else.

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 [19, 55, 65]. We intentionally avoid the Universal Composability framework [27] because such definitions, proposed in prior work for standard symmetric-key encryption, are cumbersome to work with (e.g. see [56]).

We remark that over the past two decades, a large body of work has considered various notions of security for standard authenticated encryption [18–20, 26, 42, 52–54, 62–65, 65] 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.

*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  $sk_1, sk_2, \dots, sk_n$ , 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* [19, 55]. Alternatively, in the case of unified definitions [65], the adversary is restricted from querying the decryption oracle with a ciphertext it received from the encryption oracle<sup>2</sup>. 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.

## 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 the full version [16].

*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.<sup>3</sup> Naor et al. [60] 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  $c$ .

<sup>2</sup>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  $\perp$ .

<sup>3</sup>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.

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 [60].* The first (to the best of our knowledge) proposal for a distributed encryption is due to Naor et al. [60] (NPR in short). They propose to (i) first encrypt the message  $m$  locally to produce a ciphertext  $e = \text{SE}_w(m)$  by a “standard” symmetric-key encryption scheme  $\text{SE}$  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  $\text{AE}$  to compute  $e = \text{AE}_w(m)$ . 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  $\text{AE}$  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 the full version [16] 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<sup>4</sup> 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.

To summarize, our final construction can be informally described as follows: (i) the encryptor with identity  $j$  chooses a random  $\rho$  to compute  $\alpha := \text{Com}(m; \rho)$  where  $\text{Com}$  is a commitment and sends  $(j, \alpha)$  to the participating parties, (ii) the participating parties then first check if the message  $(j, \alpha)$  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  $e = \text{PRG}(w) \oplus (m||\rho)$  and outputs the ciphertext  $(j, \alpha, e)$ .

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 compromised, no information about the key is leaked. A popular key management tool called Vault [13] takes this approach. It uses Shamir’s secret sharing [68] to split the master secret key into *shards*. According to the documentation [14], “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<sup>5</sup>. 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 [28, 33–35, 43, 60, 69]. 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

<sup>4</sup>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.

<sup>5</sup>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.

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 [25, 36–39, 57, 60, 61]. 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. [60], which also proposes a mechanism for encrypting messages using their DPRF construction. But as we have discussed (c.f. Sec 2.1), 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 [3, 7, 8].

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<sup>6</sup> for all practical MPC instantiations; and the protocols require all pairs of parties to interact. Thus, a general-purpose MPC protocol for evaluating symmetric ciphers such as AES in any encryption mode [2, 31, 51, 66] 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  $\{1, 2, \dots, n\}$ . A function  $f : \mathbb{N} \rightarrow \mathbb{N}$  is negligible, denoted by  $\text{negl}$ , if for every polynomial  $p$ ,  $f(n) < 1/p(n)$  for all large enough values of  $n$ . We use  $D(x) := y$

<sup>6</sup>A recent surge of results [22, 44–47, 58] construct two round MPC protocols. However, these constructions focus mainly on generic feasibility and minimizing assumptions and are far from being practical.

or  $y := D(x)$  to denote that  $y$  is the output of the *deterministic* algorithm  $D$  on input  $x$ . Also,  $R(x) \rightarrow y$  or  $y \leftarrow R(x)$  denotes that  $y$  is the output of the *randomized* algorithm  $R$  on input  $x$ .  $R$  can be derandomized as  $R(x; r) := y$ , 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  $X \approx_{\text{comp}} Y$  to denote that they are computationally indistinguishable and  $X \approx_{\text{stat}} Y$  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  $\kappa$  to denote the security parameter. We make the natural identification between players and elements of  $\{1, \dots, n\}$ .

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  $\pi$ , 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  $[i : x_i]_{i \in S}$  or more compactly  $[x]_S$  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 [57]. 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 [36], extensive interaction [37, 61], consider only passive corruption [25, 60], or achieve stronger properties which makes them more expensive [38]. 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. [25]). 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.<sup>7</sup> We build on the constructions of Naor et al. [60] to obtain these properties from our DPRF instantiations.

Naor et al. [60], 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 [60], 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.

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

- $Setup(1^\kappa, n, t) \rightarrow ((sk_1, \dots, sk_n), pp)$ . The setup algorithm generates  $n$  secret keys  $(sk_1, sk_2, \dots, sk_n)$  and public parameters  $pp$ . The  $i$ -th secret key  $sk_i$  is given to party  $i$ .

- $Eval(sk_i, x, pp) \rightarrow z_i$ . 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  $pp$ .
- $Combine(\{(i, z_i)\}_{i \in S}, pp) =: z/\perp$ . The Combine algorithm combines the partial shares  $\{z_i\}_{i \in S}$  from parties in the set  $S$  to generate a value  $z$ . If the algorithm fails, its output is denoted by  $\perp$ .

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

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

where  $z_i \leftarrow Eval(sk_i, x, pp)$  for  $i \in S$ ,  $z'_j \leftarrow Eval(sk_j, x, pp)$  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  $DP := (Setup, Eval, Combine)$  is *pseudorandom* if for all PPT adversaries  $\mathcal{A}$ , there exists a negligible function  $\text{negl}$  such that

$$\left| \Pr[PseudoRand_{DP, \mathcal{A}}(1^\kappa, 0) = 1] - \Pr[PseudoRand_{DP, \mathcal{A}}(1^\kappa, 1) = 1] \right| \leq \text{negl}(\kappa),$$

where  $PseudoRand$  is defined below.

$PseudoRand_{DP, \mathcal{A}}(1^\kappa, b):$

- **Initialization.** Run  $Setup(1^\kappa, n, t)$  to get  $((sk_1, \dots, sk_n), pp)$ . 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  $(Eval, x, i)$  for some  $i \in [n] \setminus C$ , return  $Eval(sk_i, x, pp)$  to  $\mathcal{A}$ . Repeat this step as many times as  $\mathcal{A}$  desires.
- **Build the list.** Add an  $x$  to  $L$  if  $|\{i \mid \mathcal{A} \text{ made a } (Eval, x, i) \text{ query}\}| \geq g$ . In other words, if  $\mathcal{A}$  contacts at least  $g$  honest parties on a value  $x$ , it has enough information to compute the PRF output on  $x$ .
- **Challenge.**  $\mathcal{A}$  outputs  $(Challenge, x^*, S, \{(i, z_i^*)\}_{i \in U})$  such that  $|S| \geq t$  and  $U \subseteq S \cap C$ . If  $x^* \in L$ , output 0 and stop. Let  $z_i \leftarrow Eval(sk_i, x, pp)$  for  $i \in S \setminus U$  and  $z^* := Combine(\{(i, z_i)\}_{i \in S \setminus U} \cup \{(i, z_i^*)\}_{i \in U}, pp)$ . If  $z^* = \perp$ , return  $\perp$ . Else, if  $b = 0$ , return  $z^*$ ; otherwise, return a uniformly random value.

<sup>7</sup>This is a weaker requirement than robustness for DPRFs which guarantees that an honest party *will* receive the correct DPRF value. However, Dodis [36], 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.



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

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

- *Initialization.* Run  $\text{Setup}(1^\kappa, n, t)$  to get  $((sk_1, \dots, sk_n), pp)$ . 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  $(\text{Eval}, x, i)$  for some  $i \in [n] \setminus C$ , return  $\text{Eval}(sk_i, x, pp)$  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  $x^*$ , and shares  $\{(i, z_i^*)\}_{i \in S \cap C}$  from  $\mathcal{A}$ . Let  $z_j \leftarrow \text{Eval}(sk_j, x^*, pp)$  for  $j \in S$  and  $z'_i \leftarrow \text{Eval}(sk_i, x^*, pp)$  for  $i \in S \setminus C$ . Also, let  $z := \text{Combine}(\{(j, z_j)\}_{j \in S}, pp)$  and  $z^* := \text{Combine}(\{(i, z'_i)\}_{i \in S \setminus C} \cup \{(i, z_i^*)\}_{i \in S \cap C}, pp)$ . Output 1 if  $z^* \in \{z, \perp\}$ ; 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  $(\text{Setup}, \text{DistEnc}, \text{DistDec})$  that satisfies the consistency property below.

- $\text{Setup}(1^\kappa, n, t) \rightarrow ([\text{sk}]_{[n]}, pp)$ : Setup is a randomized algorithm that takes the security parameter as input, and outputs  $n$  secret keys  $sk_1, \dots, sk_n$  and public parameters  $pp$ . The  $i$ -th secret key  $sk_i$  is given to party  $i$ .
- $\text{DistEnc}([\text{sk}]_{[n]}, [j : m, S], pp) \rightarrow [j : c/\perp]$ : 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.
- $\text{DistDec}([\text{sk}]_{[n]}, [j : c, S], pp) \rightarrow [j : m/\perp]$ : 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  $([\text{sk}]_{[n]}, pp)$  output by  $\text{Setup}(1^\kappa)$ , 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  $\text{negl}$  such that

$$\Pr \left[ [j' : m] \leftarrow \text{DistDec}([\text{sk}]_{[n]}, [j' : c, S'], pp) \mid [j : c] \leftarrow \text{DistEnc}([\text{sk}]_{[n]}, [j : m, S], pp) \right] \geq 1 - \text{negl}(\kappa),$$

where the probability is over the random coin tosses of the parties involved in DistEnc and 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 DistEnc outputs a ciphertext  $c$  for an input message  $m$  (i.e., it does not fail), then 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  $\text{TSE} := (\text{Setup}, \text{DistEnc}, \text{DistDec})$  is *correct* if for all PPT adversaries  $\mathcal{A}$ , there exists a negligible function  $\text{negl}$  such that the following game outputs 1 with probability at least  $1 - \text{negl}(\kappa)$ .

- *Initialization.* Run  $\text{Setup}(1^\kappa)$  to get  $([\text{sk}]_{[n]}, pp)$ . Give  $pp$  to  $\mathcal{A}$ .
- *Corruption.* Receive the set of corrupt parties  $C$  from  $\mathcal{A}$ , where  $|C| < t$ . Give the secret-keys  $\{sk_i\}_{i \in C}$  of these parties to  $\mathcal{A}$ .
- *Encryption.* Receive  $(\text{Encrypt}, j, m, S)$  from  $\mathcal{A}$  where  $j \in S \setminus C$  and  $|S| \geq t$ . Initiate the protocol 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  $(\text{Decrypt}, j', S')$  from  $\mathcal{A}$  where  $j' \in S' \setminus C$  and  $|S'| \geq t$ . Initiate the protocol  $\text{DistDec}$  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  $\text{TSE} := (\text{Setup}, \text{DistEnc}, \text{DistDec})$  satisfies *message privacy* if for all PPT adversaries  $\mathcal{A}$ , there exists a negligible function  $\text{negl}$  such that

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

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

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

- *Initialization*. Run  $\text{Setup}(1^\kappa, n, t)$  to get  $(\llbracket \text{sk} \rrbracket_n, pp)$ . 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  $(\text{Encrypt}, j, m, S)$ , where  $j \in S$  and  $|S| \geq t$ , run an instance of the protocol  $\text{DistEnc}$  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  $(\text{Decrypt}, j, c, S)$ , where  $j \in S \setminus C$  and  $|S| \geq t$ , party  $j$  initiates  $\text{DistDec}$  with inputs  $c$  and  $S$ . Repeat this step as many times as  $\mathcal{A}$  desires.
- *Challenge*.  $\mathcal{A}$  outputs  $(\text{Challenge}, j^*, m_0, m_1, S^*)$  where  $|m_0| = |m_1|$ ,  $j^* \in S^* \setminus C$  and  $|S^*| \geq t$ . Initiate the protocol  $\text{DistEnc}$  from party  $j^*$  with inputs  $m_b$  and  $S^*$ . Give  $c^*$  (or  $\perp$ ) output by  $j^*$  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  $\text{DistEnc}$  is run in the challenge phase with  $S^* \cap C \neq \emptyset$ , corrupt parties can easily cause the protocol to fail, leading  $j^*$  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  $\lfloor ct/g \rfloor$  ciphertexts at the end of the game.

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  $\text{TSE} := (\text{Setup}, \text{DistEnc}, \text{DistDec})$  satisfies *authenticity* if for all PPT adversaries  $\mathcal{A}$ , there

<sup>8</sup>Note that  $j$  can be either honest or corrupt here. So both types of encryption queries are captured.

exists a negligible function  $\text{negl}$  such that

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

where AUTH is defined below.

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

- *Initialization.* Run  $\text{Setup}(1^\kappa, n, t)$  to get  $(\llbracket \text{sk} \rrbracket_{[n]}, pp)$ . Give  $pp$  to  $\mathcal{A}$ . Initialize a counter  $ct := 0$  and an ordered list  $L_{\text{ctxt}} := \emptyset$ . 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  $(\text{Encrypt}, j, m, S)$  from  $\mathcal{A}$ , run the protocol  $\text{DistEnc}$  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  $L_{\text{ctxt}}$ .
- *Decryption queries.* On receiving  $(\text{Decrypt}, j, c, S)$  from  $\mathcal{A}$ , run the protocol  $\text{DistDec}$  with  $c, S$  as the inputs of  $j$ . If  $j \in C$ , increment  $ct$  by  $|S \setminus C|$ .
- *Targeted decryption queries.* On receiving  $(\text{TargetDecrypt}, j, \ell, S)$  from  $\mathcal{A}$  for some  $j \in S \setminus C$ , run  $\text{DistDec}$  with  $c, S$  as the inputs of  $j$ , where  $c$  is the  $\ell$ -th ciphertext in  $L_{\text{ctxt}}$ .
- *Forgery.* Let  $k := \lfloor ct/g \rfloor$ .  $\mathcal{A}$  outputs  $((j_1, S_1, c_1), (j_2, S_2, c_2), \dots, (j_{k+1}, S_{k+1}, c_{k+1}))$  such that  $j_1, \dots, j_{k+1} \notin C$  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  $\text{DistDec}$  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) [19], 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  $\ell$  forgeries would be accepted only if they decrypt to  $\ell$  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  $j \parallel \alpha$ , 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.

Ingredients:	
<ul style="list-style-type: none"> <li>– An <math>(n, t)</math>-DPRF protocol <math>\text{DP} := (\text{DP.Setup}, \text{Eval}, \text{Combine})</math> (Def. 5.1).</li> <li>– A pseudorandom generator <math>\text{PRG}</math> of polynomial stretch.</li> <li>– A commitment scheme <math>\Sigma := (\Sigma.\text{Setup}, \text{Com})</math> (Def. A.2).</li> </ul>	
$\text{Setup}(1^\kappa, n, t) \rightarrow (\llbracket \text{sk} \rrbracket_{[n]}, pp)$ :	Run $\text{DP.Setup}(1^\kappa, n, t)$ to get $((rk_1, \dots, rk_n), pp_{\text{DP}})$ and $\Sigma.\text{Setup}(1^\kappa)$ to get $pp_{\text{com}}$ . Set $sk_i := rk_i$ for $i \in [n]$ and $pp := (pp_{\text{DP}}, pp_{\text{com}})$ .
$\text{DistEnc}(\llbracket \text{sk} \rrbracket_{[n]}, [j : m, S], pp) \rightarrow [j : c/\perp]$ :	To encrypt a message $m$ with the help of parties in $S$ : <ul style="list-style-type: none"> <li>– Party <math>j</math> computes <math>\alpha := \text{Com}(m, pp_{\text{com}}; \rho)</math> for a randomly chosen <math>\rho</math> and sends <math>\alpha</math> to all parties in <math>S</math>.</li> <li>– For every <math>i \in S</math>, party <math>i</math> runs <math>\text{Eval}(sk_i, j \parallel \alpha, pp)</math> to get <math>z_i</math>, and sends it to party <math>j</math>.</li> <li>– Party <math>j</math> runs <math>\text{Combine}(\{(i, z_i)\}_{i \in S}, pp)</math> to get <math>w</math> or <math>\perp</math>. In the latter case, it outputs <math>\perp</math>. Otherwise, it computes <math>e := \text{PRG}(w) \oplus (m \parallel \rho)</math> and then outputs <math>c := (j, \alpha, e)</math>.</li> </ul>
$\text{DistDec}(\llbracket \text{sk} \rrbracket_{[n]}, [j' : c, S], pp) \rightarrow [j' : m/\perp]$ :	To decrypt a ciphertext $c$ with the help of parties in $S$ : <ul style="list-style-type: none"> <li>– Party <math>j'</math> first parses <math>c</math> into <math>(j, \alpha, e)</math>. Then it sends <math>j \parallel \alpha</math> to all the parties in <math>S</math>.</li> <li>– For <math>i \in S</math>, party <math>i</math> receives <math>x</math> and checks if it is of the form <math>j^* \parallel \alpha^*</math> for some <math>j^* \in [n]</math>. If not, then it sends <math>\perp</math> to party <math>j'</math>. Else, it runs <math>\text{Eval}(sk_i, x, pp)</math> to get <math>z_i</math>, and sends it to party <math>j'</math>.</li> <li>– Party <math>j'</math> runs <math>\text{Combine}(\{(i, z_i)\}_{i \in S}, pp)</math> to get <math>w</math> or <math>\perp</math>. In the latter case, it outputs <math>\perp</math>. Otherwise, it computes <math>m \parallel \rho := \text{PRG}(w) \oplus e</math> and checks if <math>\alpha = \text{Com}(m, pp_{\text{com}}; \rho)</math>. If the check succeeds, it outputs <math>m</math>; otherwise, it outputs <math>\perp</math>.</li> </ul>

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

**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.

**LEMMA 7.2 (CORRECTNESS).** *DiSE is a correct TSE scheme.*

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

**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  $e := \text{PRG}(w) \oplus (m \parallel \rho)$ , 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 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  $j \parallel \alpha$ ). Therefore,  $\text{PRG}(w') \oplus e$  in the last step of decryption would give  $\text{PRG}(w') \oplus \text{PRG}(w) \oplus (m \parallel \rho) = m \parallel \rho$ .  $\square$

For the following three lemma, we provide a sketch here and defer formal proofs to the full version [16].

**LEMMA 7.4 (MESSAGE PRIVACY).** *If DP is a secure DPRF, then DiSE is a message-private TSE scheme.*

**PROOF SKETCH.** The challenge ciphertext  $c^*$  has the form  $(j^*, \alpha^*, e^*)$  where  $e^* = \text{PRG}(w^*) \oplus (m_b \parallel \rho^*)$ ,  $\alpha^* = \text{Com}(m_b, pp_{\text{com}}; \rho^*)$  and  $w^*$  is the output of DPRF DP on  $j^* \parallel \alpha^*$ . 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  $w^*$  is indistinguishable from random. The pseudorandomness property of DP ensures this as long as  $\mathcal{A}$  has no way of evaluating the DPRF on  $j^* \parallel \alpha^*$  itself. (Note that  $\alpha^*$  does not reveal information about  $m_b$  due to the hiding property of  $\Sigma$ .)

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

**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  $\alpha$ . 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,  $(m_1, \rho_1)$  and  $(m_2, \rho_2)$  recovered from  $c_1$  and  $c_2$ , respectively, would be different. Due to the binding property of  $\Sigma$ ,  $\alpha$  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  $(j, \alpha)$ .

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  $(j, \alpha)$  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  $(j, \alpha)$ . Hence, it can not produce a valid ciphertext with it.

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).  $\square$

**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.  $\square$

**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 [13]. 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.*

## 8 DPRF INSTANTIATIONS

In this section, we revisit the distributed pseudo-random function (DPRF) constructions of Naor, Pinkas, and Reingold [60] (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 [30, 67] via the Fiat-Shamir transform [41]) 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  $H : \{0, 1\}^* \rightarrow G$  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  $s_1, \dots, s_n$  of  $s$ . Share  $s_i$  is given privately to the party  $i$ . We know that for any set  $S$  of  $\ell \geq t$  parties  $S := \{i_1, \dots, i_\ell\} \subseteq [n]$ , there exists integers (i.e. Lagrange coefficients)  $\lambda_{0,1,S}, \dots, \lambda_{0,\ell,S} \in \mathbb{Z}_p$  such that  $\sum_{j \in S} s_{i_j} \lambda_{0,j,S} = s$ . Therefore, it holds that

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

which can be computed in a distributed manner running the protocol  $\Pi_{\text{DDH-DP}}$  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  $\mathcal{H} : \{0, 1\}^* \rightarrow G$  be a hash function modeled as a random oracle. Let SSS be Shamir's secret sharing scheme (Def. A.5).

- Setup( $1^\kappa, n, t$ )  $\rightarrow (\llbracket \text{sk} \rrbracket_{[n]}, pp)$ : Sample  $s \leftarrow_{\$} \mathbb{Z}_p$  and get  $(s_1, \dots, s_n) \leftarrow \text{SSS}(n, t, p, s)$ . Set  $pp := (p, g, G)$  and  $sk_i := s_i$  and give  $(sk_i, pp)$  to party  $i$ , for  $i \in [n]$ .
- Eval( $sk_i, x, pp$ )  $\rightarrow z_i$ : Compute  $w := \mathcal{H}(x)$ ,  $h_i := w^{sk_i}$  and output  $h_i$ .
- 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  $\Pi_{\text{DDH-DP}}$  based on DDH.**

**THEOREM 8.1.** *Protocol  $\Pi_{\text{DDH-DP}}$  in Figure 3 is a secure DPRF under the DDH assumption in ROM.*

We defer the proof to the full version [16].

*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  $\Pi_{\text{DDH-DP}}$ , we obtain the protocol  $\Pi_{\text{ZK-DDH-DP}}$ , described in detail in Figure 4. This protocol also satisfies correctness and hence achieves *strong security*. Formally:

**THEOREM 8.2.** *Protocol  $\Pi_{\text{ZK-DDH-DP}}$  in Figure 4 is a strongly secure DPRF under the DDH assumption in ROM.*

We defer the proof to the full version [16].

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

- Setup( $1^\kappa, n, t$ )  $\rightarrow (\llbracket \text{sk} \rrbracket_{[n]}, pp)$ . Sample  $s \leftarrow_{\$} \mathbb{Z}_p$  and get  $(s_1, \dots, s_n) \leftarrow \text{SSS}(n, t, p, s)$ . Run Setup $_{\text{com}}(1^\kappa)$  to get  $pp_{\text{com}}$ . Compute a commitment  $\gamma_i := \text{Com}(s_i, pp_{\text{com}}; r_i)$  by picking  $r_i$  at random. Set  $pp = (p, g, G, \gamma_1, \dots, \gamma_n, pp_{\text{com}})$ ,  $sk_i := (s_i, r_i)$  and give  $sk_i$  to party  $i$ , for  $i \in [n]$ .
- Eval( $sk_i, x, pp$ )  $\rightarrow z_i$ . Compute  $w := H(x)$  and  $h_i := w^{s_i}$ . Run Prove $^{\mathcal{H}'}$  with the statement  $\text{stmt}_i : \{\exists s, r \text{ s.t. } h_i = w^s \wedge \gamma_i = \text{Com}(s, pp_{\text{com}}; r)\}$  and witness  $(s_i, r_i)$  to obtain a proof  $\pi_i$ . Output  $((w, h_i), \pi_i)$ .
- 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  $\text{Verify}^{\mathcal{H}'}(\text{stmt}_i, \pi_i) = 1$  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  $\Pi_{\text{ZK-DDH-DP}}$  based on DDH. Differences from  $\Pi_{\text{DDH-DP}}$  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,  $\mathcal{H} : \{0, 1\}^* \rightarrow G$  and  $\mathcal{H}' : \{0, 1\}^* \rightarrow \mathbb{Z}_p$  be hash functions. Let SSS be Shamir's secret sharing scheme (Def. A.5).

- Setup( $1^\kappa, n, t$ )  $\rightarrow (\llbracket \text{sk} \rrbracket_{[n]}, pp)$ . Sample  $s \leftarrow_{\$} \mathbb{Z}_p$  and get  $(s_1, \dots, s_n) \leftarrow \text{SSS}(n, t, p, s)$ . 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  $pp = (p, g, G, \mathcal{H}, \mathcal{H}', \gamma_1, \dots, \gamma_n, h)$ ,  $sk_i := (s_i, r_i)$  and give  $sk_i$  to party  $i$ , for  $i \in [n]$ .
- Eval( $sk_i, x, pp$ )  $\rightarrow z_i$ . 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  $\pi_i$  to be  $(c_i, u_i, u'_i)$  and output  $((w, h_i), \pi_i)$ .
- 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  $\Pi_{\text{ZK-DDH-DP}}$  from Figure 4 using Pedersen commitments and Schnorr-style proof (via the Fiat-Shamir transform).**

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  $\Pi_{\text{ZK-DDH-DP}}$  in Figure 5 and use it for our experiments in the following section. The concrete protocol remains secure under DDH (in random oracle model).

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

- $\text{Setup}(1^\kappa, n, t) \rightarrow (\llbracket \text{sk} \rrbracket_{[n]}, pp)$ . Sample  $s \leftarrow_{\$} \mathbb{Z}_p$  and get  $(s_1, \dots, s_n) \leftarrow \text{SSS}(n, t, p, s)$ . Set  $pp = (p, g, G, \mathcal{H}, \mathcal{H}', g^{s_1}, \dots, g^{s_n})$ ,  $sk_i := s_i$  and give  $sk_i$  to party  $i$ , for  $i \in [n]$ .
- $\text{Eval}(sk_i, x, pp) \rightarrow z_i$ . 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  $\pi_i$  to be  $(c_i, u_i)$  and output  $((w, h_i), \pi_i)$ .
- $\text{Combine}(\{(i, z_i)\}_{i \in S}, pp) =: z/\perp$ . 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.**

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.<sup>9</sup> 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  $k_1, \dots, k_d$  are chosen. We assume that  $d$  is polynomial in the security parameter so that all the DPRF algorithms are polynomial time. Let  $D_1, \dots, D_d$  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  $D_1, \dots, D_d$  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  $\Pi_{f\text{-DP}}$  in Figure 7 is a secure DPRF.*

<sup>9</sup>Micali and Sydney provided a similar construction but for more general access structures [57].

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

- $\text{Setup}(1^\kappa, n, t) \rightarrow (\llbracket \text{SK} \rrbracket_{[n]}, pp)$ : Pick  $k_1, \dots, k_d \leftarrow_{\$} \{0, 1\}^\kappa$  where  $d := \binom{n}{n-t+1}$ . Let  $D_1, \dots, D_d$  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  $i \in [n]$ .
- $\text{Eval}(SK_i, x, pp) \rightarrow z_i$ : Compute  $h_{i,k} := f_k(x)$  for all  $k \in SK_i$  and output  $\{h_{i,k}\}_{k \in SK_i}$ .
- $\text{Combine}(\{(i, z_i)\}_{i \in S}, pp) =: z/\perp$ : 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  $\bigcup_{i \in S} SK'_i = \{k_1, \dots, k_d\}$  and  $SK'_i \subseteq SK_i$  for every  $i$ . Output  $\bigoplus_{k \in SK'_i, i \in S} h_{i,k}$ .

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

We defer the formal proof to the full version [16]. 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 [29, 50]. 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
$\Pi_{f\text{-DP}}$ (Fig. 7)	$\Gamma_{\text{AES}}$	OWF	Standard
$\Pi_{\text{DDH-DP}}$ (Fig. 3)	$\Gamma_{\text{DDH}}$	DDH	Standard
$\Pi_{\text{ZK-DDH-DP}}$ (Fig. 5)	$\Gamma_{\text{DDH}}^S$ (Strong)	DDH	ROM
$\Pi_{\text{ZK-DDH-DP}}$ (Fig. 6)	$\Gamma_{\text{DDH}}^{\text{PV}}$ (Strong)	DDH	ROM

**Figure 8: The four TSE schemes we implemented by instantiating DiSE. There are two concrete instantiations of  $\Pi_{\text{ZK-DDH-DP}}$ , 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 [9] and the PRF/PRGs are constructed from AES-NI. The DDH-based DPRF [60] uses the Miracl library [1] 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.

**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  $\Gamma_{\text{AES}}$  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  $t = 4$ ,  $\Gamma_{\text{AES}}$  achieves 1

$t$	$n$	Throughput (enc/s)				Latency (ms/enc)				Bandwidth (Throughput Mbps)			
		$\Gamma_{\text{AES}}$	$\Gamma_{\text{DDH}}$	$\Gamma_{\text{DDH}}^S$	$\Gamma_{\text{DDH}}^{\text{PV}}$	$\Gamma_{\text{AES}}$	$\Gamma_{\text{DDH}}$	$\Gamma_{\text{DDH}}^S$	$\Gamma_{\text{DDH}}^{\text{PV}}$	$\Gamma_{\text{AES}}$	$\Gamma_{\text{DDH}}$	$\Gamma_{\text{DDH}}^S$	$\Gamma_{\text{DDH}}^{\text{PV}}$
$n/3$	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)			
		$\Gamma_{\text{AES}}$	$\Gamma_{\text{DDH}}$	$\Gamma_{\text{DDH}}^S$	$\Gamma_{\text{DDH}}^{\text{PV}}$	$\Gamma_{\text{AES}}$	$\Gamma_{\text{DDH}}$	$\Gamma_{\text{DDH}}^S$	$\Gamma_{\text{DDH}}^{\text{PV}}$	$\Gamma_{\text{AES}}$	$\Gamma_{\text{DDH}}$	$\Gamma_{\text{DDH}}^S$	$\Gamma_{\text{DDH}}^{\text{PV}}$
$n/3$	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
$n/2$	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
$2n/3$	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  $t$ .**

million encryption per second while  $\Gamma_{\text{DDH}}$ , the next fastest, is  $2000\times$  slower with 556 encryptions. Increasing the parameters to  $n = 24, t = 16$ ,  $\Gamma_{\text{AES}}$  achieves 902 encryptions per second while  $\Gamma_{\text{DDH}}$  is still  $5\times$  slower with 173 encryptions. The protocol  $\Gamma_{\text{DDH}}^S$  which achieves strong correctness incurs a 2 to  $5\times$  overhead compared to the weaker  $\Gamma_{\text{DDH}}$  while the publicly verifiable variant  $\Gamma_{\text{DDH}}^{\text{PV}}$  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.  $\Gamma_{\text{AES}}$  achieves sub-millisecond latency for most configurations. On the other hand,  $\Gamma_{\text{DDH}}^{\text{PV}}$  with its strong security guarantees achieves a latency between 10 and 100ms.

**Communication.** In addition to achieving the best throughput and latency, the  $\Gamma_{\text{AES}}$  protocol has the smallest communication overhead of  $32(t-1)$  bytes per encryption. The  $\Gamma_{\text{DDH}}$  incurs slightly more communication with  $49(t-1)$  bytes per encryption while  $\Gamma_{\text{DDH}}^S$  and  $\Gamma_{\text{DDH}}^{\text{PV}}$  have the most communication with  $148(t-1)$  bytes.

However, despite having comparable communication overheads, the pure symmetric-key  $\Gamma_{\text{AES}}$  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  $\Gamma_{\text{DDH}}$ ,  $\Gamma_{\text{DDH}}^S$  and  $\Gamma_{\text{DDH}}^{\text{PV}}$  is that the key size is either constant (33 bytes) or linear in the threshold ( $33t$  bytes). The  $\Gamma_{\text{AES}}$  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  $\Gamma_{\text{AES}}$  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  $\Gamma_{\text{AES}}$  protocol due to it easily saturating the network. With  $n = 6$ ,  $t = 2$ , we observe that the throughput drops  $7\times$  to 153,332 encryption per second. However, this is near optimal given that simply communicating  $\kappa$  bits requires 37 out of the 40Mbps bandwidth limit. Additionally, the latency of the  $\Gamma_{\text{AES}}$  is near the optimal of 80ms in most cases. The  $\Gamma_{\text{DDH}}$  protocol require slightly more time of roughly 90ms in most cases while the strongly-correct  $\Gamma_{\text{DDH}}^{\text{PV}}$  and  $\Gamma_{\text{DDH}}^{\text{S}}$  protocols require between 95 and 170ms.

## REFERENCES

- [1] BLAKE2 - fast secure hashing. <https://blake2.net/>.
- [2] Dyadic Security. <https://www.dyadicsec.com>.
- [3] ePrint Archive MPC Papers. <http://users-cs.au.dk/psn/list/>.
- [4] Infrastructure Secret Management Software Overview. <https://gist.github.com/maxvt/bb49a6c7243163b8120625fc8ae3f3cd>.
- [5] Kerberos: Encryption Types. <http://web.mit.edu/kerberos/krb5-latest/doc/admin/encetypes.html>.
- [6] Kerberos: The Network Authentication Protocol. <http://web.mit.edu/kerberos/>.
- [7] List of MPC Software. <http://www.multipartycomputation.com/mpc-software>.
- [8] List of resources on MPC. <https://github.com/rdragos/awesome-mpc>.
- [9] MIRACL Cryptographic SDK. <https://www.miracl.com/>.
- [10] Porticor Cloud Security. <http://www.porticor.com/>. Acquired by Intuit.
- [11] Sepior. <https://sepior.com>.
- [12] Vault Architecture. <https://www.vaultproject.io/docs/internals/architecture.html>.
- [13] Vault by HashiCorp. <https://www.vaultproject.io/>.
- [14] Vault Docs, Basic Concepts, Seal-Unseal. <https://www.vaultproject.io/docs/concepts/seal.html>.
- [15] M. Abdalla, S. Miner, and C. Namprempre. Forward-secure threshold signature schemes. In *Cryptographers Track at the RSA Conference*, pages 441–456. Springer, 2001.
- [16] S. Agrawal, P. Mohassel, P. Mukherjee, and P. Rindal. DisE: Distributed symmetric-key encryption. Cryptology ePrint Archive, Report 2018/727, 2018. <https://eprint.iacr.org/2018/727>.
- [17] M. Bellare, O. Goldreich, and A. 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>.
- [18] M. Bellare and S. Keelveedhi. Authenticated and misuse-resistant encryption of key-dependent data. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of LNCS, pages 610–629. Springer, Heidelberg, Aug. 2011.
- [19] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of LNCS, pages 531–545. Springer, Heidelberg, Dec. 2000.
- [20] M. Bellare and B. Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of LNCS, pages 247–276. Springer, Heidelberg, Aug. 2016.
- [21] R. Bendlin and I. Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In D. Micciancio, editor, *TCC 2010*, volume 5978 of LNCS, pages 201–218. Springer, Heidelberg, Feb. 2010.
- [22] F. Benhamouda and H. Lin.  $k$ -round multiparty computation from  $k$ -round oblivious transfer via garbled interactive circuits. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of LNCS, pages 500–532. Springer, Heidelberg, Apr. / May 2018.
- [23] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of LNCS, pages 31–46. Springer, Heidelberg, Jan. 2003.
- [24] D. Boneh, X. Boyen, and S. Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In D. Pointcheval, editor, *CT-RSA 2006*, volume 3860 of LNCS, pages 226–243. Springer, Heidelberg, Feb. 2006.
- [25] D. Boneh, K. Lewi, H. W. Montgomery, and A. Raghunathan. Key homomorphic PRFs and their applications. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of LNCS, pages 410–428. Springer, Heidelberg, Aug. 2013.
- [26] P. Bose, V. T. Hoang, and S. Tessaro. Revisiting AES-GCM-SIV: Multi-user security, faster key derivation, and better bounds. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of LNCS, pages 468–499. Springer, Heidelberg, Apr. / May 2018.
- [27] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.
- [28] R. Canetti and S. Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of LNCS, pages 90–106. Springer, Heidelberg, May 1999.
- [29] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. 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.
- [30] D. Chaum and H. Van Antwerpen. Undeniable signatures. In G. Brassard, editor, *CRYPTO'89*, volume 435 of LNCS, pages 212–216. Springer, Heidelberg, Aug. 1990.
- [31] I. Damgård and M. Keller. Secure multiparty AES. In R. Sion, editor, *FC 2010*, volume 6052 of LNCS, pages 367–374. Springer, Heidelberg, Jan. 2010.
- [32] I. Damgård and M. Kopprowski. Practical threshold RSA signatures without a trusted dealer. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of LNCS, pages 152–165. Springer, Heidelberg, May 2001.
- [33] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *26th ACM STOC*, pages 522–533. ACM Press, May 1994.
- [34] C. Delerablée and D. Pointcheval. Dynamic threshold public-key encryption. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of LNCS, pages 317–334. Springer, Heidelberg, Aug. 2008.
- [35] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *CRYPTO'89*, volume 435 of LNCS, pages 307–315. Springer, Heidelberg, Aug. 1990.
- [36] Y. Dodis. Efficient construction of (distributed) verifiable random functions. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of LNCS, pages 1–17. Springer, Heidelberg, Jan. 2003.
- [37] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In S. Vaudenay, editor, *PKC 2005*, volume 3386 of LNCS, pages 416–431. Springer, Heidelberg, Jan. 2005.
- [38] Y. Dodis, A. Yampolskiy, and M. Yung. Threshold and proactive pseudo-random permutations. In S. Halevi and T. Rabin, editors, *TCC 2006*, volume 3876 of LNCS, pages 542–560. Springer, Heidelberg, Mar. 2006.
- [39] A. Everspaugh, R. Chatterjee, S. Scott, A. Juels, and T. Ristenpart. The pythia PRF service. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 547–562, 2015.
- [40] S. Faust, M. Kohlweiss, G. A. Marson, and D. Venturi. On the non-malleability of the Fiat-Shamir transform. In S. D. Galbraith and M. Nandi, editors, *INDOCRYPT 2012*, volume 7668 of LNCS, pages 60–79. Springer, Heidelberg, Dec. 2012.
- [41] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *CRYPTO'86*, volume 263 of LNCS, pages 186–194. Springer, Heidelberg, Aug. 1987.
- [42] E. Fleischmann, C. Forler, and S. Lucks. McOE: A family of almost foolproof on-line authenticated encryption schemes. In A. Canteaut, editor, *FSE 2012*, volume 7549 of LNCS, pages 196–215. Springer, Heidelberg, Mar. 2012.
- [43] Y. Frankel. A practical protocol for large group oriented networks. In J.-J. Quisquater and J. Vandewalle, editors, *EUROCRYPT'89*, volume 434 of LNCS, pages 56–61. Springer, Heidelberg, Apr. 1990.
- [44] S. Garg, C. Gentry, S. Halevi, and M. Raykova. Two-round secure MPC from indistinguishability obfuscation. In Y. Lindell, editor, *TCC 2014*, volume 8349 of LNCS, pages 74–94. Springer, Heidelberg, Feb. 2014.
- [45] S. Garg, P. Mukherjee, O. Pandey, and A. Polychroniadou. The exact round complexity of secure computation. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of LNCS, pages 448–476. Springer, Heidelberg, May 2016.
- [46] S. Garg and A. Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In *58th FOCS*, pages 588–599. IEEE Computer Society Press, 2017.
- [47] S. Garg and A. Srinivasan. Two-round multiparty secure computation from minimal assumptions. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of LNCS, pages 468–499. Springer, Heidelberg, Apr. / May 2018.
- [48] R. Gennaro, S. Halevi, H. Krawczyk, and T. Rabin. Threshold RSA for dynamic and ad-hoc groups. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of LNCS, pages 88–107. Springer, Heidelberg, Apr. 2008.
- [49] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In U. M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of LNCS, pages 354–371. Springer, Heidelberg, May 1996.
- [50] I. Giacomelli, J. Madsen, and C. Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security Symposium*, pages 1069–1083, 2016.
- [51] L. Grassi, C. Rechberger, D. Rotaru, P. Scholl, and N. P. Smart. MPC-friendly symmetric key primitives. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 16*, pages 430–443. ACM Press, Oct. 2016.
- [52] S. Gueron and Y. Lindell. GCM-SIV: Full nonce misuse-resistant authenticated encryption at under one cycle per byte. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 15*, pages 109–119. ACM Press, Oct. 2015.
- [53] V. T. Hoang, T. Krovetz, and P. Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of LNCS, pages 15–44. Springer, Heidelberg, Apr. 2015.
- [54] V. T. Hoang, R. Reyhanitabar, P. Rogaway, and D. Vízár. Online authenticated-encryption and its nonce-reuse misuse-resistance. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of LNCS, pages 493–517. Springer, Heidelberg, Aug. 2015.
- [55] J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In B. Schneier, editor, *FSE 2000*, volume 1978 of LNCS, pages 284–299.



- Springer, Heidelberg, Apr. 2001.
- [56] R. Kusters and M. Tuengerthal. Universally composable symmetric encryption. In *2009 22nd IEEE Computer Security Foundations Symposium*, pages 293–307, July 2009.
  - [57] S. Micali and R. Sidney. A simple method for generating and sharing pseudo-random functions, with applications to clipper-like escrow systems. In D. Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 185–196. Springer, Heidelberg, Aug. 1995.
  - [58] P. Mukherjee and D. Wichs. Two round multiparty computation via multi-key FHE. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.
  - [59] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.
  - [60] M. Naor, B. Pinkas, and O. Reingold. Distributed pseudo-random functions and KDCs. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 327–346. Springer, Heidelberg, May 1999.
  - [61] J. B. Nielsen. A threshold pseudorandom function construction and its applications. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 401–416. Springer, Heidelberg, Aug. 2002.
  - [62] K. G. Paterson and G. 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.
  - [63] P. Rogaway. Authenticated-encryption with associated-data. In V. Atluri, editor, *ACM CCS 02*, pages 98–107. ACM Press, Nov. 2002.
  - [64] P. Rogaway. The Evolution of Authenticated Encryption. <https://crypto.stanford.edu/RealWorldCrypto/slides/phil.pdf>, 2013.
  - [65] P. Rogaway and T. Shrimpton. A provable-security treatment of the key-wrap problem. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006.
  - [66] D. Rotaru, N. P. Smart, and M. Stam. Modes of operation suitable for computing on encrypted data. *Cryptology ePrint Archive*, Report 2017/496, 2017. <http://eprint.iacr.org/2017/496>.
  - [67] C.-P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, Aug. 1990.
  - [68] A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, Nov. 1979.
  - [69] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In K. Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 1–16. Springer, Heidelberg, May / June 1998.
  - [70] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, 2002.

## A CRYPTOGRAPHIC PRIMITIVES

### A.1 Authenticated Encryption

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

- Kgen( $1^\kappa$ )  $\rightarrow$   $sk$  : On input the security parameter, Kgen outputs a secret key  $sk$ .
- Encrypt( $sk, m$ )  $\rightarrow$   $c$  : On input the secret key  $sk$  and a message  $m$ , Encrypt outputs a ciphertext  $c$ .
- 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 Kgen( $1^\kappa$ ), and any message  $m$ , there exists a negligible function  $\text{negl}$  such that

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

where the probability is over the randomness of Encrypt.

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

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

where SymCPA is defined as follows:

1. *Initialize.* Run Kgen to get a key  $sk$ .
2. *Pre-challenge encryption queries.* On receiving (Encrypt,  $m$ ) from  $\mathcal{A}$ , return  $c \leftarrow \text{Encrypt}(sk, m)$ . This step can be repeated any number of times.
3. *Challenge.* When  $\mathcal{A}$  sends (Challenge,  $(m_0, m_1)$ ) such that  $|m_0| = |m_1|$ , choose a random bit  $b \leftarrow_{\$} \{0, 1\}$  and return  $c^* \leftarrow \text{Encrypt}(sk, m_b)$ .
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* [19, 55, 65]. A symmetric encryption scheme  $\Pi = (\text{Kgen}, \text{Encrypt}, \text{Decrypt})$  satisfies authenticity if for all PPT adversary  $\mathcal{A}$ , there exists a negligible function  $\text{negl}$  such that

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

where SymAUTH is defined as follows:

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

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

### A.2 Commitment

*Definition A.2.* A (non-interactive) commitment scheme  $\Sigma$  consists of two PPT algorithms (Setup<sub>com</sub>, Com) which satisfy hiding and binding properties:

- Setup<sub>com</sub>( $1^\kappa$ )  $\rightarrow$   $pp_{\text{com}}$  : It takes the security parameter as input, and outputs some public parameters.
- Com( $m, pp_{\text{com}}; r$ )  $=: \alpha$  : It takes a message  $m$ , public parameters  $pp_{\text{com}}$  and randomness  $r$  as inputs, and outputs a commitment  $\alpha$ .

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

$$|\Pr[\mathcal{A}(pp_{\text{com}}, \text{Com}(m_0, pp_{\text{com}}; r_0)) = 1] -$$

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

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

*Binding.* A commitment scheme  $\Sigma = (\text{Setup}_{\text{com}}, \text{Com})$  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  $pp_{\text{com}} \leftarrow \text{Setup}_{\text{com}}(1^\kappa)$ , then there exists a negligible function  $\text{negl}$  such that

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

where the probability is over the randomness of Setup<sub>com</sub> and the coin tosses of  $\mathcal{A}$ .

*Definition A.3 (Trapdoor (Non-interactive) Commitments.).* Let Com = (Setup<sub>com</sub>, Com) be a (non-interactive) commitment scheme.

A trapdoor commitment scheme has two more PPT algorithms  $\text{SimSetup}$  and  $\text{SimOpen}$ :

- $\text{SimSetup}(1^\kappa) \rightarrow (pp_{\text{com}}, \tau_{\text{com}})$ : It takes the security parameter as input, and outputs public parameters  $pp_{\text{com}}$  and a trapdoor  $\tau_{\text{com}}$ .
- $\text{SimOpen}(pp_{\text{com}}, \tau_{\text{com}}, m', (m, r)) =: r'$ : It takes parameters  $pp_{\text{com}}$ , trapdoor  $\tau_{\text{com}}$ , a message  $m'$  and a message-randomness pair  $(m, r)$  as inputs, and outputs a randomness  $r'$ .

For every  $(m, r)$  and  $m'$ , there exists a negligible function  $\text{negl}$  such that  $pp_{\text{com}} \approx_{\text{stat}} pp'_{\text{com}}$ , where  $pp_{\text{com}} \leftarrow \text{Setup}_{\text{com}}(1^\kappa)$  and  $(pp'_{\text{com}}, \tau_{\text{com}}) \leftarrow \text{SimSetup}(1^\kappa)$ ; and

$$\Pr [\text{Com}(m, pp'_{\text{com}}; r) = \text{Com}(m', pp'_{\text{com}}; r')] \geq 1 - \text{negl}(\kappa),$$

where  $r' := \text{SimOpen}(pp'_{\text{com}}, \tau_{\text{com}}, m', (m, r))$  and  $(pp'_{\text{com}}, \tau_{\text{com}}) \leftarrow \text{SimSetup}(1^\kappa)$ .

**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,  $\text{Setup}_{\text{com}}(1^\kappa)$  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  $\mathcal{H} : \{0, 1\}^* \rightarrow \mathbb{Z}_p$  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,  $\text{SimSetup}$  picks a random generator  $g$ , a random integer  $a$  in  $\mathbb{Z}_p^*$  and sets  $h$  to be  $g^a$ . Given  $(m, r)$ ,  $m'$  and  $a$ ,  $\text{SimOpen}$  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 [59]. We briefly describe here a non-interactive variant of the scheme for commitment to  $n$ -bit strings.  $\text{Setup}_{\text{com}}$  outputs a randomly chosen string  $\text{crs}$  of length  $4n$ . Let  $\text{pad} : \{0, 1\}^n \rightarrow \{0, 1\}^{4n}$  be the function that prepends  $3n$  zeroes to its argument,  $\mathcal{H} : \{0, 1\}^* \rightarrow \{0, 1\}^n$  be a collision-resistant hash function, and  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{4n}$  be a pseudo-random generator. Then, a commitment to a message  $m$  is given by  $G(r) + \text{crs} \cdot \text{pad}(\mathcal{H}(x))$  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  $s_1, \dots, s_n \in \mathbb{Z}_p$  such that the following two conditions hold for any set  $\{i_1, \dots, i_\ell\}$ :

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

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

Concretely, Shamir's secret sharing works as follows. Pick  $a_1, \dots, a_{t-1} \leftarrow_{\$} \mathbb{Z}_p$ . Let  $f(x)$  be the polynomial  $s + a_1 \cdot x + a_2 \cdot x^2 + \dots + a_{t-1} \cdot x^{t-1}$ . 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. [40].

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

- $\text{Prove}^{\mathcal{H}}(s, w)$  takes as input a statement  $s$  and a witness  $w$ , and outputs a proof  $\pi$  if  $(s, w) \in R$  and  $\perp$  otherwise.
- $\text{Verify}^{\mathcal{H}}(s, \pi)$  takes as input a statement  $s$  and a candidate proof  $\pi$ , and outputs a bit  $b \in \{0, 1\}$  denoting acceptance or rejection.

These two algorithms must satisfy the following properties:

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

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

- **Zero-knowledge:** There must exist a pair of PPT simulators  $(S_1, S_2)$  such that for all PPT adversary  $\mathcal{A}$ ,

$$\left| \Pr[\mathcal{A}^{\mathcal{H}, \text{Prove}^{\mathcal{H}}}(1^\kappa) = 1] - \Pr[\mathcal{A}^{S_1(\cdot), S'_2(\cdot, \cdot)}(1^\kappa) = 1] \right| \leq \text{negl}(\kappa)$$

for some negligible function  $\text{negl}$ , where

- $S_1$  simulates the random oracle  $\mathcal{H}$ ;
- $S'_2$  returns a simulated proof  $\pi \leftarrow S_2(s)$  on input  $(s, w)$  if  $(s, w) \in R$  and  $\perp$  otherwise;
- $S_1$  and  $S_2$  share states.
- **Argument of knowledge:** There must exist a PPT simulator  $S_1$  such that for all PPT adversary  $\mathcal{A}$ , there exists a PPT extractor  $\mathcal{E}^{\mathcal{A}}$  such that

$$\Pr [(s, w) \notin R \text{ and } \text{Verify}^{\mathcal{H}}(s, \pi) = 1 \mid$$

$$(s, \pi) \leftarrow \mathcal{A}^{S_1(\cdot)}(1^\kappa); w \leftarrow \mathcal{E}^{\mathcal{A}}(s, \pi, Q)] \leq \text{negl}(\kappa)$$

for some negligible function  $\text{negl}$ , where

- $S_1$  is like above;
- $Q$  is the list of (query, response) pairs obtained from  $S_1$ .

**Fiat-Shamir transform.** Let  $(\text{Prove}, \text{Verify})$  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  $(\text{Prove}^{\mathcal{H}}, \text{Verify}^{\mathcal{H}})$  derived from  $(\text{Prove}, \text{Verify})$  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. [40] for more details. (They actually show that these properties hold even when adversary can ask for proofs of false statements.)