Back to the future: simple threshold decryption secure against adaptive corruptions

Victor Shoup
Offchain Labs
victor@shoup.net

December 9, 2025

Abstract

We present a practical, non-interactive threshold decryption scheme. It can be proven CCA secure with respect to adaptive corruptions in the random oracle model under the decisional Diffie-Hellman assumption. Our scheme, called TDH2a, is a minor modification on the TDH2 scheme presented by Shoup and Gennaro at Eurocrypt 1998, which was proven secure against static corruptions under the same assumptions. The design and analysis of TDH2a are based on a straightforward extension of the simple information-theoretic argument underlying the security of the Cramer-Shoup encryption scheme presented at Crypto 1998. We also present another variant, TDH2abc, which offers a higher degree of backward compatibility with TDH2, allowing one to effectively "hot swap" TDH2abc in to replace TDH2 in a "live" system, without changing the public encryption key, so that any extant ciphertexts remain decryptable.

1 Introduction

TDH2 is a threshold decryption scheme introduced in [SG98]. It is a very simple and practical scheme. In particular, the decryption protocol is non-interactive, meaning that to decrypt a ciphertext c, it suffices to obtain decryption shares for c from sufficiently many servers and then combine these decryption shares to obtain the plaintext encrypted by c. It still remains one of the most practical, provably secure threshold decryption schemes available. For example, it is currently used in the Coinbase Open Source MPC Library (see https://github.com/coinbase/cb-mpc).

The paper [SG98] proves that TDH2 is **secure against chosen ciphertext attack (CCA secure)** under the **decisional Diffie-Hellman (DDH)** assumption. This proof models certain hash functions used in Fiat-Shamir-based proof systems as random oracles. It also assumes that the adversary corrupts decryption servers *statically*; that is, the adversary must choose which servers to corrupt at the very start of the attack game, before key generation.

It remains an open question if TDH2 is secure in an attack game where the adversary may corrupt servers *adaptively*, meaning that adversary may corrupt servers one by one throughout the attack game.

In this paper, we prove that a minor modification of TDH2, which we call TDH2a, is CCA secure in the adaptive corruption setting ("a" stands for "adaptive"). This proof works under the same assumptions as those used to prove TDH2: we model certain hash functions used in Fiat-Shamir-based proof systems as random oracles, and we assume the DDH.

The only nontrivial difference between TDH2 and TDH2a is that in TDH2, the secret key consists of a single scalar, while in TDH2a, it consists of 2 scalars; in addition, the proof system used by the decryption servers is slightly different.

We note that the format of the public key and encryption algorithm itself are identical in both schemes, which should simplify transitioning from TDH2 to TDH2a. However, one cannot transition without generating a new public encryption key, which is not ideal. Because of this, we also present a minor tweak of the TDH2a scheme, which we call TDH2abc — here, "abc" stands for "adaptive backward compatible". It is backward compatible with TDH2 in the sense that one can refresh the secret keys of an existing instance of TDH2 and start using TDH2abc *without changing the public encryption key.* Thus, one can effectively "hot swap" TDH2abc in to replace TDH2 in a "live" system, so that any extant ciphertexts remain decryptable.

Intuition behind TDH2a. We give here some intuition behind the design and analysis of TDH2a, and its relation to other schemes. First, consider TDH2. This can be seen as being derived from a non-threshold decryption scheme, with the reduction from static threshold security to non-threshold security being quite generic and simple. Let us call that latter non-threshold scheme DH2.

The scheme DH2 can be viewed as a variant of the Cramer-Shoup (CS) encryption scheme from [CS98, CS03]. The difference is that in DH2, a Fiat-Shamir-based proof system is used to prove ciphertext validity, while in CS, a hash-proof system is used (see [CS02]). Indeed, historically speaking, the CS encryption scheme is best seen as a refinement of DH2, where the Fiat-Shamir-based proof system is replaced by a hash-proof system. The advantage of hash-proof systems is that no random oracle is needed. The disadvantage is that the proofs are not publicly verifiable, making them hard to use in a threshold setting. While the proof technique used to analyze the CS encryption scheme in [CS98] is quite different from that used to analyze TDH2 in [SG98], one could, in fact, use the technique from [CS98] to analyze DH2, resulting in an alternative security proof for DH2 that is very different from the one in [SG98].

In this alternative security proof of DH2, just as in the proof of the CS scheme, one would split the secret key into two pieces at some point in the analysis of the scheme, even if these two pieces get combined in the final scheme. For example, in [CS98], the scheme is presented with a combined secret key, which is split in the analysis. However, in [CS03], a scheme CS1 is presented in which the secret key is split, and then a variation is presented in which the secret key is combined. The basic idea for the design and analysis of TDH2a is to simply leave the secret key split in the final scheme as well as the proof. This allows us to easily utilize the simple information-theoretic arguments developed in [CS98] to also deal with the potential leakage of information via adaptive corruptions. Leaving the secret key split in the final scheme seems necessary — in the non-threshold setting, we can switch between a split and combined secret key "behind the scenes", while in the adaptive threshold setting, we cannot. The design of TDH2a and its analysis actually become remarkably straightforward when one views the problem through this lens.

Related work. The literature on threshold decryption is vast. We refer the reader to [LY20, BBN+25] for overviews of the literature. We mention here the most relevant recent works.

In our work, we have given ourselves the luxury of working in the random oracle model. However, there has been a remarkable amount of progress on the problem of building a non-interactive threshold decryption scheme that can be proven CCA secure with respect to adaptive corruptions *without* relying on the random oracle model.

The most practical such schemes are based on CS encryption modified so as to make the hash-proof system used to prove ciphertext validity publicly verifiable, but without resorting to random oracles as was done originally in `DH2`. This line of work starts with [LY12] and continues in [LPJY14] (see especially Appendix I in the full version of [LPJY14]). Subsequent improvements to the underlying proof system appear in [ABP15, KW15]. Note that [ABP15] discusses applications to threshold decryption, but apparently only under static corruptions.[1] Also note that [KW15] does not discuss applications to threshold decryption at all. Nevertheless, it is fairly clear that these proof systems can be plugged into the construction in [LPJY14].

Unlike `TDH2a`, the above constructions all rely on elliptic curves that support pairings to obtain publicly verifiable hash proof systems. All of these schemes have constant sized public keys, secret keys, and ciphertexts. With the more efficient proof systems from [ABP15, KW15], the ciphertext size is a small constant factor larger than in `TDH2a` (due to the somewhat larger size of group element encodings for pairing-friendly curves). However, the computational cost of encryption will be a much more significant constant factor worse than in `TDH2a` (there are a few more operations, each of these are significantly more expensive in pairing-friendly curves).

The most practical such schemes that do not rely on pairing-friendly curves appear to be those in [DLN+21]. They present two schemes, one based on the *decision composite residuosity* assumption, and another based on the *learning with errors* assumption. Both of these schemes appear to be significantly less practical than the ones above that rely on pairings, as they utilize much more expensive correlation intractable hash functions. In particular, the secret decryption keys are not of constant size.

It may be fair to say that, by now, the very simple ideas underlying the design and analysis of `TDH2a` are essentially "folklore"; however, we flesh them out here more completely than was previously done in the literature. Essentially the same ideas were already employed in the constructions in [LY12, LPJY14], even if the presentation of these ideas could have been a bit more detailed and were somewhat obscured by the emphasis on techniques to design proof systems that avoid the random oracle model.[2] Indeed, it seems that research on adaptive corruptions has, more often than not, eschewed the random oracle model, and perhaps because of that, the fact that a simple variation on `TDH2` is already CCA secure under adaptive corruptions in the random oracle model seems to have mostly escaped the notice of the broader cryptographic research community. We would suggest that utilizing the random oracle model to get the most practical threshold schemes that are secure under adaptive corruptions is a pragmatic approach that is more likely to lead to the actual adoption of such schemes in practice.

We also note that the analysis of the variation `TDH2abc` additionally relies on a technique more recently introduced in [DR24]. We give a self-contained and (we believe) simpler exposition of that technique, which may also be of independent interest.

Comparison to [DRY25]. Prior to our work, [DRY25] also presented a practical, non-interactive threshold decryption scheme that can be proven CCA secure with respect to adaptive corruptions in the random oracle model under the DDH assumption. We stress that the design and analysis of

[1] Appendix D.1 of the full version of [ABP15] contains a high-level proof sketch for a non-threshold scheme that is essentially the same as `DH2`, also with a combined secret key, but with a different type of publicly verifiable proof system. This proof is essentially the "alternative security proof of `DH2`" mentioned above. It then just refers to [SG98] for details on thresholdizing their non-threshold scheme.

[2] See Appendix C.2 in the full version of [LY12], "universality" property, as well as the proof of Theorem 3 in the journal version of same [LY20]. Also see Game 9 in Appendix I.2 in the full version of [LPJY14].

their scheme are based on completely different principles than those used to develop TDH2a. Indeed, they utilize techniques recently introduced in [DR24]. In contrast, the design and analysis of TDH2a are based on nothing more than the simple information-theoretic argument underlying CS encryption. Despite this simplicity (or perhaps because of it), TDH2a is more efficient than the scheme in [DRY25]. In particular, in [DRY25], each server's decryption key consists of three scalars, instead of two, as in TDH2a, and their decryption and decryption-share combining algorithms are correspondingly less efficient. Moreover, their decryption and decryption-share combining algorithms need to hash ciphertexts to group elements, with these hash functions modeled as random oracles. In contrast, TDH2a, just like TDH2, does not need to hash to group elements. Indeed, the only use of random oracles in TDH2a, just like in TDH2, is in the Fiat-Shamir-based proof systems used to prove validity of ciphertexts and decryption shares.

The rest of the paper. In Section 2 we present and analyze TDH2a. In our analysis in Section 2.3, we assume that parties can erase ephemeral data. In Section 2.5, we discuss the security of TDH2a in the erasure-free setting. In Section 2.6, we discuss how to make TDH2a proactively secure as well. In Section 3, we present and analyze TDH2abc, and discuss how to transition from TDH2 to TDH2abc in a "live" system. Finally, in Section 4, we give an in-depth treatment of an alternative, simulation-based definition of CCA security for threshold decryption schemes, which also models erasure-free adaptive corruptions, and we show that TDH2a satisfies this definition. We believe this simulation-based definition is much more natural to work with in most applications of threshold decryption.

2 The TDH2a threshold decryption scheme

2.1 Building blocks

2.1.1 A cryptographic group

We assume a group $\mathbb{G}$ of prime order q generated by $\mathcal{G} \in \mathbb{G}$. We write $\mathbb{G}$ using additive notation, and $\mathcal{O} \in \mathbb{G}$ denotes the identity. Our scheme uses a second, random generator $\mathcal{G}' \in \mathbb{G}$, which may be viewed as a system parameter, implicitly input to all parties (along with a description of $\mathbb{G}$ and $\mathcal{G}$). The group element $\mathcal{G}'$ should be generated in such a way that an adversary has no information about the relationship between $\mathcal{G}$ and $\mathcal{G}'$.

The computational assumption we need is the **decisional Diffie-Hellman (DDH)** assumption. In our setting, this assumption says that the two distributions

$$\{(\beta\mathcal{G}, \beta\mathcal{G}') : \beta \xleftarrow{\$} \mathbb{Z}_q\}$$

and

$$\{(\beta\mathcal{G}, \beta'\mathcal{G}') : \beta \xleftarrow{\$} \mathbb{Z}_q, \beta' \xleftarrow{\$} \mathbb{Z}_q \setminus \{\beta\}\}$$

are computationally indistinguishable.

2.1.2 A proof system for equality of discrete logs

Let $\boldsymbol{AD}$ be some space of associated data. We need a non-interactive proof of knowledge for the relation

$$\{(\beta, (\mathcal{V}, \mathcal{V}', ad)) \in \mathbb{Z}_q \times (\mathbb{G} \times \mathbb{G} \times \boldsymbol{AD}) : \mathcal{V} = \beta\mathcal{G} \text{ and } \mathcal{V}' = \beta\mathcal{G}'\}.$$

The associated data ad in the relation is used for non-malleability purposes (see below).

We use the standard Fiat-Shamir transform (see Section 20.3.3 of [BS23]) applied to the standard Sigma protocol for this relation (i.e., the Chaum-Pedersen protocol, see Section 19.5.2 of [BS23]). The resulting proof generation and verification algorithms are invoked as

$$\mathit{GenPrf}(\beta, (\mathcal{V}, \mathcal{V}', ad)) \to \pi$$

and

$$\mathit{VrfyPrf}((\mathcal{V}, \mathcal{V}', ad), \pi) \to \texttt{accept/reject}.$$

The associated data ad is processed by including it in the hash that derives the challenge in the underlying Sigma protocol.

The properties we use are that, in the random oracle model, we have both *zero knowledge* (see Section 20.3.5 of [BS23]) and *(one-time) simulation soundness* (see Exercise 20.22 of [BS23]). One-time simulation soundness means that even if an adversary sees a simulated proof of a false statement, it cannot feasibly generate any other proof for that statement, or any valid proof for a different false statement. Note that statements in this setting are of the form $(\mathcal{V}, \mathcal{V}', ad)$, and so include the associated data ad. In particular, given a simulated proof of a false statement $(\mathcal{V}, \mathcal{V}', ad)$, the adversary cannot give a proof for any statement $(\mathcal{V}, \mathcal{V}', ad')$, where $ad' \neq ad$.

2.1.3 A proof system for equality of representations

We need a non-interactive proof of knowledge for the relation

$$\{((\alpha, \alpha'), (\mathcal{V}, \mathcal{V}', \mathcal{U}, \mathcal{W})) \in \mathbb{Z}_q^2 \times \mathbb{G}^4 : \mathcal{U} = \alpha\mathcal{G} + \alpha'\mathcal{G}' \text{ and } \mathcal{W} = \alpha\mathcal{V} + \alpha'\mathcal{V}'\}.$$

Again, we use the standard Fiat-Shamir transformation applied to the standard Sigma protocol for this relation (see Section 19.5.3 of [BS23]). The resulting proof generation and verification algorithms are invoked as

$$\mathit{GenPrf}'((\alpha, \alpha'), (\mathcal{V}, \mathcal{V}', \mathcal{U}, \mathcal{W})) \to \pi'$$

and

$$\mathit{VrfyPrf}'((\mathcal{V}, \mathcal{V}', \mathcal{U}, \mathcal{W}), \pi') \to \texttt{accept/reject}.$$

In the random oracle model, this proof system is zero knowledge. However, it turns out we do not even need this property. The underlying Sigma protocol is special honest verifier zero knowledge, and it follows that the resulting non-interactive protocol provides *witness independence* (see Section 19.8.1 of [BS23]), without using any properties of the underlying hash function. Witness independence means that the distribution of the proof depends only on the statement being proved, and not on the particular witness. In our application, whenever we need to generate a proof π' for a statement $(\mathcal{V}, \mathcal{V}', \mathcal{U}, \mathcal{W})$, it will be the case that $(\mathcal{V}, \mathcal{V}') = (\beta\mathcal{G}, \beta\mathcal{G}')$ for some (unknown) β, which means $\mathcal{W} = \beta\mathcal{U}$. This means that in an information-theoretic sense, the proof π' leaks no additional information about (α, α'), other than that implied by the relation $\mathcal{U} = \alpha\mathcal{G} + \alpha'\mathcal{G}'$. Indeed, from the underlying Sigma protocol, it is an easy exercise to devise a zero knowledge simulator that is perfect, straightline, but inefficient. We will refer to this as the **WI simulator** for the proof system. See Appendix A for details.

The fact that we need only witness independence rather than zero knowledge for this proof system is not essential to the security analysis of our threshold decryption scheme. Rather, it just lets us get by with one less random oracle in that analysis.

We will also use soundness properties of this protocol in the random oracle model, but in the following way. In our application, whenever we verify a proof π' for a statement $(\mathcal{V}, \mathcal{V}', \mathcal{U}, \mathcal{W})$, we may assume that $(\mathcal{V}, \mathcal{V}') = (\beta\mathcal{G}, \beta\mathcal{G}')$ for some (unknown) β. This assumption will be justified by the (ordinary) soundness of the proof system in Section 2.1.2. In this restricted setting, the proof π' is really just a proof that $\mathcal{W} = \beta\mathcal{U}$. One can check that the special soundness property of the underlying Sigma protocol (see Section 19.4.1 of [BS23]) guarantees that if $\mathcal{W} \neq \beta\mathcal{U}$, after the prover generates its commitment in the Sigma protocol, there is only one challenge to which it can correctly respond.

It follows that in using this proof system, as well as the one in Section 2.1.2, we will never need to employ any kind of "rewinding" argument in the security analysis.

We note that for the proof system in Section 2.1.2, we use the soundness property both for the CCA security analysis and robustness analysis of our threshold decryption scheme. However, for the proof system in this section, we only need to use soundness to prove robustness of our scheme.

2.1.4 A symmetric encryption scheme

We need a symmetric encryption scheme, comprised of a deterministic encryption algorithm E_{s} and a deterministic decryption algorithm D_{s}. Keys are drawn from a finite key space $\boldsymbol{K}$. We write $e \leftarrow E_{\mathrm{s}}(k, m)$ to encrypt a message m under key k. We write $m \leftarrow D_{\mathrm{s}}(k, e)$ to decrypt a ciphertext c under key k.

2.1.5 A key derivation function

We need a key derivation function $H : \mathbb{G} \to \boldsymbol{K}$. The security property for such a function is that the two distributions

$$\{H(\mathcal{W}) : \mathcal{W} \xleftarrow{\$} \mathbb{G}\}$$

and

$$\{k : k \xleftarrow{\$} \boldsymbol{K}\}$$

are computationally indistinguishable.

2.1.6 Lagrange interpolation

We use the following basic fact about Lagrange interpolation. For any subset $\boldsymbol{J} \subseteq \{1, \ldots, N\}$ of size t, one can efficiently compute scalars $\lambda_j \in \mathbb{Z}_q$ for $j \in \boldsymbol{J}$ such that for any polynomial $f \in \mathbb{Z}_q[X]_{<t}$, we have $f(0) = \sum_{j \in \boldsymbol{J}} \lambda_j f(j)$.

2.2 Definition of the TDH2a threshold decryption scheme

We now define the scheme TDH2a. We follow the notation and terminology from Section 22.3 of [BS23].

- **Key generation**

  $G(N, t) \to (pk,\ pkc,\ sk_1, \ldots, sk_N)$:

  [Note: N = # parties, t = reconstruction threshold, pk is the public encryption key, pkc is the public combiner key, and sk_i is the secret key for server P_i for $i = 1, \ldots, N$]

- generate random polynomials $f, f' \in \mathbb{Z}_q[X]_{<t}$
- for $i = 1, \ldots, N$, compute $sk_i \coloneqq (\alpha_i, \alpha'_i) \coloneqq (f(i), f'(i))$
- for $i = 0, \ldots, N$, compute $\mathcal{U}_i \coloneqq \alpha_i \mathcal{G} + \alpha'_i \mathcal{G}'$
- set $pk \coloneqq \mathcal{U} \coloneqq \mathcal{U}_0$, $pkc = (\mathcal{U}_1, \ldots, \mathcal{U}_N)$
- output $(pk,\ pkc,\ sk_1, \ldots, sk_N)$

- **Encryption**

  $E(pk, m, d) \to c$, where $pk = \mathcal{U}$:

  [Note: m is the message, d is associated data, c is the ciphertext]

  - set $c = (\mathcal{V}, \mathcal{V}', e, \pi)$, where

    $\beta \xleftarrow{\$} \mathbb{Z}_q$, $\mathcal{V} \leftarrow \beta\mathcal{G}$, $\mathcal{V}' \leftarrow \beta\mathcal{G}'$, $\mathcal{W} \leftarrow \beta\mathcal{U}$
    $k \leftarrow H(\mathcal{W})$, $e \leftarrow E_{\mathrm{s}}(k, m)$
    $\pi \xleftarrow{\$} \mathit{GenPrf}(\beta, (\mathcal{V}, \mathcal{V}', (e, d)))$

- **Decryption**

  $D(sk_i, c, d) \to c'_i$, where $sk_i = (\alpha_i, \alpha'_i)$ and $c = (\mathcal{V}, \mathcal{V}', e, \pi)$:

  [Note: c'_i is a decryption share]

  - check that

    $$\mathit{VrfyPrf}((\mathcal{V}, \mathcal{V}', (e, d)), \pi) = \texttt{accept};$$

    if not, output `reject`; otherwise, output $c'_i = (\mathcal{W}_i, \pi'_i)$, where

    $$\mathcal{W}_i = \alpha_i \mathcal{V} + \alpha'_i \mathcal{V}'$$

    and

    $$\pi'_i \xleftarrow{\$} \mathit{GenPrf}'((\alpha_i, \alpha'_i), (\mathcal{V}, \mathcal{V}', \mathcal{U}_i, \mathcal{W}_i));$$

    here, $\mathcal{U}_i = \alpha_i \mathcal{G} + \alpha'_i \mathcal{G}'$, which is technically not a part of the secret key, but it can be stored along with the secret key;

- **Combination of decryption shares**

  $C(pkc, c, d, \boldsymbol{J}, \{c'_j\}_{j \in \boldsymbol{J}}) \to m$, where $pkc = (\mathcal{U}_1, \ldots, \mathcal{U}_N)$, $c = (\mathcal{V}, \mathcal{V}', e, \pi)$, $\boldsymbol{J}$ is a subset of $\{1, \ldots, N\}$ of size t, and where $c'_j = \texttt{reject}$ or $c'_j = (\mathcal{W}_j, \pi'_j)$ for $j \in \boldsymbol{J}$:

  - check that

    $$\mathit{VrfyPrf}((\mathcal{V}, \mathcal{V}', (e, d)), \pi) = \texttt{accept};$$

    if not, output `reject` and halt;
  - check if for some $j \in \boldsymbol{J}$, we have either
    - $c'_j = \texttt{reject}$, or
    - $c'_j = (\mathcal{W}_j, \pi'_j)$ and $\mathit{VrfyPrf}'((\mathcal{V}, \mathcal{V}', \mathcal{U}_j, \mathcal{W}_j), \pi'_j) = \texttt{reject}$;

    if so, output $\texttt{blame}(\boldsymbol{J}^*)$, where $\boldsymbol{J}^*$ is the set of all such $j \in \boldsymbol{J}$;
  - compute $\mathcal{W} \leftarrow \sum_{j \in \boldsymbol{J}} \lambda_j \mathcal{W}_j$, where $\{\lambda_j\}_{j \in \boldsymbol{J}}$ are the Lagrange interpolation coefficients that move from the set of evaluation points $\boldsymbol{J}$ to the evaluation point 0;
  - compute $k \leftarrow H(\mathcal{W})$, $m \leftarrow D_{\mathrm{s}}(k, e)$ and output m.

2.2.1 Discussion

The scheme presented here is very similar to the scheme `TDH2` from [SG98]. The only non-cosmetic difference is that in `TDH2`, we effectively set the polynomial f' in the key generation algorithm of `TDH2a` to zero. This means that $\alpha'_i = 0$ for $i = 0, \ldots, N$ in `TDH2`. Besides simplifying the computation of $\mathcal{W}_i$ in the decryption algorithm, it also simplifies the proof system used to generate and verify the proof π'_i. Indeed, in `TDH2`, the proof of correct decryption is essentially Chaum-Pedersen, just like in the proof of valid encryption.

Throughout this paper, we treat key generation as a trusted, centralized process. In a real system, key generation will typically be implemented by a distributed key generation protocol that emulates this process.

2.3 Proof of CCA security

Our definition of CCA security closely follows that in Section 22.3.4 of [BS23], modified to model adaptive corruptions. Security is defined in terms of an attack game, played between a challenger and adversary, which runs as follows:

> The challenger first runs the key generation algorithm
>
> $$(pk,\ pkc,\ sk_1, \ldots, sk_N) \xleftarrow{\$} G(N, t)$$
>
> and gives the public information (pk, pkc) to the adversary. The challenger also chooses $b \in \{0, 1\}$ at random.
>
> Next, the adversary makes a sequence of encryption, decryption, and corruption queries.
>
> - The adversary may make just one encryption query, which is of the form (m_0, m_1, d), where m_0 and m_1 are messages of the same length and d is associated data. The challenger computes the ciphertext $c \xleftarrow{\$} E(pk, m_b, d)$ and gives c to the adversary.
> - Each decryption query is of the form $(\hat{c}, \hat{d}, i)$, where $\hat{c}$ is a ciphertext, $\hat{d}$ is associated data, and $i \in \{1, \ldots, N\}$ is the index of an uncorrupted server P_i. A restriction is that if the adversary has already made its encryption query, then $(\hat{c}, \hat{d}) \neq (c, d)$. The challenger computes the decryption share $c'_i \xleftarrow{\$} D(sk_i, \hat{c}, \hat{d})$ and sends this to the adversary.
> - The adversary may make at most $t - 1$ corruption queries. Each corruption query is of the form $\ell \in \{1, \ldots, N\}$, where ℓ is the index of an uncorrupted server P_ℓ. In response, the challenger sends the secret key sk_ℓ of P_ℓ to the adversary.
>
> In addition, if we wish to model any hash functions as random oracles, then the adversary may also make queries to obtain the value of these random oracles at points of its choosing. The challenger will also internally access these random oracles when processing encryption and decryption queries.
>
> At the end of the game, the adversary outputs $\hat{b} \in \{0, 1\}$.

The adversary's **advantage** is defined to be $|\Pr[\hat{b} = b] - 1/2|$, and the scheme is said to be **CCA secure under adaptive corruptions** if this advantage is negligible for all efficient adversaries.

Remarks:

1. Our definition of CCA security for a threshold decryption scheme is the same as presented in Section 22.3.4 of [BS23], except as follows:

   - First, we allow for adaptive corruptions. This means that in Attack Game 22.5, instead of having the adversary specify a set of up to $t-1$ corrupt servers in the setup phase of the game, we allow the adversary to corrupt up to $t-1$ servers adaptively. More precisely, in addition to *encryption* and *decryption* queries, we also allow for up to $t-1$ *corruption* queries. Each such query specifies the index $\ell \in \{1, \ldots, N\}$ of a previously uncorrupted decryption server P_ℓ. In response to such a query, the challenger gives the secret key sk_ℓ to the adversary.
   - Second, we enhance the attack game to achieve full CCA security, as discussed in Remark 22.10 of [BS23].
   - Third, we assume only one encryption query. By a standard "hybrid argument", this is sufficient to prove security in the setting where we allow multiple encryption queries.
   - Fourth, we work with the "bit guessing" version of the attack game (see Section 2.2.5 of [BS23]), where the challenger chooses the index $b \in \{0, 1\}$ to choose which of two messages to encrypt, and the adversary's goal is essentially to make a guess $\hat{b} \in \{0, 1\}$ at the challenger's hidden bit b.
   - Fifth, each decryption query specifies an individual server.

2. As we only leak the secret key sk_ℓ of a corrupted decryption server to the adversary, this definition of adaptive corruption models a setting where decryption servers, as well as encryptors, are able to erase ephemeral data. See Section 2.5 for a discussion of the security of `TDH2a` in the erasure-free setting.

3. This attack game effectively models trusted key generation. In an actual deployment, one would want to use a distributed key generation protocol that emulates a corresponding ideal functionality with respect to adaptive corruptions. There are certainly many such protocols to choose from [BK25].

4. Our definition of adaptive CCA security is the same as the notion of "context-free low-threshold adaptive CCA security" defined in Section 8.1 of [BBN+25]. In this notion of security, the adversary is not allowed to decrypt a ciphertext/associated-data pair corresponding to an encryption query.

   Theorem 4 of [BBN+25] also shows that this notion is equivalent to a stronger notion of security, "context-free *high*-threshold adaptive CCA security", which allows such ciphertext/associated-data pairs to be decrypted, so long as the number of parties who are corrupt or decrypt that pair stays below the reconstruction threshold t.

   Section 8.2 of [BBN+25] also defines a simulation-based definition of adaptive CCA security, and observes that any scheme that satisfies the game-based definition can be easily converted to one that satisfies the simulation-based definition. Moreover, analogous to the observation made there, our scheme `TDH2a` satisfies this definition, even in the erasure-free setting, if we implement symmetric encryption as $e = F(k) \oplus m$, where F is modeled as a random oracle. This is explored in detail in Section 4, where we give a self-contained treatment of

this, including full proofs (which do not appear in [BBN+25]). This simulation-based security notion better models a system where all ciphertexts may eventually be decrypted, but we want to ensure that information about their contents leaks only when ciphertexts are legitimately compromised. We believe that this simulation-based definition is in fact the strongest and most natural definition to work with in analyzing the security of larger systems that use a threshold decryption scheme. Moreover, `TDH2a` satisfies this definition.

We now prove that `TDH2a` is CCA secure under adaptive corruptions via a sequence of games.

Game 0. This is the actual attack game, as described above. Note also that we are modeling the hash function used to implement the proof system described in Section 2.1.2 as a random oracle. This is the proof system used to generate the proof π in the encryption algorithm and to verify such proofs in the decryption algorithm. Recall that at the end of the game, the adversary outputs $\hat{b} \in \{0,1\}$. We let E_0 be the event that $\hat{b} = b$. Our goal is to show that $|\Pr[E_0] - 1/2|$ is negligible. In the following, we define Games 1, 2, ..., in each Game i, we define E_i to be the same event in that game.

Game 1. We make a modification to the encryption query logic. In Game 0, the encryption query (m_0, m_1, d) is processed by the challenger as follows:

$\beta \xleftarrow{\$} \mathbb{Z}_q$, $\mathcal{V} \leftarrow \beta\mathcal{G}$, $\mathcal{V}' \leftarrow \beta\mathcal{G}'$, $\mathcal{W} \leftarrow \beta\mathcal{U}$
$k \leftarrow H(\mathcal{W})$, $e \leftarrow E_s(k, m)$
$\pi \xleftarrow{\$} \mathit{GenPrf}(\beta, (\mathcal{V}, \mathcal{V}', (e, d)))$
return $c = (\mathcal{V}, \mathcal{V}', e, \pi)$

In Game 1, we replace the computation of

$$\pi \xleftarrow{\$} \mathit{GenPrf}(\beta, (\mathcal{V}, \mathcal{V}', (e, d)))$$

by

$$\pi \xleftarrow{\$} \mathit{Sim}(\mathcal{V}, \mathcal{V}', (e, d)),$$

where *Sim* is the simulator guaranteed by the zero-knowledge property of the proof system from Section 2.1.2. Note that *Sim* in is also in charge of simulating the random oracle queries (as discussed in Section 20.3.5 of [BS23]).

By the zero-knowledge property, $|\Pr[E_1] - \Pr[E_0]|$ is negligible.

Game 2. We make another change to the encryption query logic. We change the way $\mathcal{W}$ is computed. Instead of computing it as $\mathcal{W} \leftarrow \beta\mathcal{U}$, we instead compute it as $\mathcal{W} \leftarrow \alpha_0\mathcal{V} + \alpha_0'\mathcal{V}'$. Recall that $\mathcal{U} = \alpha_0\mathcal{G} + \alpha_0'\mathcal{G}'$, and since $(\mathcal{V}, \mathcal{V}') = (\beta\mathcal{G}, \beta\mathcal{G}')$, this is completely equivalent.

Thus, $\Pr[E_2] = \Pr[E_1]$.

Game 3. The encryption query logic in Game 2 runs as follows:

$\beta \xleftarrow{\$} \mathbb{Z}_q$, $\mathcal{V} \leftarrow \beta\mathcal{G}$, $\mathcal{V}' \leftarrow \beta\mathcal{G}'$, $\mathcal{W} \leftarrow \alpha_0\mathcal{V} + \alpha_0'\mathcal{V}'$
$k \leftarrow H(\mathcal{W})$, $e \leftarrow E_s(k, m)$
$\pi \xleftarrow{\$} \mathit{Sim}(\mathcal{V}, \mathcal{V}', (e, d))$
return $c = (\mathcal{V}, \mathcal{V}', e, \pi)$

Notice that β is only used in the computation of $\mathcal{V}$ and $\mathcal{V}'$, and nowhere else. So we are now free to "play our DDH card", replacing the computation of $\mathcal{V}$ and $\mathcal{V}'$ by

$$\beta \xleftarrow{\$} \mathbb{Z}_q,\ \beta' \xleftarrow{\$} \mathbb{Z}_q \setminus \{\beta\},\ \mathcal{V} \leftarrow \beta\mathcal{G},\ \mathcal{V}' \leftarrow \beta'\mathcal{G}'.$$

Under the DDH assumption, $|\Pr[E_3] - \Pr[E_2]|$ is negligible.

Note that in this game, the simulator *Sim* is actually proving a false statement.

Game 4. We now modify the logic of decryption queries. In any such query, the challenger is given a ciphertext $\hat{c} = (\hat{\mathcal{V}}, \hat{\mathcal{V}}', \hat{e}, \hat{\pi})$ and associated data $\hat{d}$. If this decryption query follows the encryption query, then $(\hat{c}, \hat{d}) \neq (c, d)$.

The modification we make is as follows. First, we generate the system parameter $\mathcal{G}'$ as

$$\zeta \xleftarrow{\$} \mathbb{Z}_q^*,\ \mathcal{G}' \leftarrow \zeta\mathcal{G},$$

so that now the challenger knows the discrete log of $\mathcal{G}'$ with respect to $\mathcal{G}$. This is not a problem, as we have already played our DDH card. Second, in processing a decryption query as above, we introduce a *special rejection rule*. Namely, we first check if $\hat{\mathcal{V}}' = \zeta\hat{\mathcal{V}}$. If not, we set each decryption share c_i' to `reject`, and return this to the adversary without further processing.

By the simulation soundness property for the proof system from Section 2.1.2, $|\Pr[E_4] - \Pr[E_3]|$ is negligible.

With this modification, we now have the following property. Any decryption query $((\hat{\mathcal{V}}, \hat{\mathcal{V}}', \hat{e}, \hat{\pi}), \hat{d}, i)$ that is not rejected out of hand by the special rejection rule must satisfy $(\hat{\mathcal{V}}, \hat{\mathcal{V}}') = (\hat{\beta}\mathcal{G}, \hat{\beta}\mathcal{G}')$ for some (unknown) $\hat{\beta} \in \mathbb{Z}_q$.

Game 5. We now are at the heart of the proof. In the previous game, the encryption query logic runs as follows:

$\beta \xleftarrow{\$} \mathbb{Z}_q$, $\beta' \xleftarrow{\$} \mathbb{Z}_q \setminus \{\beta\}$, $\mathcal{V} \leftarrow \beta\mathcal{G}$, $\mathcal{V}' \leftarrow \beta'\mathcal{G}'$, $\mathcal{W} \leftarrow \alpha_0\mathcal{V} + \alpha_0'\mathcal{V}'$
$k \leftarrow H(\mathcal{W})$, $e \leftarrow E_{\mathrm{s}}(k, m)$
$\pi \xleftarrow{\$} Sim(\mathcal{V}, \mathcal{V}', (e, d))$
return $c = (\mathcal{V}, \mathcal{V}', e, \pi)$

We make one simple change to this logic, replacing the computation of $\mathcal{W}$ as

$$\mathcal{W} \leftarrow \alpha_0\mathcal{V} + \alpha_0'\mathcal{V}'$$

by

$$\mathcal{W} \xleftarrow{\$} \mathbb{G}.$$

We claim that $\Pr[E_5] = \Pr[E_4]$. We give the proof of this claim below in Lemma 1.

Game 6. In the encryption query logic, we replace the computation of

$$k \leftarrow H(\mathcal{W})$$

by

$$k \xleftarrow{\$} \boldsymbol{K}.$$

By the security of H as a key derivation function, as in Section 2.1.5, $|\Pr[E_6] - \Pr[E_5]|$ is negligible. Moreover, by the semantic security of the symmetric encryption scheme from Section 2.1.4, $|\Pr[E_6] - 1/2|$ is negligible. It follows that $|\Pr[E_0] - 1/2|$ is negligible.

Putting everything together, we have:

Theorem 1. *Modeling the hash function used in the proof system from Section 2.1.2 as a random oracle, and assuming the DDH as in Section 2.1.1, the semantic security of the symmetric encryption scheme from Section 2.1.4, and the security of H as a key derivation function as in Section 2.1.5, the scheme* `TDH2a` *is CCA secure under adaptive corruptions.*

To finish the proof, we need to prove the following lemma:

Lemma 1. *We have* $\Pr[E_5] = \Pr[E_4]$.

Proof. The proof is a purely information-theoretic argument that is an easy generalization of the technique in [CS98].

We want to show that the outcomes in Games 4 and 5 are determined by random variables that have identical distributions. Since this is a purely information-theoretic argument, we can work exclusively with discrete logarithms to the base $\mathcal{G}$, rather than with group elements.

In the key generation algorithm, we generate polynomials

$$f = \rho_0 + \rho_1 X + \cdots + \rho_{t-1} X^{t-1} \quad \text{and} \quad f' = \rho'_0 + \rho'_1 X + \cdots + \rho'_{t-1} X^{t-1}$$

where the values ρ_j and ρ'_j for $j = 0, \ldots, t-1$ are generated at random. Let us characterize the information that is revealed about these values during the course of the attack game.

From the key generation algorithm, the values

$$\epsilon_j \coloneqq \rho_j + \rho'_j \zeta \quad (j = 0, \ldots, t-1),$$

are revealed, where $\mathcal{G}' = \zeta \mathcal{G}$ and $\zeta \neq 0$. The actual values given to the adversary by key generation can be derived from this information. Indeed, for $i = 0, \ldots, N$, we have

$$\sum_j \epsilon_j i^j = \alpha_i + \alpha'_i \zeta$$

and so

$$\mathcal{U}_i = \alpha_i \mathcal{G} + \alpha'_i \mathcal{G}' = \sum_j \epsilon_j i^j \mathcal{G}.$$

The decryption queries reveal no further information. Indeed, as we have already argued, any decryption query $((\hat{\mathcal{V}}, \hat{\mathcal{V}}', \hat{e}, \hat{\pi}), \hat{d}, i)$ that is not rejected out of hand by the special rejection rule introduced in that game must satisfy $(\hat{\mathcal{V}}, \hat{\mathcal{V}}') = (\hat{\beta}\mathcal{G}, \hat{\beta}\mathcal{G}')$ for some $\hat{\beta} \in \mathbb{Z}_q$. Therefore, the value $\hat{W}_i$ computed by the decryption algorithm is $\hat{W}_i = \hat{\beta}\mathcal{U}_i = \hat{\beta} \sum_j \epsilon_j i^j \mathcal{G}$. Moreover, as discussed in Section 2.1.3, the proof system used to generate the proof π'_i provides witness independence, which means that nothing more is revealed about the values ρ_j and ρ'_j for $j = 0, \ldots, t-1$ from these proofs. More concretely, we assume that in both games we use the WI simulator to generate these proofs.

For a given corruption query, which specifies an index $\ell \in \{1, \ldots, N\}$, the value $\alpha_\ell = \sum_j \rho_j \ell^j$ is revealed. In the attack game, the value $sk_\ell = (\alpha_\ell, \alpha'_\ell)$ is given to the adversary. However, the value α'_ℓ can be derived as

$$\alpha'_\ell = \Big(\sum_j \epsilon_j \ell^j - \alpha_\ell \Big) / \zeta.$$

Indeed, the right-hand side is equal to

$$\Big(\sum_j \epsilon_j \ell^j - \alpha_\ell \Big) / \zeta = \Big((\alpha_\ell + \alpha'_\ell \zeta) - \alpha_\ell \Big) / \zeta = \alpha'_\ell.$$

Finally, in Game 4, the discrete logarithm ω of the group element $\mathcal{W}$ computed in processing the encryption query can be written as

$$\omega = \beta \alpha_0 + \beta' \zeta \alpha'_0,$$

where $\beta \neq \beta'$, while in Game 5, we set $\omega := \omega^*$ where ω^* is a new, independent random variable (uniformly distributed over $\mathbb{Z}_q$). This is the only difference between the two games.

For Game 4, we can write all of this as a system of linear equations as follows:

$$\overbrace{\begin{pmatrix} \omega \\ \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_{t-1} \\ \alpha_{\ell_1} \\ \vdots \\ \alpha_{\ell_s} \end{pmatrix}}^{V :=} = \overbrace{\begin{pmatrix} \beta & & & & \beta'\zeta & & & 0 \\ 1 & & & & \zeta & & & \\ & 1 & & & & \zeta & & \\ & & \ddots & & & & \ddots & \\ & & & 1 & & & & \zeta \\ 1 & \ell_1 & \cdots & \ell_1^{t-1} & & & & \\ & & \vdots & & & & & \\ 1 & \ell_s & \cdots & \ell_s^{t-1} & & & & \end{pmatrix}}^{M :=} \cdot \begin{pmatrix} \rho_0 \\ \rho_1 \\ \vdots \\ \rho_{t-1} \\ \rho'_0 \\ \rho'_1 \\ \vdots \\ \rho'_{t-1} \\ \omega^* \end{pmatrix}. \tag{1}$$

Here, $\ell_1, \ldots, \ell_s$ are distinct, nonzero elements of $\mathbb{Z}_q$, where $s \leq t-1$. So the matrix has $2t+1$ columns and $1 + t + s \leq 2t$ rows. Note that the right-most column of M is zero, as ω^* is not used in this game.

We claim that the rows of M are linearly independent. To see this, let us number the rows $0, 1, \ldots, t+s$, where the top row in the matrix is row 0. Let us also number the columns $1, \ldots, 2t+1$. Consider any linear combination of the rows with scalars $c_0, \ldots, c_{t+s} \in \mathbb{Z}_q$ that produces the zero vector. We want to show that $c_0 = \cdots = c_{t+s} = 0$. Looking at columns $t+2, \ldots, 2t$ of M, it is clear that we must have $c_2 = \cdots = c_t = 0$. So ignoring rows $2, \ldots, t$, and looking at columns $2, \ldots, t$, it is clear that we must have $c_{t+1} = \cdots = c_{t+s} = 0$ — this is because by properties of Vandermonde matrices, the vectors

$$(\ell_1, \ldots, \ell_1^{t-1}), \ldots, (\ell_s, \ldots, \ell_s^{t-1})$$

are linearly independent. So we can ignore all but rows 0 and 1. But these two rows are clearly linearly independent. To see this, observe that the matrix

$$\begin{pmatrix} \beta & \beta'\zeta \\ 1 & \zeta \end{pmatrix}$$

has determinant $\zeta \cdot (\beta - \beta') \neq 0$.

Since M has full rank, and the entries in the column vector on the right-hand side of (1) are uniformly and independently distributed over $\mathbb{Z}_q$, it follows that the entries in the vector V on the left-hand side of (1) are also uniformly and independently distributed over $\mathbb{Z}_q$.

One technical nuance we have glossed over is that the entries in the matrix M are not really fixed. While first $t + 1$ rows of M are fixed, the remaining rows are chosen adaptively by the adversary, and each such row may depend on the previous rows of V. However, this does not change the result: each new row in V is still uniformly distributed and independent of the previous rows (see, for example, Lemma 6.7 in [CS03], which is also Lemma 9 in [CS01]).

Observe that the outcome of the attack is completely determined by:

(i) the random tape of the adversary,

(ii) the random values β, β', ζ,

(iii) the random values used by *Sim* in the encryption query,

(iv) the random values used by the WI simulator in the decryption queries, and

(v) the vector V.

The above argument shows that the values in (v) are independently and uniformly distributed, and independent of (i), (ii), (iii), and (iv).

In going from Game 4 to Game 5, we just replace the top row of M by the vector $(0, \ldots, 0, 1)$. The resulting matrix M' is also seen to have full rank. From this, it is clear that the distributions of the random variables that determine the outcome of each game are identical. □

2.4 Robustness

We define robustness exactly as in Section 22.3.4.1 of [BS23]. The proof of this property for `TDH2a` follows from the soundness property discussed in Sections 2.1.2 and 2.1.3. In the random oracle model, this soundness property holds unconditionally, and proof of robustness does not require any kind of "rewinding" argument.

2.5 To erase or not to erase

As we already mentioned in Section 2.3, our attack game defining CCA security only leaks the secret key sk_ℓ of a corrupted decryption server to the adversary. This definition of security models a setting where decryption servers, as well as encryptors, are able to erase ephemeral data.

2.5.1 Corrupting decryption servers

Consider the adaptive corruption of decryption servers. In our protocol, if decryption servers cannot erase, in addition to leaking secret keys, we must leak the random values that were generated in constructing the proofs of correct decryption.

However, it is easy to see that for this proof system, from a proof and a witness, it is easy to derive the corresponding random values. As such, the proof of Lemma 1 can be trivially adapted to model erasure-free corruptions of the decryption servers. See Appendix A for details. A similar

observation was also made in [DR24]; note also that while [DRY25] does not explicitly discuss erasures, the same observation applies.

One caveat to the above argument is that, in practice, these random values may be the output of some implementation-dependent pseudorandom generation process, and so it is not enough to leak the random values, but rather, one must leak whatever information is input to this pseudorandom generation process. This can be made to work if these random values are generated as the output of a cryptographic hash function that may be modeled as a random oracle, which can then be programmed to hit a random value derived from a proof and witness.

2.5.2 Corrupting encryptors

Consider the adaptive corruption of encryptors. In protocols where decryption servers also play the role of encryptors, one must consider this situation.

If encryptors can erase, then our protocol certainly remains secure, as there is no information leaked upon corrupting an encryptor.

If encryptors cannot erase, then we need to work with a simulation-based notion of security to model this setting. We give a full treatment of this in Section 4.

2.6 Proactive security

To achieve proactive security [OY91], one can employ the standard technique of starting a new epoch periodically by having the parties refresh their secret keys, for example, by generating two random sharings of zero and then locally adding these shares to their existing keys. The combiner public keys are also adjusted accordingly. For this to work, we also need to assume parties can erase their secret keys from previous epochs. One can show that `TDH2a` achieves proactive adaptive CCA security. The proof of this is not modular, that is, it does not follow from the fact that `TDH2a` is adaptive CCA secure. One must carry out a monolithic proof over all epochs simultaneously. The information-theoretic argument in Lemma 1 based on linear algebra carries over in a straightforward way, as in each new epoch, we introduce $2(t-1)$ new variables and at most $2(t-1)$ equations, and we get one big system of linear equations.

3 The `TDH2abc` threshold decryption scheme

We now present a minor tweak of the `TDH2a` scheme, which we call `TDH2abc` — here, "abc" stands for "adaptive backward compatible". It is backward compatible with `TDH2` in the sense that one can refresh the secret keys of an existing instance of `TDH2` and start using `TDH2abc` *without changing the public encryption key.* Thus, one can effectively "hot swap" `TDH2abc` in to replace `TDH2` in a "live" system, so that any extant ciphertexts remain decryptable.

There is only one difference between `TDH2a` and `TDH2abc`:

> in `TDH2a` the polynomial f' chosen in key generation was a random polynomial of degree less than t, while in `TDH2abc`, it is a random polynomial of degree less than t with $f'(0) = 0$.

Everything else is the same. In particular, the public key is computed as

$$\mathcal{U} = \mathcal{U}_0 = \alpha_0 \mathcal{G} + \alpha'_0 \mathcal{G}' = \alpha_0 \mathcal{G}.$$

So to transition from TDH2 to TDH2abc, the parties refresh their secret keys, generating two random sharings of zero and then locally adding these shares to their original keys (with the second original key defined to be zero). The combiner public keys are also adjusted accordingly.

We shall give a proof that TDH2abc is just as secure as TDH2a. In a more complete analysis, one would have to carry out an analysis of a more complete system, consisting of one epoch running TDH2 in which the adversary corrupts parties statically, followed by a key refresh and a second epoch in which all parties start out as honest but may then be corrupted adaptively. Such an analysis can easily be done using the proof techniques presented here, but we leave the details to the reader. This analysis can also be extended to a more general proactive system, consisting of several epochs of TDH2 with static corruptions in each epoch, followed by several epochs of TDH2abc with adaptive corruptions in each epoch.

Theorem 2. *Modeling the hash function used in the proof system from Section 2.1.2 as a random oracle, if TDH2a is CCA secure under adaptive corruptions, then so is TDH2abc.*

The proof is also by a sequence of games. Game 0 is the same as Game 0 in the proof of Theorem 1, except that key generation is now as in TDH2abc, where the constant term of the polynomial f' is zero. We define events $E_0, E_1, \ldots$ just as in the proof of Theorem 1.

In Game 1, we introduce the same special rejection rule that we introduced in Game 4 of the proof of Theorem 1. By the (ordinary) soundness property for the proof system from Section 2.1.2, $|\Pr[E_1] - \Pr[E_0]|$ is negligible.

With this modification, we now have the following property. Any decryption query $((\hat{\mathcal{V}}, \hat{\mathcal{V}}', \hat{e}, \hat{\pi}), \hat{d}, i)$ that is not rejected out of hand by the special rejection rule must satisfy $(\hat{\mathcal{V}}, \hat{\mathcal{V}}') = (\hat{\beta}\mathcal{G}, \hat{\beta}\mathcal{G}')$ for some (unknown) $\hat{\beta} \in \mathbb{Z}_q$.

In Game 2, we modify key generation so that the constant term of the polynomial f' is random, just as in TDH2a. We claim that $\Pr[E_2] = \Pr[E_1]$. We give the proof of this claim below in Lemma 2.

In Game 3, we drop the special rejection rule introduced above in Game 1. By the (ordinary) soundness property for the proof system from Section 2.1.2, $|\Pr[E_3] - \Pr[E_2]|$ is negligible.

Notice that Game 3 is identical to Game 0 of Theorem 1. Apart from proving the above claim that $\Pr[E_2] = \Pr[E_1]$, that completes the proof of Theorem 2.

Lemma 2. *We have* $\Pr[E_2] = \Pr[E_1]$.

Proof. The proof is a purely information-theoretic argument. It borrows a key idea from [DR24], but we present that idea here in a simplified form, following the general line of reasoning used in the proof of Lemma 1 based on simple linear algebra. In particular, we do not need to utilize the "H-coefficient technique".

We want to show that the outcomes in Games 1 and 2 are determined by random variables that have identical distributions. Just as in the proof of Lemma 1, since this is a purely information-theoretic argument, we can work exclusively with discrete logarithms to the base $\mathcal{G}$, rather than with group elements. Moreover, we assume that in both games we use the WI simulator to generate the decryption proofs.

In the key generation algorithm, we generate polynomials

$$f = \rho_0 + \rho_1 X + \cdots + \rho_{t-1} X^{t-1} \quad \text{and} \quad f' = \rho'_0 + \rho'_1 X + \cdots + \rho'_{t-1} X^{t-1}$$

where the values ρ_j for $j = 0, \ldots, t-1$ and ρ'_j for $j = 1, \ldots, t-1$ are generated at random. In Game 1, ρ'_0 is zero while in Game 2, ρ'_0 is set to a random value ρ^*.

Let us characterize the information that is revealed about these values during the course of the attack game.

From the key generation algorithm, the values

$$\epsilon_j \coloneqq \rho_j + \rho'_j\zeta \quad (j = 0, \ldots, t-1),$$

are revealed, where $\mathcal{G}' = \zeta\mathcal{G}$ and $\zeta \neq 0$. Just as we argued in the proof of Lemma 1, all other information revealed by key generation can be derived from this information.

Also just as we argued in the proof of Lemma 1, the decryption queries reveal no further information.

Also just as we argued in the proof of Lemma 1, for a given corruption query, which specifies an index $\ell \in \{1, \ldots, N\}$, the value $\alpha_\ell = \sum_j \rho_j \ell^j$ is revealed, from which the value α'_ℓ can be derived.

We can write all of this as a system of linear equations as follows:

$$\overbrace{\begin{pmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_{t-1} \\ \alpha_{\ell_1} \\ \vdots \\ \alpha_{\ell_s} \end{pmatrix}}^{V \coloneqq} = \overbrace{\begin{pmatrix} 1 & & & & & & \\ & 1 & & & \zeta & & \\ & & \ddots & & & \ddots & \\ & & & 1 & & & \zeta \\ 1 & \ell_1 & \cdots & \ell_1^{t-1} & & & \\ & & \vdots & & & & \\ 1 & \ell_s & \cdots & \ell_s^{t-1} & & & \end{pmatrix}}^{M \coloneqq} \cdot \begin{pmatrix} \rho_0 \\ \rho_1 \\ \vdots \\ \rho_{t-1} \\ \rho'_1 \\ \vdots \\ \rho'_{t-1} \end{pmatrix} + \begin{pmatrix} \rho'_0\zeta \\ 0 \\ \\ \vdots \\ \vdots \\ \\ 0 \end{pmatrix}. \tag{2}$$

Here, $\ell_1, \ldots, \ell_s$ are distinct, nonzero elements of $\mathbb{Z}_q$, where $s \leq t-1$. So the matrix has $2t-1$ columns and $t+s \leq 2t-1$ rows.

We claim that the rows of M are linearly independent. To see this, let us number the rows $1, \ldots, t+s$ and the columns $1, \ldots, 2t-1$. Consider any linear combination of the rows with scalars $c_1, \ldots, c_{t+s} \in \mathbb{Z}_q$ that produces the zero vector. We want to show that $c_1 = \cdots = c_{t+s} = 0$. Looking at columns $t+1, \ldots, 2t-1$ of M, it is clear that we must have $c_2 = \cdots = c_t = 0$. So ignoring rows $2, \ldots, t$, and looking at columns $1, \ldots, t$, it is clear that we must have $c_1 = c_{t+1} = \cdots = c_{t+s} = 0$ — this is because by properties of Vandermonde matrices, the vectors

$$(1, 0, \ldots, 0), (1, \ell_1, \ldots, \ell_1^{t-1}), \ldots, (1, \ell_s, \ldots, \ell_s^{t-1})$$

are linearly independent.

Since M has full rank, and the entries in the first column vector on the right-hand side of (2) are uniformly and independently distributed over $\mathbb{Z}_q$, it follows that the entries in the vector V on the left-hand side of (2) are also uniformly and independently distributed over $\mathbb{Z}_q$. (See also the discussion in the proof of Lemma 1 regarding the fact that some rows of M are chosen adaptively.)

To complete the proof, observe that the outcome of the attack is completely determined by:

(i) the random tape of the adversary,

(ii) the random value ζ,

(iii) the random values used in the encryption query,

(iv) the random values used by the WI simulator in the decryption queries, and

(v) the vector V.

The above argument shows that the values in (v) are independently and uniformly distributed, and independent of (i), (ii), (iii), and (iv), regardless of whether $\rho'_0 = 0$, as in Game 1, or $\rho'_0 = \rho^*$, as in Game 2. □

4 Simulation-CCA security

We present an alternative to the game-based definition of security in Section 2.3. This definition is a simulation-based definition that models threshold decryption as a *service*, and which may make it much easier to analyze systems that use threshold decryption. We then show that `TDH2a` provides simulation-CCA security under the same assumptions we used to prove CCA security, provided we implement symmetric encryption as $e = F(k) \oplus m$, where F is modeled as a programmable global random oracle.

4.1 Definition of simulation-CCA security

The definition presented here is adapted from Section 8.2 of [BBN+25]. The main differences are, unlike [BBN+25], we model *erasure-free* adaptive corruptions, we model encryptor corruptions, and we do not model context-dependent decryptions (which simplifies the definition).[3]

To define this notion of security, we describe both a **real-world** and an **ideal-world** attack game.

The real world.

- Setup: the challenger runs the key generation algorithm

$$(pk,\ pkc,\ sk_1, \ldots, sk_N) \xleftarrow{\$} G(N, t)$$

  and gives the public information (pk, pkc) to the adversary.

  The challenger also initializes the set $\mathcal{C}$ to the empty set — this will track the set of corrupted decryption servers.

- The adversary makes a series of *encryption*, *decryption*, and *corruption* queries. There are two types of corruption queries: *server-corruption* and *encryptor-corruption*.

  - An encryption query consists of a pair (m, d), where m is a message and d is associated data. This query is only allowed if $|\mathcal{C}| < t$. The challenger does the following:
    - it computes $c \xleftarrow{\$} E(pk, m, d)$;
    - it sends c to the adversary.
  - A decryption query consists of a tuple $(\hat{c}, \hat{d}, i)$, where $\hat{c}$ is a ciphertext, $\hat{d}$ is associated data, and $i \in [N] \setminus \mathcal{C}$. The challenger does the following:
    - it computes $c'_i \xleftarrow{\$} D(sk_i, \hat{c}, \hat{d})$;
    - it sends c'_i to the adversary.

[3] A somewhat different simulation-based definition of CPA security was briefly sketched in [CG99].

– A server-corruption query specifies an index $\ell \in [N] \setminus \mathcal{C}$. The challenger does the following:
  * it adds ℓ to $\mathcal{C}$;
  * it sends to the adversary sk_ℓ along with any random values generated in processing decryption queries associated with the server P_ℓ (i.e., all decryption queries of the form $(\cdot, \cdot, \ell)$).
– An encryptor-corruption query specifies the index j of an encryption query that has not been specified in any previous encryptor-corruption query.

  The challenger sends to the adversary the message that was encrypted in that query along with any random values generated in the process.

- At the end of the game, the adversary outputs a bit.

The ideal world. The ideal-world attack game is defined relative to a simulator *Sim*.

- The setup phase is the same as in the real world, except that instead of invoking $G(N, t)$, the challenger invokes *Sim* on input $(\texttt{setup}, N, t, \mathcal{C})$ to obtain (pk, pkc).

  In addition, the challenger initializes an associative array *Decrypted* to the empty map — for a given (c, d) arising from an encryption query, $Decrypted[c, d]$ will track the set of servers that have decrypted (c, d).

- To process an (allowable) encryption query (m, d), the challenger does the following:
  – instead of invoking $E(pk, m, d)$ to obtain the ciphertext c, it invokes *Sim* on input $(\texttt{enc}, |m|, d)$, where $|m|$ is the length of m;
  – it then checks if $(c, d) \in \mathsf{Domain}(Decrypted)$;
    * if not, it sets $Decrypted[c, d] \leftarrow \emptyset$ and sends c to the adversary;
    * otherwise, it "aborts".

- To process a decryption query $(\hat{c}, \hat{d}, i)$, the challenger does the following:
  – if $(\hat{c}, \hat{d}) \in \mathsf{Domain}(Decrypted)$, it adds i to $Decrypted[\hat{c}, \hat{d}]$;
  – instead of invoking $D(sk_i, \hat{c}, \hat{d})$ to obtain the decryption share c'_i, it invokes *Sim* on input $(\texttt{dec}, \hat{c}, \hat{d}, i, leakage)$.

  The value *leakage* will be discussed below.

- To process a server-corruption query ℓ, the challenger does the following:
  – it adds ℓ to $\mathcal{C}$;
  – it invokes *Sim* on input $(\texttt{dcor}, \ell, leakage)$.

  The value *leakage* will be discussed below.

- To process an encryptor-corruption query j, the challenger invokes *Sim* on input $(\texttt{ecor}, j, leakage)$. The value *leakage* will be discussed below.

As indicated above, the challenger may "abort". Concretely, this means that the challenger sends the special symbol `abort` to the adversary, and the adversary may not make any further queries.

We now discuss the value *leakage* given to the simulator in processing decryption and corruption queries. Consider the jth encryption query with corresponding message m and ciphertext/associated-data pair (c, d). Initially, we shall say that this encryption query is **protected**. Such a query may later become **compromised** in processing a decryption or corruption query when either

- $|\mathcal{C} \cup \mathit{Decrypted}[c, d]| \geq t$, or
- an encryptor-corruption query on the jth encryption query is invoked.

In processing either a decryption or corruption query, the value *leakage* is set to the set of all triples (c, d, m) corresponding to encryption queries that have just become compromised. Note that for a decryption or encryptor-corruption query, this set has size at most 1, while for a server-corruption query, this set may be larger.

Note that we allow t (or more) servers to be corrupted. Of course, at the point in time when t servers become corrupted, any remaining protected encryption queries become compromised. In both real and ideal worlds, we do not allow encryption queries after t servers have been corrupted — any such queries would anyway have to be classified as compromised from the start.

Definition of security. We say a threshold decryption scheme is **simulation-CCA secure** if there is an efficient simulator *Sim* such that no efficient adversary can effectively distinguish between the real-world attack game and the ideal-world attack game with *Sim*.

Relation to universal composability. While this simulation-based definition is not explicitly given in the universal composability (UC) framework [Can20], it is compatible with it, and can easily be used to analyze complex systems within that framework. The essential point is that the adversary in our definition plays the role of the environment in the UC framework. In addition, our simulator plays the role of the ideal world adversary in the UC framework, and our challenger in the ideal world plays the role of the ideal functionality in the UC framework.

Applications. Simulation-CCA security better models a system where all ciphertexts may eventually be decrypted, but we want to ensure that information about their contents leaks only when ciphertexts are legitimately compromised. As such, we believe that this simulation-based definition is in fact the most natural definition to work with in analyzing the security of larger systems that use a threshold decryption scheme.

4.2 Proving simulation-CCA security of `TDH2a`

We now show that `TDH2a` provides simulation-CCA security under the same assumptions we used to prove CCA security, provided we implement symmetric encryption as $e = F(k) \oplus m$, where F is modeled as a random oracle.

To do so, we have to define a simulator for the ideal world in Section 4.1, and prove that no efficient adversary can distinguish between the real and ideal worlds. To simplify the presentation,

we assume the message space $\boldsymbol{M}$ consists of all bit strings of some fixed length, and the random oracle F is a random map from $\boldsymbol{K}$ to $\boldsymbol{M}$.

It will be convenient to define two operations $eval_F$ and $program_F$. The operation $y \leftarrow eval_F(k)$ evaluates the random oracle F and returns its value at the input k. The operation $y \leftarrow program_F(k, x)$ works as follows: if no operation of the form $eval_F(k)$ or $program_F(k, \cdot)$ was previously invoked, it sets the value of F at k to x and returns `true`; otherwise, it returns `false`. We say that F **is probed at** k whenever an operation of the form $eval_F(k)$ or $program_F(k, \cdot)$ is performed.

We note that we can efficiently implement these operations by maintaining an associative array Map_F, initially empty, as follows:

- $eval_F(k)$:
  - if $k \notin \mathsf{Domain}(Map_F)$, set $Map_F(k) \xleftarrow{\$} \boldsymbol{M}$;
  - return $Map_F(k)$;
- $program_F(k, x)$:
  - if $k \notin \mathsf{Domain}(Map_F)$, set $Map_F(k) \leftarrow x$ and return `true`;
  - otherwise, return `false`.

Our simulator only needs to access F via these two operations. Our analysis works if the adversary also gets access to both of these operations. That is, all entities — challenger, adversary, and simulator — in both the real and ideal worlds access the random oracle via these operations. We are essentially working in the "programmable global random oracle model" of [CDG+18]. As such, our protocol has very robust composability properties.

Finally, the simulator. The simulator runs the actual key generation, encryption, and decryption algorithms without change, except as follows:

> for an encryption query, instead of computing $e = F(k) \oplus m$, it instead chooses $e \in \boldsymbol{M}$ at random; if and when this encryption query becomes compromised, the challenger will give the simulator a list of triples that include (c, d, m) for this ciphertext, and the simulator executes $program_F(k, e \oplus m)$.

4.3 Interlude: `TDH2a` as a KEM

To finish our proof of simulation-CCA security for `TDH2a`, we need to show that no efficient adversary can distinguish between the real and ideal worlds. This argument becomes quite easy if we first establish that a variant of `TDH2a` is CCA-secure threshold KEM (key encapsulation mechanism).

The syntax of a threshold KEM is very similar to that of a threshold decryption scheme. The main difference is that instead of encrypting a message m, we are just generating an encryption of a key k, which is an *output* of the encryption algorithm (rather than an input). Also, decryption shares are combined to output a key, rather than a message.

The CCA security property can be defined via an attack game very similar to that in Section 2.3.

> The challenger first runs the key generation algorithm
>
> $$(pk,\ pkc,\ sk_1, \ldots, sk_N) \xleftarrow{\$} G(N, t)$$

and gives the public information (pk, pkc) to the adversary. The challenger also chooses $b \in \{0,1\}$ at random.

Next, the adversary makes a sequence of encryption, decryption, and corruption queries.

- The adversary may make just one encryption query, which is of the form d, where d is associated data. The challenger computes the ciphertext $(c, k_0) \xleftarrow{\$} E(pk, d)$, chooses $k_1 \in \boldsymbol{K}$ at random, and gives (c, k_b) to the adversary.
- Each decryption query is of the form $(\hat{c}, \hat{d}, i)$, where $\hat{c}$ is a ciphertext, $\hat{d}$ is associated data, and $i \in \{1, \ldots, N\}$ is the index of an uncorrupted server P_i. A restriction is that if the adversary has already made its encryption query, then $(\hat{c}, \hat{d}) \neq (c, d)$. The challenger computes the decryption share $c'_i \xleftarrow{\$} D(sk_i, \hat{c}, \hat{d})$ and sends this to the adversary.
- The adversary may make at most $t-1$ corruption queries. Each corruption query is of the form $\ell \in \{1, \ldots, N\}$, where ℓ is the index of an uncorrupted server P_ℓ. In response, the challenger sends the secret key sk_ℓ of P_ℓ, along with any random values generated by P_ℓ in processing its decryption queries, to the adversary.

In addition, if we wish to model any hash functions as random oracles, then the adversary may also make queries to obtain the value of these random oracles at points of its choosing. The challenger will also internally access these random oracles when processing encryption and decryption queries.

At the end of the game, the adversary outputs $\hat{b} \in \{0,1\}$.

The adversary's **advantage** is defined to be $|\Pr[\hat{b} = b] - 1/2|$, and the KEM is said to be **CCA secure under erasure-free adaptive corruptions** if this advantage is negligible for all efficient adversaries.

4.4 The KEM version of `TDH2a`

The KEM version of `TDH2a` is easy to define.

- Key generation is exactly the same as in `TDH2a`.
- Encryption is the same, except that
  - m is not an input,
  - e is not computed, and is not included in the input to *GenPrf* or in the resulting ciphertext, and
  - k is included in the output.
- Decryption is the same, except for the lack of the e component of the ciphertext.
- The decryption share combiner algorithm is the same, except that after computing k, we stop and output k.

Let us call this scheme `TDH2a-kem`. For our immediate purposes, we do not need to prove any robustness properties for `TDH2a-kem` (although these certainly can be proven); rather, we just need to prove that `TDH2a-kem` is CCA secure under erasure-free adaptive corruptions.

Theorem 3. *Modeling the hash function used in the proof system from Section 2.1.2 as a random oracle, and assuming the DDH as in Section 2.1.1, and the security of H as a key derivation function as in Section 2.1.5, the scheme* `TDH2a-kem` *is CCA secure under erasure-free adaptive corruptions.*

The proof of Theorem 3 is almost identical to that of Theorem 1, and is omitted. (Note that one also needs to take into account the discussion in Section 2.5.1.)

4.5 Finishing the proof

Let $\mathfrak{A}$ be an efficient adversary. We now show that $\mathfrak{A}$ cannot distinguish between the real and ideal worlds. We do this by a series of games, where in each game we modify the challenger. In each Game i, we let X_i be the event that $\mathfrak{A}$ outputs 1 in that game.

Game 0. This is the real-world attack game.

Game 1. In this game, we add "duplicate rejection logic" that mimics that of the ideal world. The setup phase is the same as in the real world, except that a set S is initialized to empty. To process an encryption query (m, d), the challenger does the following:

- $c \leftarrow E(pk, m, d)$;
- it then checks if $(c, d) \in S$;
  - if not, it adds (c, d) to S and sends c to $\mathfrak{A}$;
  - otherwise, it "aborts";

Clearly, $|\Pr[X_0] - \Pr[X_1]| \leq Q_{\text{enc}}^2/(2q)$, where Q_{enc} is a bound on the number of encryption queries.

Game 2. We now make a conceptual change. Consider an encryption query (m, d). In Game 1, the ciphertext is computed by first computing $\mathcal{V}$, $\mathcal{V}'$, and k. Then we compute

$$x \leftarrow \mathit{eval}_F(k)$$
$$e \leftarrow x \oplus m$$

In Game 2, we instead compute

$$x \xleftarrow{\$} \boldsymbol{M}$$
$$(*) \quad \text{if not } \mathit{program}_F(k, x) \text{ then } x \leftarrow \mathit{eval}_F(k)$$
$$e \leftarrow x \oplus m$$

This is just an equivalent implementation of the random oracle, and so it is clear that $\Pr[X_1] = \Pr[X_2]$.

Game 3. In this game, for each encryption query, we drop the line marked $(*)$ in the above "encryption logic", and in addition, we add the following "compromise logic": at the point in time when this encryption query becomes compromised, we execute

$$\mathit{program}_F(k, x)$$

To implement this, the challenger incorporates the same logic as the ideal world challenger for determining when ciphertexts become compromised.

We may view Games 2 and 3 as operating on the same underlying probability space — that is, they each operate deterministically on the same random values.

Let k_j be the key generated in the jth encryption query in Game 3. Let Z be the event that, in Game 3, for some encryption query index j^*, either

- $k_{j^*} = k_j$ for some $j < j^*$, or
- $\mathfrak{A}$ probes F at k_{j^*} at any point in time before the j^*th encryption query is compromised.

It is straightforward to verify that for any choice of random values for which Z does not occur, the output of $\mathfrak{A}$ in Games 2 and 3 is identical. It follows that $|\Pr[X_2] - \Pr[X_3]| \leq \Pr[Z]$. We will show below that $\Pr[Z]$ is negligible, assuming `TDH2a-kem` is secure.

Game 4. We again make a conceptual change. We modify the above encryption logic to be

$$e \xleftarrow{\$} \boldsymbol{M}$$

and we modify the above compromise logic to be

$$\mathit{program}_F(k, e \oplus m)$$

It is clear that $\Pr[X_3] = \Pr[X_4]$. Moreover, it is also clear that this game is equivalent to the ideal world.

So we have reduced our problem to that of showing that $\Pr[Z]$ is negligible, assuming `TDH2a-kem` is secure, where Z is as defined above in Game 3. This is done by a "guessing argument": we guess at random the index j^* that will make Z hold. Let Z_* be the event that Z holds for this j^*. Clearly, $\Pr[Z] \leq Q_{\text{enc}} \cdot \Pr[Z_*]$, so it suffices to show that $\Pr[Z_*]$ is negligible. We do this via a sequence of games. In what follows, we call the j^*th encryption query the "target encryption query".

Game 3′. We modify Game 3 as follows: if t servers become corrupted, or if the target encryption query becomes compromised, the game is immediately halted (before completing the corruption or decryption query that triggers this condition). If Z'_* is the event corresponding to Z_* in Game 3′, it is clear that $\Pr[Z_*] = \Pr[Z'_*]$.

Game 3″. We now modify Game 3′ so that the key k associated with the target encryption query is replaced by a truly random key.

Let Z''_* be the event corresponding to Z_* in Game 3″. We present below an efficient adversary $\mathfrak{A}_{\text{kem}}$ such that $|\Pr[Z'_*] - \Pr[Z''_*]| \leq 2\epsilon$, where ϵ is the advantage of $\mathfrak{A}_{\text{kem}}$ in the CCA attack game for `TDH2a-kem`. It follows that $\Pr[Z''_*] \leq (Q_{\text{enc}} + Q_F)/|\boldsymbol{K}|$, where Q_F is a bound on the number of probes of the random oracle F made by $\mathfrak{A}$. Therefore, $\Pr[Z_*] \leq 2\epsilon + (Q_{\text{enc}} + Q_F)/|\boldsymbol{K}|$.

Adversary $\mathfrak{A}_{\text{kem}}$ is a wrapper around $\mathfrak{A}$, and incorporates the logic of the challenger in Game 3′ and the random oracle F. It works as follows.

- In the setup phase, when $\mathfrak{A}_{\text{kem}}$ receives (pk, pkc) from its challenger, it forwards this to $\mathfrak{A}$.
- Any non-target encryption query is processed directly by $\mathfrak{A}_{\text{kem}}$, using the logic of the challenger in Game 3′.
- When $\mathfrak{A}$ makes the target encryption query (m, d), adversary $\mathfrak{A}_{\text{kem}}$ *first* computes $x \xleftarrow{\$} \boldsymbol{M}$, $e \leftarrow x \oplus m$, and *then* makes an encryption query to its own challenger with associated data (e, d), obtaining (c, k).
- Suppose that at some time following the target encryption query, $\mathfrak{A}$ makes a decryption query of the form (c, d, i), where (c, d) is the ciphertext/associated-data pair from the target encryption query. Adversary $\mathfrak{A}_{\text{kem}}$ invokes its own challenger to make a *server-corruption query* on server P_i. Using the secret key of P_i, adversary $\mathfrak{A}_{\text{kem}}$ can itself compute the decryption share, and process any subsequent decryption or corruption query involving P_i. Note that this change does not affect in any way how $\mathfrak{A}_{\text{kem}}$ implements the logic determining which ciphertexts are classified as compromised; in particular, this query is still treated as a decryption query for that purpose.
- In processing any other decryption query from $\mathfrak{A}$ on a ciphertext/associated-data pair $(\hat{c}, \hat{d})$, where $\hat{c} = (\hat{\mathcal{V}}, \hat{\mathcal{V}}', \hat{e}, \hat{\pi})$, adversary $\mathfrak{A}_{\text{kem}}$ makes a corresponding decryption query to its own challenger on the ciphertext $(\hat{\mathcal{V}}, \hat{\mathcal{V}}', \hat{\pi})$ with associated data $(\hat{e}, \hat{d})$ (except for those mentioned above that it processes itself).
- Adversary $\mathfrak{A}_{\text{kem}}$ passes any server-corruption queries from $\mathfrak{A}$ directly to its own challenger (except for those mentioned above that it processes itself).
- Adversary $\mathfrak{A}_{\text{kem}}$ processes all encryptor-corruption queries from $\mathfrak{A}$ itself using the logic of the challenger in Game 3′. Note that an encryptor-corruption query on the target encryption query immediately halts the game.
- When the game ends, $\mathfrak{A}_{\text{kem}}$ outputs 1 iff the condition defining Z_* holds.

As we have defined it, adversary $\mathfrak{A}_{\text{kem}}$ satisfies the restrictions in the CCA attack game for `TDH2a-kem`. Specifically, $\mathfrak{A}_{\text{kem}}$ makes at most one encryption query, and never makes a decryption query on the ciphertext/associated-data pair associated with that query. In addition, although $\mathfrak{A}_{\text{kem}}$ converts decryption queries on (c, d) from $\mathfrak{A}$ to corruption queries, it can make at most $t - 1$ corruption queries before the target encryption query is considered compromised. Moreover, one can easily verify that when the hidden bit b of $\mathfrak{A}_{\text{kem}}$'s challenger is 0 (so k is a real key), $\mathfrak{A}_{\text{kem}}$ outputs 1 with probability $\Pr[Z'_*]$; similarly, when the hidden bit b of $\mathfrak{A}_{\text{kem}}$'s challenger is 1 (so k is a random key), $\mathfrak{A}_{\text{kem}}$ outputs 1 with probability $\Pr[Z''_*]$.

So we have proven:

Theorem 4. *Suppose the symmetric encryption scheme in `TDH2a` is implemented as $e = F(k) \oplus m$, where F is modeled as a programmable global random oracle. Suppose that `TDH2a-kem` is CCA secure under erasure-free adaptive corruptions, and that $1/|\boldsymbol{K}|$ is negligible. Then `TDH2a` is simulation-CCA secure.*

Moreover, Theorem 3 establishes the hypothesis that `TDH2a-kem` is CCA secure under erasure-free adaptive corruptions. We remark that Theorem 4 also holds for `TDH2abc` (indeed, Theorem 3 clearly holds for the KEM variant of `TDH2abc`).

Acknowledgements

The author thanks Sourav Das for enlightening discussions regarding [DR24], as well as Benoît Libert for helpful discussions.

References

[ABP15] M. Abdalla, F. Benhamouda, and D. Pointcheval. Disjunctions for hash proof systems: New constructions and applications. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 69–100. Springer, 2015. https://doi.org/10.1007/978-3-662-46803-6_3. Full version at https://eprint.iacr.org/2014/483.

[BBN+25] D. Boneh, B. Bünz, K. Nayak, L. Rotem, and V. Shoup. Context-dependent threshold decryption and its applications. Cryptology ePrint Archive, Paper 2025/279, 2025. https://eprint.iacr.org/2025/279.

[BK25] R. Bacho and A. Kavousi. SoK: Dlog-based distributed key generation. Cryptology ePrint Archive, Paper 2025/819, 2025. https://eprint.iacr.org/2025/819.

[BS23] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography (v0.6)*. 2023. Available at https://toc.cryptobook.us/.

[Can20] R. Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020. https://doi.org/10.1145/3402457. Also at https://eprint.iacr.org/2000/067.

[CDG+18] J. Camenisch, M. Drijvers, T. Gagliardoni, A. Lehmann, and G. Neven. The wonderful world of global random oracles. Cryptology ePrint Archive, Report 2018/165, 2018. https://eprint.iacr.org/2018/165.

[CG99] R. Canetti and S. Goldwasser. An efficient *Threshold* public key cryptosystem secure against adaptive chosen ciphertext attack. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 90–106. Springer, 1999. https://doi.org/10.1007/3-540-48910-X_7.

[CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998. https://doi.org/10.1007/BFB0055717.

[CS01] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. Cryptology ePrint Archive, Paper 2001/108, 2001. https://eprint.iacr.org/2001/108.

[CS02] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In L. R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2002. https://doi.org/10.1007/3-540-46035-7_4.

[CS03] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, 2003. https://doi.org/10.1137/S0097539702403773.

[DLN+21] J. Devevey, B. Libert, K. Nguyen, T. Peters, and M. Yung. Non-interactive CCA2-secure threshold cryptosystems: Achieving adaptive security in the standard model without pairings. In *PKC 2021*, volume 12710 of *LNCS*, pages 659–690. Springer, 2021. https://doi.org/10.1007/978-3-030-75245-3_24. Full version at https://eprint.iacr.org/2021/630.

[DR24] S. Das and L. Ren. Adaptively secure BLS threshold signatures from DDH and co-CDH. In L. Reyzin and D. Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VII*, volume 14926 of *Lecture Notes in Computer Science*, pages 251–284. Springer, 2024. https://doi.org/10.1007/978-3-031-68394-7_9. Also at https://eprint.iacr.org/2023/1553.

[DRY25] S. Das, L. Ren, and Z. Yang. Adaptively secure threshold ElGamal decryption from DDH. Cryptology ePrint Archive, Paper 2025/1477, 2025. https://eprint.iacr.org/2025/1477.

[KW15] E. Kiltz and H. Wee. Quasi-adaptive NIZK for linear subspaces revisited. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 101–128. Springer, 2015. https://doi.org/10.1007/978-3-662-46803-6_4. Full version at https://eprint.iacr.org/2015/216.

[LPJY14] B. Libert, T. Peters, M. Joye, and M. Yung. Non-malleability from malleability: Simulation-sound quasi-adaptive NIZK proofs and CCA2-secure encryption from homomorphic signatures. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2014. https://doi.org/10.1007/978-3-642-55220-5_29. Full version at https://eprint.iacr.org/2013/691.

[LY12] B. Libert and M. Yung. Non-interactive cca-secure threshold cryptosystems with adaptive security: New framework and constructions. In R. Cramer, editor, *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, volume 7194 of *Lecture Notes in*

Computer Science, pages 75–93. Springer, 2012. https://doi.org/10.1007/978-3-642-28914-9_5. Full version at https://perso.ens-lyon.fr/benoit.libert/threshold-framework-full-version.pdf.

[LY20] B. Libert and M. Yung. Adaptively secure non-interactive CCA-secure threshold cryptosystems: Generic framework and constructions. *J. Cryptol.*, 33(4):1405–1441, 2020. https://doi.org/10.1007/S00145-020-09350-3.

[OY91] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In L. Logrippo, editor, *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, 1991*, pages 51–59. ACM, 1991. https://doi.org/10.1145/112600.112605.

[SG98] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In K. Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, volume 1403 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1998. https://doi.org/10.1007/BFB0054113. Subsequently published in *J. Cryptol.*, 15(2), pages 75–86, 2002.

A A proof system and WI simulator

Here, we sketch the details of the proof system in Section 2.1.3, along with the WI simulator.

The proof system is the standard Fiat-Shamir transformation applied to the standard Sigma protocol (see Section 19.5.3 of [BS23]) for the relation

$$\{((\alpha,\alpha'),(\mathcal{V},\mathcal{V}',\mathcal{U},\mathcal{W})) \in \mathbb{Z}_q^2 \times \mathbb{G}^4 : \mathcal{U} = \alpha\mathcal{G} + \alpha'\mathcal{G}' \text{ and } \mathcal{W} = \alpha\mathcal{V} + \alpha'\mathcal{V}'\}$$

Given the statement $(\mathcal{V},\mathcal{V}',\mathcal{U},\mathcal{W})$ along with the witness (α,α'), we generate a proof as follows:

1. Generate $\dot{\alpha},\dot{\alpha}' \in \mathbb{Z}_q$ at random and compute $\dot{\mathcal{U}} \coloneqq \dot{\alpha}\mathcal{G} + \dot{\alpha}'\mathcal{G}'$ and $\dot{\mathcal{W}} \coloneqq \dot{\alpha}\mathcal{V} + \dot{\alpha}'\mathcal{V}'$.
2. Compute c as a hash of $(\mathcal{V},\mathcal{V}',\mathcal{U},\mathcal{W},\dot{\mathcal{U}},\dot{\mathcal{W}})$
3. Compute $\ddot{\alpha} \coloneqq \dot{\alpha} + c\alpha$ and $\ddot{\alpha}' \coloneqq \dot{\alpha}' + c\alpha'$.

The proof is $(\dot{\mathcal{U}},\dot{\mathcal{W}},\ddot{\alpha},\ddot{\alpha}')$.

To verify such a proof, one checks that

$$\ddot{\alpha}\mathcal{G} + \ddot{\alpha}'\mathcal{G}' = \dot{\mathcal{U}} + c\mathcal{U} \quad \text{and} \quad \ddot{\alpha}\mathcal{V} + \ddot{\alpha}'\mathcal{V}' = \dot{\mathcal{W}} + c\mathcal{W}.$$

Since the Sigma protocol underlying the above proof system is special honest verifier zero knowledge, it follows that the Sigma protocol itself provides witness independence, and that implies the existence of a perfect, straightline, but inefficient zero knowledge simulator. For concreteness, we describe such a "WI simulator" explicitly and give a self-contained analysis.

As discussed in Section 2.1.3, whenever we need to generate a proof for a statement $(\mathcal{V},\mathcal{V}',\mathcal{U},\mathcal{W})$, it will be the case that $(\mathcal{V},\mathcal{V}') = (\beta\mathcal{G},\beta\mathcal{G}')$ for some β. Since the simulator need not be efficient, we may assume that it knows

- ζ such that $\mathcal{G}' = \zeta\mathcal{G}$, and
- ϵ such that $\mathcal{U} = \epsilon\mathcal{G}$, so $\epsilon = \alpha + \zeta\alpha'$.

Note also that $\mathcal{W} = \beta\mathcal{U} = \beta\epsilon\mathcal{G} = \epsilon\mathcal{V}$.

For an honestly generated proof, we have: $\dot{\mathcal{U}} = \dot{\alpha}\mathcal{G} + \dot{\alpha}'\mathcal{G}' = \dot{\epsilon}\mathcal{G}$, where

$$\dot{\epsilon} = \dot{\alpha} + \zeta\dot{\alpha}'. \tag{3}$$

We also have $\dot{\mathcal{W}} = \dot{\epsilon}\mathcal{V}$.

Finally, we have

$$\ddot{\alpha} = \dot{\alpha} + c\alpha \tag{4}$$

and

$$\ddot{\alpha}' = \dot{\alpha}' + c\alpha'. \tag{5}$$

Adding ζ times (5) to (4), we get

$$\ddot{\alpha} + \zeta\ddot{\alpha}' = \dot{\epsilon} + c\epsilon. \tag{6}$$

By (3), $\dot{\epsilon}$ and $\dot{\alpha}'$ are random and independent. Since c is a function of $\dot{\epsilon}$, it follows from (5) that $\dot{\epsilon}$ and $\ddot{\alpha}'$ are random and independent. It follows that the following simulator is perfect:

1. Generate $\dot{\epsilon} \in \mathbb{Z}_q$ at random.
2. Compute $\dot{\mathcal{U}} \coloneqq \dot{\epsilon}\mathcal{G}$ and $\dot{\mathcal{W}} \coloneqq \dot{\epsilon}\mathcal{V}$.
3. Compute c as the hash of $(\mathcal{V}, \mathcal{V}', \mathcal{U}, \mathcal{W}, \dot{\mathcal{U}}, \dot{\mathcal{W}})$.
4. Generate $\ddot{\alpha}' \in \mathbb{Z}_q$ at random and solve (6) for $\ddot{\alpha}$.
5. Output the proof $(\dot{\mathcal{U}}, \dot{\mathcal{W}}, \ddot{\alpha}, \ddot{\alpha}')$.

In discussing adaptive corruptions, the following observation is helpful. Given a proof $(\dot{\mathcal{U}}, \dot{\mathcal{W}}, \ddot{\alpha}, \ddot{\alpha}')$ and a witness (α, α'), we can easily obtain the random values $(\dot{\alpha}, \dot{\alpha}')$ generated while constructing the proof from the equations (4) and (5).