Statement-Oblivious Threshold Witness Encryption
(Full Version)

Sebastian Faust[1], Carmit Hazay[2], David Kretzler[1], and Benjamin Schlosser[1]

[1] Technical University of Darmstadt, Germany
{first.last}@tu-darmstadt.de
[2] Bar-Ilan University, Israel
carmit.hazay@biu.ac.il

Abstract. The notion of witness encryption introduced by Garg et al. (STOC'13) allows to encrypt a message under a statement x from some NP-language $\mathcal{L}$ with associated relation $(x, w) \in \mathcal{R}$, where decryption can be carried out with the corresponding witness w. Unfortunately, known constructions for general-purpose witness encryption rely on strong assumptions, and are mostly of theoretical interest. To address these shortcomings, Goyal et al. (PKC'22) recently introduced a blockchain-based alternative, where a committee decrypts ciphertexts when provided with a valid witness w. Blockchain-based committee solutions have recently gained broad interest to offer security against more powerful adversaries and construct new cryptographic primitives.
We follow this line of work, and propose a new notion of *statement-oblivious* threshold witness encryption. Our new notion offers the functionality of committee-based witness encryption while additionally hiding the statement used for encryption. We present two ways to build statement-oblivious threshold witness encryption, one generic transformation based on anonymous threshold identity-based encryption (A-TIBE) and one direct construction based on bilinear maps. Due to the lack of efficient A-TIBE schemes, the former mainly constitutes a feasibility result, while the latter yields a concretely efficient scheme.

Keywords: Threshold Witness Encryption, Statement Obliviousness, Committee-Based Decryption, Threshold Tag-Based Encryption

1 Introduction

The notion of *witness encryption* as introduced by Garg et al. [1] allows a party to encrypt a message m under some problem instance x such that the ciphertext can only be decrypted by someone holding a witness w. There are countless applications of witness encryption ranging from public key encryption with fast key generation, attribute-based encryption for general circuits [1], to using it for encrypting a prize for solving an NP-hard puzzle like the millennium problems, or

achieving fairness in MPC [2]. More formally, witness encryption is defined for an NP language $\mathcal{L}$ with associated relation $(x, w) \in \mathcal{R}$, where x is the *statement* and w is the corresponding *witness*. Security as defined by Garg et al. [1] states that for any ciphertext that was created for x *not in* the language $\mathcal{L}$, ciphertexts do not reveal information about the encrypted message. While this notion only deals with statements that are not in the language, Goldwasser et al. [3] introduced the notion of *extractable witness encryption* stating that even for a statement *in* the language, ciphertexts hide the message.

Although great progress has been made over the last years [1, 3, 4, 5, 6, 7], witness encryption still has limitations. First, known constructions rely on strong assumptions like multilinear maps [1, 3, 5, 6], indistinguishability obfuscation [4] or cryptographic invariant maps [7], and its constructions are not practically efficient yet. Second, even the stronger notion of extractable witness encryption does not hide the statement for which the ciphertext was created. This rules out interesting applications that require the statement to be private until decryption takes place, as it may disclose sensitive information.

The first shortcoming of state-of-the-art witness encryption can be circumvented via so-called *extractable Witness Encryption on Blockchains* (eWEB) put forward by Goyal et al. [8]. It is based on a blockchain following a recent trend in cryptography, where constructions leverage the power of blockchains, e.g., [2, 9, 10, 11, 12]. In the context of witness encryption, this results in a shift from relying on strong number theoretic assumptions to relying on an honest quorum of users within a committee. This trend is further fueled by a line of work that presents constructions of how such committees can be obtained in a blockchain setting [13, 14, 10].

In a nutshell, the scheme of [8] works as follows. Parties encrypt a message by secret sharing it to a committee and labeling the shares with a statement x. To decrypt, parties need to send a witness to the committee proving that x is in the language $\mathcal{L}$ and getting the secret shares back. While the construction of Goyal et al. is certainly more efficient than standard general-purpose witness encryption, the downside of their solution is the storage complexity of the committee, which grows linearly with the number of ciphertexts. Improving on the approach of [8], [9] propose as an application for their large-scale non-interactive threshold cryptosystem a solution, in which the decryption committee stores only secret key shares of a labeled threshold encryption scheme. The committee receives ciphertext-witness-pairs and decrypts only if the witness corresponds to the statement encoded as the label of the ciphertext. This reduces the storage complexity to be only constant. Following [8, 9], Campanelli et al. [10] presents a similar construction called *Blockchain Witness Encryption (BWE)*. However, their construction is not practical (e.g., for each encryption a smart contract deployment is required).

In this work, we start with the approach of [9], which we abstractly call *threshold witness encryption*, and address the second shortcoming by a new feature called *statement obliviousness*, which guarantees that the statement is hidden given the ciphertext. This new feature allows us to extend applications of

standard (threshold) witness encryption with an additional privacy property. For instance, we can construct time-lock encryption from witness encryption, as proposed by [6], without leaking the concrete time at which a decryption can happen to third parties, or we can construct a dead-man's switch, as proposed by [8], without revealing for which person it was created. Moreover, this feature enables a new class of applications that inherently require the privacy property and are not covered by standard witness encryption. As a concrete example, imagine a user wants to buy some shares of a company or some tokens on a Decentralized Finance (DeFi) trading platform, once the price of the asset reaches a certain value; however, without the necessity of having to stay online. Privacy is an important aspect in this scenario, since revealing information, e.g., the intended purchase price, could lead to financial disadvantages, e.g., due to insider trading. To support the described scenario, the user can exploit statement-oblivious threshold witness encryption in the following way. The user encrypts its transaction with the desired share price as statement and a signature of a trusted price oracle service as the required witnesses. Trusted price oracles are already available in the DeFi ecosystem and heavily used for building various financial products. The ciphertext is sent to the user's broker who repeatedly requests the signed current share price from the oracle service, attempts decryption, and, if this gives a valid transaction, executes the trade. For decrypting, the broker sends the ciphertext together with the current share price to the decryption committee. As the statement is hidden, no one, not even the oracle service, learns the desired share price until the transaction is successfully decrypted. Due to the required signature of the oracle service, the broker cannot send incorrect share prices to the decryption committee. We provide more details about use-cases of our new security feature in Section 9.

While [8] and [9] tackled the first limitation and present more efficient constructions that are effectively the same as witness encryption, both schemes still suffer from the fact that the statement is public. In this work, we address the privacy feature mentioned above. To this end, we introduce a novel notion that we call *statement-oblivious threshold witness encryption (SO-TWE)* and show how to instantiate it.

1.1 Contribution

We start by giving a summary of our contribution and defer an high-level overview of our constructions as well as a discussion of the technical challenges to the technical overview.

Primitive definition We introduce the notion of *statement-oblivious threshold witness encryption (SO-TWE)*. This primitive provides effectively the same functionality as witness encryption while requiring a committee with a fixed number of corrupted parties as typically done in threshold cryptography. As we envision the committee to perform decryption on request, we define a *security notion against chosen-ciphertext attacks* (CCA) which is at least as strong as

the notion of extractability for threshold witness encryption. In addition, the *statement-obliviousness* property guarantees that the statement used to generate a ciphertext is hidden. We provide a formal security game combining the CCA security with our new statement-obliviousness property.

We do not follow up on the existing notions of extractable Witness Encryption on Blockchains, proposed by [8], or Blockchain Witness Encryption, proposed by [10], as both notions are tied to the blockchain setting. We take a more general approach by using the committee to achieve witness encryption without defining the origin of the committee. In contrast to earlier works, however, our notion considers only static corruptions.

Instantiating SO-TWE We show how to instantiate SO-TWE via a series of transformations, as depicted in Figure 1. For all constructions and transformations, we provide formal security proofs. As a first step, we introduce the notion of *oblivious threshold tag-based encryption (O-TTBE)* as an extension of standard threshold tag-based encryption as presented in [15]. Similar to statement-obliviousness, obliviousness in this context ensures that the tag used for encryption is hidden. Then, we present a general transformation from CCA secure O-TTBE to CCA secure SO-TWE.

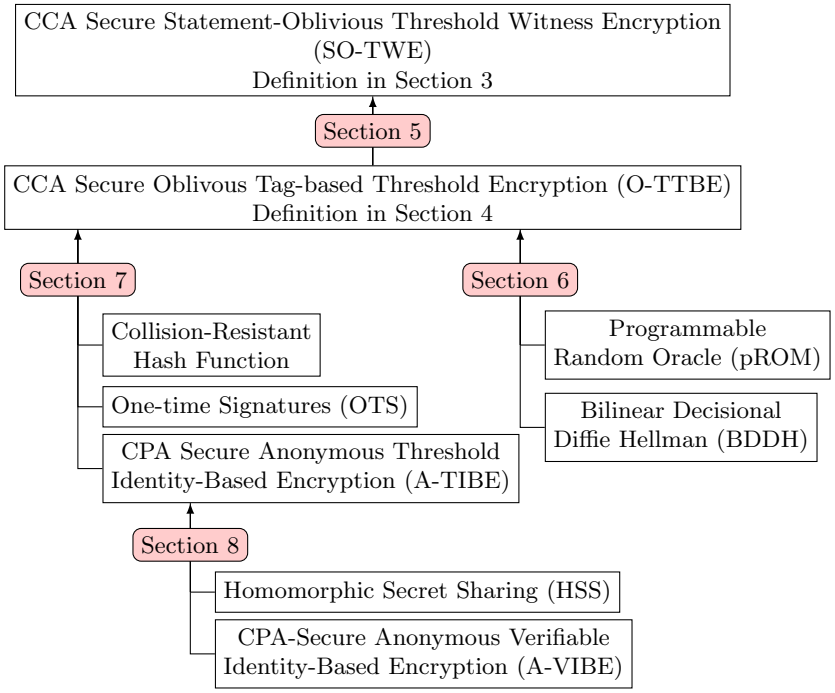

Fig. 1. The Landscape of Our Contributions.

As a second step, we show two ways to construct CCA secure O-TTBE schemes. First, we generically build a O-TTBE scheme from collision-resistant hash functions, one-time signatures and CPA secure anonymous threshold identity-based encryption (A-TIBE). To the best of our knowledge, there are constructions for anonymous identity-based encryption [16, 17] and threshold identity-based encryption [18], but there is no construction of an A-TIBE scheme. The techniques used for anonymous IBE do not allow for a straightfoward thresholdization via secret sharing while maintaining a high threshold and non-interactive decryption at the same time. As a feasibility result, we show how to instantiate A-TIBE from non-threshold anonymous identity-based encryption, a signature scheme and homomorphic secret sharing (HSS). This transformation follows [10] which constructs non-anonymous threshold identity-based encryption from HSS. While proving the security of our construction, we discovered a gap in the analysis of [10]. In particular, the construction in [10] allows corrupted parties to trick honest parties into accepting invalid identity keys, and hence, does not provide *key generation consistency*. We propose a solution to fix this gap. While the A-TIBE-based construction constitutes a feasibility result, we emphasize that any progress in constructing these building blocks, e.g., in terms of efficiency, immediately yields more efficient constructions of SO-TWE.

As a second way, we present a concretely efficient instantiation of O-TTBE in the random oracle model. Our construction extends Hash-ElGamal with a bilinear mapping and efficient non-interactive zero knowledge arguments. The resulting scheme is concretely efficient in terms of ciphertext size and bilinear mapping evaluations. The construction also yields the first efficient threshold witness encryption scheme that additionally achieves statement obliviousness. This is because our generic transformation from O-TTBE to SO-TWE only adds simple hash function evaluations and a check of the witness relation. We formally prove the security of this construction via a reduction to the *Decision Bilinear Diffie-Hellman assumption*.

1.2 Technical Overview

In this section, we outline the main techniques used to construct SO-TWE and discuss the major challenges.

Emulation of the witness encryption functionality. We consider the setup of a SO-TWE scheme to be executed by a trusted dealer or via a distributed key generation protocol. During the setup, the public key and the verification key are published while the secret key shares are distributed to the committee members. It is assumed that an adversary can statically corrupt a subset of the committee members. We allow the adversary to corrupt all but one committee member. Upon corruption the adversary takes full control over the committee members, and hence, learns their secret key shares. Users can encrypt messages non-interactively based on the public key and a self-chosen statement. Decryption is performed in an interactive way via a request-response protocol. To this end, a user sends the ciphertext, a statement candidate and a witness to the committee. All committee members compute and send their decryption shares to the

user, who attempts to combine the shares to the actual message. This will only be successful if it receives sufficiently many valid decryption shares and the witness relation has been verified successfully. Further, the statement-obliviousness property provides that the combined shares will only yield the original plaintext if the statement candidate used for the decryption is the same as the one used for encryption. While emulating the functionality of witness encryption using a committee-based approach seems to be easy at first glance, achieving statement obliviousness in combination with CCA security is highly non-trivial, as discussed next.

Achieving obliviousness in the CCA-setting. Due to the committee setting in which decryption is executed on request, we require security against chosen-ciphertext attacks (CCA). The major challenge is to simultaneously guarantee CCA security and achieve our new notion of statement obliviousness. A common technique to achieve CCA security in the threshold setting is to incorporate ciphertext validation before decryption [19, 18, 20, 21, 22]. The validation ensures that each decryption request issued by the adversary in the security game is either declined or yields exactly the original plaintext created by some user. This feature is required by the security proofs of known CCA secure threshold constructions, e.g., to prevent the adversary from exploiting homomorphisms in the group structure to decrypt valid ciphertexts that contain related messages. The difficulty in our setting is that the decryption committee may not know the statement used for encryption. In fact, the information if the correct statement has been used for decryption must not be leaked before decryption is completed. Any such leakage would allow corrupted servers to break the obliviousness property. It follows that we have to allow for multiple decryptions, with different statements, of the same ciphertext, and hence, cannot follow the standard approach of previous work. The described scenario makes it highly challenging to achieve obliviousness in combination with CCA security in the threshold setting. In particular, the challenge is to render decryptions useless for statements different than the one used for encryption despite applying the correct secret key shares when generating the decryption shares. Prior CCA-secure encryption schemes apply the secret key (shares) during decryption only after ensuring that the resulting (combined) decryption yields exactly the original message. Hence, we cannot use existing approaches to solve the described challenge.

SO-TWE from oblivious threshold tag-based encryption (cf. Section 5). As a first step towards SO-TWE, we present a transformation from a primitive called *oblivious threshold tag-based encryption (O-TTBE)*. To this end, we first extend the standard notion of threshold tag-based encryption presented by Arita and Tsurudome [15] with an obliviousness property. Similar to statement-obliviousness, obliviousness for a tag-based encryption scheme requires that two ciphertexts created with different tags cannot be distinguished.

Our first transformation takes a CCA secure O-TTBE scheme in order to construct SO-TWE. The high-level idea is to use the hash of the statement as a tag for the O-TTBE scheme. For decryption, a user needs to provide a statement candidate together with a corresponding witness. The decryption servers

first check if the witness is valid and then use the hash of the provided statement candidate as the tag in the decryption of the O-TTBE scheme. The statement-obliviousness property is directly obtained from the obliviousness property of the O-TTBE scheme but constructing CCA secure O-TTBE still faces the challenges explained above. As depicted in Figure 1, we follow two different paths to overcome these challenges and to construct CCA secure O-TTBE as described below.

O-TTBE from programmable random oracles and bilinear maps (cf. Section 6). In general, independently of the obliviousness setting, the major difficulty when proving CCA security is to answer decryption queries without knowledge of the secret key. When instantiating O-TTBE from black-box primitives, this task is realized by using oracles of the underlying primitive in the reduction. For example, in our transformation from O-TTBE to SO-TWE the reduction to O-TTBE uses the decryption oracle of the O-TTBE security game to answer decryption queries of the SO-TWE adversary. When combining CCA security with an obliviousness property, we additionally face the discussed challenge to answer different decryption queries for the same ciphertext. Here, a random looking value needs to be returned except for the decryption query that contains the tag used for encryption. For a concrete O-TTBE scheme, we need to address both challenges in parallel. Due to the strict ciphertext validation used in existing CCA secure encryptions schemes (e.g., [19, 18, 20, 21, 22]) extending these schemes to support tag obliviousness cannot be done in a straightforward way.

We propose a new construction starting from CPA secure Hash-ElGamal, which is a variant of classical ElGamal [23]. In Hash-ElGamal, the encryption algorithm given a message m samples a random exponent a and outputs two elements $A = g^a$ and $M = m \oplus H(X^a)$ for a group generator g, a random oracle H, and a public key $X = g^x$. In the threshold setting, the secret key x is secret shared among the decryption servers. The decryption shares of the servers are calculated as $d_i := A^{x_i}$, where x_i is the share of the i-th server. We apply an extension to this scheme that allows us to solve both aforementioned challenges at once. We do so by applying a random offset T to A in both, encryption and decryption. This offset is unique for each ciphertext-tag pair to obtain random values from decryption for tags different to the one used for encryption. When applying the offset via multiplication or exponentiation, e.g., $M = m \oplus H(X^{a \cdot T})$, an adversary can easily perform an homomorphic attack, i.e., $A^{x_i \cdot T} = (A^{x_i \cdot T'})^{\frac{T}{T'}}$. In order to prevent this, we apply the offset using a bilinear mapping e, i.e., $M = m \oplus H(e(T, X^a))$.

Further, we ensure that a ciphertext component A cannot be reused in different ciphertexts expect by the party that generated A, and hence, knows the plaintext anyway. We do so, by adding a non-interactive zero-knowledge argument of knowledge of a to the ciphertext. The second ciphertext component, M, is used for computing the challenge value of the non-interactive zero-knowledge argument, in order to link this component to the zero-knowledge argument. In classical ElGamal-based schemes, adding a zero-knowledge argument of knowledge of a to the ciphertext is not sufficient to achieve CCA security, as demon-

strated in detail by [19]. Instead, it is necessary to provide an additional trapdoor to solve the general challenge of CCA security, to answer decryption queries. Interestingly, in our construction, the tag-dependent offset does not only give us tag obliviousness but also provides us with such a trapdoor for free. In particular, in the reduction, we can simulate the random oracle used to compute the offset such that we learn the discrete logarithm of all offsets sampled by the random oracle. This allows us to compute $e(T, X^a)$ via $e(A, X)^{\log_g(T)}$. We elaborate further on the concrete challenges and the intuition of our construction in Section 6 before presenting the formal specification.

Despite being the first instantiation of O-TTBE, our construction yields a concretely efficient scheme. The ciphertexts consist of a bitstring with length equal to the message length, a group element of the bilinear mapping's base group and two exponents (in $\mathbb{Z}_q$, where q is the bilinear group's order). Decryption shares consist of one group element in the mapping's target group and two exponents. Encryption requires a single evaluation and decryption three evaluations of the bilinear map.

O-TTBE from anonymous threshold identity-based encryption (cf. Section 7). While the construction described in the previous paragraph yields an efficient scheme, we also present a generic solution. Boneh et al. [18] show how to achieve CCA security from one-time signatures and CPA secure identity-based encryption. Following this approach, we achieve CCA security in the threshold setting by combining one-time signatures with CPA secure *anonymous threshold identity-based encryption* (A-TIBE). The anonymity property of the TIBE is utilized to achieve obliviousness of the TTBE scheme. The high-level idea is to encode the tag into the identity of the IBE ciphertext. Since the anonymity property guarantees that no information about the identity can be obtained from the ciphertext, the tag stays hidden as well. Only the decryption with the correct tag, i.e., with the identity key corresponding to the tag, reveals information about the plaintext.

Constructing A-TIBE (cf. Section 8). As a final step, we explore two directions to obtain anonymous threshold identity-based encryption (A-TIBE). First, we present a black-box construction based on homomorphic secret sharing (HSS). The same approach was used by Campanelli et al. [10] in order to construct threshold IBE without anonymity. When exploring this direction, we discovered a gap in the security analysis of [10]. The construction in [10] does not provide *key generation consistency*, a security property that enables parties to validate correctness of received identity keys. Without that property, maliciously corrupted committee members can provide arbitrary identity key shares. This may result in an incorrect identity key such that the decryption of some ciphertext yields a different plaintext than the originally encrypted message. As such an attack is not possible in the non-threshold setting, standard IBE does not provide means to validate identity keys. It follows that the straightforward thresholdization of IBE using HSS is not sufficient to provide a secure threshold IBE scheme.

To overcome this problem, we propose a new IBE primitive with an additional verifiability property. Verifiable IBE contains a check if an identity key

is computed correctly which may be of independent interest in other settings where malicious security is required. Such a scheme can be built from a standard IBE scheme together with an existentially unforgeable signature scheme. Eventually, we construct anonymous threshold IBE by executing the key generation algorithm of the verifiable IBE scheme within HSS. We provide a formal proof showing security of the construction, including the discussed identity key generation consistency property. We note that in this black-box construction, we need to consider general-purpose HSS like [10].

Finally, we explore the transformation of the concrete anonymous non-threshold IBE scheme of Boyen and Waters [16]. The challenge in this transformation is that the identity key generation requires multiplication of secret values and freshly chosen randomness that needs to remain private. A direct secret sharing of these values pose some challenges which we discuss in Appendix H. While general-purpose secure multi-party computation can solve this task, we aim for a threshold IBE scheme that requires no interaction during identity key generation. We point out and discuss two ways how the aforementioned issues can be tackled and leave formal specifications and security analyses of these approaches to future work.

2 Preliminaries

Here, we present the most important primitives. Throughout this work, we denote the security parameter by $\kappa \in \mathbb{N}$. We denote the set $\{1, \ldots, k\}$ as $[k]$. For a negligible function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$, it holds that for every $c \in \mathbb{N}$ there exists a $n_0 \in \mathbb{N}$ such that for all $n > n_0$: $|\mathsf{negl}(n)| < \frac{1}{n^c}$. For the sake of expressiveness, we often denote a negligible function by negl. We use the abbreviation PPT to denote a probabilistic polynomial-time algorithm. An NP language L is a language that can be decided by a deterministic Turing machine M in polynomial time. More precisely, a language L is in NP iff there exist a deterministic Turing machine running in polynomial time in the length of the first input such that for all $x \in \text{Ł}$ there exists a string y such that $M(x, y) = 1$ and for all $x \notin \text{Ł}$ there exists no string y such that $M(x, y) = 1$.

2.1 Bilinear Maps

We briefly recall the basics of bilinear maps following [24, 18]. Let BGen be a randomized algorithm that on input a security parameter κ outputs a prime q, such that $\log_2(q) = O(\kappa)$, two cyclic groups of prime order q and a pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.

We call e a bilinear map if the following properties hold:

- Bilinearity: For all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_q$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
- Non-degeneracy: For generator g of $\mathbb{G}$ it holds that $e(g, g) \neq 1$. Since $\mathbb{G}_T$ is of prime order q, this implies that $e(g, g)$ is a generator of $\mathbb{G}_T$.
- Efficiency: e can be computed efficiently in polynomial time in κ.

A bilinear map satisfying the above properties is sometimes called *admissible* bilinear map. We are only interested in admissible bilinear maps and implicitly mean this type of bilinear maps when writing bilinear maps in short. We call BGen a *Bilinear Group Generator* if the algorithm can be computed efficiently in polynomial time in κ and each pairing e generated by BGen is a bilinear map.

While in the above setting the decisional Diffie-Hellman assumption (DDH) does not hold in group $\mathbb{G}$, there is an extension to the setting with bilinear maps.

Definition 1 (DBDH). *The Decision Bilinear Diffie-Hellman assumption (DBDH) states that for every Bilinear Group Generator* BGen *and algorithm* $\mathcal{D}$ *running in time polynomial in security parameter* κ *it holds that*

$$\begin{aligned}\big|\Pr[\mathcal{D}(\bar{\mathbb{G}}, g, h, g^a, g^b, e(h,g)^{ab})] - \Pr[\mathcal{D}(\bar{\mathbb{G}}, g, h, g^a, g^b, R)]\big| \\ \leq \mathsf{negl}(\kappa)\end{aligned}$$

where $\bar{\mathbb{G}} = (q, \mathbb{G}, \mathbb{G}_T, e) \leftarrow_R \mathsf{BGen}(\kappa)$*,* $g, h \in_R \mathbb{G}$*,* $R \in_R \mathbb{G}_T$*, and* $a, b, c \in_R \mathbb{Z}_q$*. The randomness is taken over the random choices of* BGen*, the group elements* g, h*,* R*, the values* a, b, c*, and the random bits of* $\mathcal{D}$*.*

2.2 Hash Functions and Digital Signatures

A *hash function* H is a function that takes as input a string $x \in \{0,1\}^*$ and returns a fixed-length output string $H(x) \in \{0,1\}^{\ell(\kappa)}$ for some polynomial $\ell(\kappa)$. A *signature scheme* $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ over message space $\mathbb{M}$ consists of three probabilistic polynomial-time algorithms. The key generation algorithm KeyGen produces a key pair $(\mathsf{SigK}, \mathsf{VerK})$ on security parameter 1^κ. The signing algorithm Sign takes a signing key SigK and a message $m \in \mathbb{M}$ and produces a signature σ. A signature σ on message m can be verified with respect to the verification key VerK using the verification algorithm Verify. As standard, we require the hash function to satisfy *collision resistance* and the digital signature scheme to provide *consistency* and *existential unforgeability against chosen-message attacks*. Formal definitions of these properties are provided in Appendix D.1 and D.2.

2.3 Anonymous Threshold Identity-Based Encryption

We derive the notion of *Anonymous Threshold Identity-Based Encryption* from [17] as follows:

Definition 2 (TIBE). *An* anonymous threshold identity-based encryption scheme *(TIBE)* TIBE *is associated with the following probabilistic polynomial-time algorithms:*

1. $\mathsf{Setup}(1^\kappa, s, n)$ *takes as input a security parameter* 1^κ*, the number of decryption servers* n *and the security threshold* s*, with* $1 \leq s \leq n$*. It generates system parameters* pk*, a verification key* vk*, and* n *master secret key shares* $\{\mathsf{sk}_i\}_{i\in[n]}$*. The* i*-th decryption server gets master secret key share* sk_i*.*

2. $\mathsf{ShareKeyGen}(\mathsf{pk}, i, \mathsf{sk}_i, \mathsf{id})$ *takes as input the public parameter* pk*, the decryption server index* i*, the corresponding secret key* sk_i *and an identity* $\mathsf{id} \in \{0,1\}^*$*. It generates an identity key share* (i, ik_i)*.*
3. $\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, i, \mathsf{ik}_i)$ *takes as input the public parameter* pk*, the verification key* vk*, an identity* id*, a decryption server index* i *and an identity key share* ik_i*. It outputs* true *or* false*.*
4. $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, \{(i, \mathsf{ik}_i)\}_{i \in \mathcal{S}})$ *takes as input the public parameter* pk*, the verification key* vk*, an identity* id *and indexed identity key shares* ik_i *and returns an identity key* ik *or* $\perp$*.*
5. $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, m)$ *takes as input the public parameter* pk*, an identity* id *and a message* m *and outputs a ciphertext* c*.*
6. $\mathsf{Decrypt}(\mathsf{pk}, \mathsf{id}, \mathsf{ik}, c)$ *takes as input the public parameter* pk*, an identity* id*, an identity key* ik *and a ciphertext* c *and outputs a message* m*.*

We require for all $\kappa, n, s \in \mathbb{N}$*, where* $1 \leq s \leq n$*, and any* $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i \in [n]}) \leftarrow \mathsf{Setup}(1^\kappa, s, n)$ *the following properties:*

- ***Share consistency****: For any identity* $\mathsf{id} \in \{0,1\}^*$ *and any* $i \in [n]$*, if* $(i, \mathsf{ik}_i) \leftarrow \mathsf{ShareKeyGen}(\mathsf{pk}, i, \mathsf{sk}_i, \mathsf{id})$*, then* $\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, i, \mathsf{ik}_i) = \mathsf{true}$*.*
- ***Decryption correctness****: For any identity* $\mathsf{id} \in \{0,1\}^*$*, if* $\mathcal{S}$ *is a subset of* $[n]$ *of size* s*,* $\mathcal{IK} := \{(i, \mathsf{ik}_i) | i \in \mathcal{S} \wedge (i, \mathsf{ik}_i) \leftarrow \mathsf{ShareKeyGen}(\mathsf{pk}, i, \mathsf{sk}_i, \mathsf{id})\}_{i \in \mathcal{S}}$*, and* $\mathsf{ik} \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, \mathcal{IK})$*, then we require that for any* m *in the message space,* $m = \mathsf{Decrypt}(\mathsf{pk}, \mathsf{ik}, \mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, m))$*.*

Security. We define security via three properties: *key generation consistency*, *security against chosen-identity attacks* and *anonymity*. Informally, the first one states that an adversary cannot generate a ciphertext and two sets of valid identity key shares for the same identity such that the shares combine to different keys and the ciphertext is decrypted to two different plaintexts. The last ones state that an adversary cannot distinguish between two encryptions and two identities used for encryption. We formally define the security game and ANON-IND-ID-CPA security in Appendix D.6.

3 Statement-Oblivious Threshold Witness Encryption

In the setting of *threshold witness encryption (TWE)*, we distinguish between users and decryption servers. Users either aim to encrypt some plaintext under a statement x in some NP language $\mathcal{L}$ or aim to decrypt some ciphertext knowing a witness corresponding to the statement $x \in \mathcal{L}$. Decryption servers possess private information and assist users while decrypting a ciphertext. The decryption servers constitute a committee with a fixed number of corrupted parties. The committee may be static or adaptive depending on the concrete instantiation. For instance, a line of work [13, 14, 10] proposed mechanisms to select committees without revealing the identity of the members until they speak to protect against adaptive adversaries. The constructions are based on techniques incorporated in many popular blockchain. We emphasize that our definition and

construction abstracts from the concrete instantiation of the committee. We only assume that a committee consists of n decryption servers and only $s-1$ of them are corrupted. Moreover, we assume the setup procedure of a TWE construction to be executed by a trusted dealer. This approach is standard in threshold cryptography and a trusted dealer could be realized by a tailored multi-party computation protocol. The dealer distributes secret information to the decryption servers and publishes public information to all parties.

In contrast to the definition of extractable witness encryption on blockchain (eWEB) by [8], we abstract away the realization of the committee while their definition explicitly considers a dynamic committee and a hand-off procedure to move from one committee to another. Since the change of the committee members is inherent to their definition, they also consider adaptive corruption in their security game. Moreover, their definition specifically considers a model where plaintexts are shared to the committee members which reveal these information only if a witness is presented. In contrast, our definition follows the approach presented by [9] where only a single secret key is shared between the committee members. In contrast to the definition of blockchain witness encryption (BWE) by Campanelli et al. [10] we do not explicitly define our TWE based notion for blockchains. Here again, we abstract away the concrete realization of the committee.

Formally, we define our new primitive as follows.

Definition 3 (TWE). *A* threshold witness encryption scheme *(TWE)* TWE *for an NP language* $\mathcal{L}$ *with associated relation* R *consists of the following five PPT algorithms:*

1. $\mathsf{Setup}(1^\kappa, s, n)$ *takes as input the security parameter* 1^κ*, a threshold* s*, and the number of decryption servers* n*, where* $1 \leq s \leq n$*. It outputs a triple* $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]})$*, where* pk *is a public key,* vk *is a verification key, and* sk_i *is the secret key share for the decryption server with index* i*.*
2. $\mathsf{Encrypt}(\mathsf{pk}, x, m)$ *takes as input the public key* pk*, a statement* x*, and a message* m*. It outputs a ciphertext* c*.*
3. $\mathsf{ShareDec}(\mathsf{pk}, c, x, w, (i, \mathsf{sk}_i))$ *takes as input a public key* pk*, a ciphertext* c*, a statement* x*, a witness* w*, and the index* i *together with the secret key share* sk_i *of the* i*-th decryption server. It outputs a decryption share* d_i *or a failure symbol* $\perp$ *together with the index* i*.*
4. $\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, x, (i, d_i))$ *takes as input a public key* pk*, a verification key* vk*, a ciphertext* c*, a statement* x*, and an indexed decryption share* (i, d_i)*. It outputs* false *if the decryption share is invalid and* true *if it is valid with respect to* pk*,* vk*,* c*, and* x*.*
5. $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, x, \{(i, d_i)\}_{i\in\mathcal{S}})$ *takes as input a public key* pk*, a verification key* vk*, a ciphertext* c*, a statement* x*, and a set of decryption shares* $\{(i, d_i)\}_{i\in\mathcal{S}}$*. It outputs message* m *or* $\perp$*.*

We require for every security parameter $\kappa \in \mathbb{N}$*, every NP-language* $\mathcal{L}$ *with associated relation* R*, every* $n, s \in \mathbb{N}$ *where* $1 \leq s \leq n$*, every output* $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]})$ *of* $\mathsf{Setup}(1^\kappa, s, n)$*, every* $x \in \mathcal{L}$ *and* w *such that* $(x, w) \in R$*, for every message* m*, and every ciphertext* $c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, x, m)$*:*

– ***Decryption share validity:*** *If* $(i, d_i) \leftarrow \mathsf{ShareDec}(\mathsf{pk}, c, x, w, (i, \mathsf{sk}_i))$*, then* $\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, x, (i, d_i)) = 1$*.*

– ***Correctness:*** *For any* $\mathcal{S} \subseteq [n]$ *of size* s*, if* $\{(i, d_i)\}_{i \in \mathcal{S}}$ *is a set of distinct decryption shares with* $(i, d_i) \leftarrow \mathsf{ShareDec}(\mathsf{pk}, c, x, w, (i, \mathsf{sk}_i))$ *for each* $i \in \mathcal{S}$*, then* $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, x, \{(i, d_i)\}_{i \in \mathcal{S}}) = m$*.*

Security. We define security via three properties: *indistinguishability under chosen-ciphertext attacks* (IND-CCA), *statement obliviousness* (SO) and *decryption consistency under chosen-ciphertext attacks* (DC-CCA). Intuitively, IND-CCA and SO state that ciphertexts created using two different messages and two different statements cannot be distinguished. We combine these property formally in the security game $\mathsf{Exp}^{\mathsf{SO\text{-}CCA}}$. The DC-CCA property states that an adversary cannot produce two sets of valid decryption shares that are combined to two different messages unequal $\perp$. Formally, we define the security game $\mathsf{Exp}^{\mathsf{SO\text{-}DC}}$.

Experiment $\mathsf{Exp}^{\mathsf{SO\text{-}CCA}}_{\mathsf{TWE},\mathcal{A}}(1^\kappa)$

$\mathcal{M} \leftarrow \mathcal{A}_0(1^\kappa)$ with $|\mathcal{M}| < s$

$(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i \in [n]}) \leftarrow \mathsf{Setup}(1^\kappa, s, n)$

$\alpha, \beta \in_R \{0, 1\}$

$(x_0, x_1, m_0, m_1) \leftarrow \mathcal{A}_1^{\mathcal{O}(\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i \in \mathcal{M}})$

$c^* \leftarrow \mathsf{Encrypt}(\mathsf{pk}, x_\alpha, m_\beta)$

$(\alpha', \beta') \leftarrow \mathcal{A}_2^{\mathcal{O}(\cdot,\cdot,\cdot,\cdot)}(c^*)$

return $(\alpha, \beta) = (\alpha', \beta')$

In the given security game, the adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ corrupts the decryption servers in $\mathcal{M}$. $\mathcal{A}_1$ and $\mathcal{A}_2$ can use the oracle $\mathcal{O}(\cdot, \cdot, \cdot, \cdot)$ to make decryption queries. To do so, the adversary sends (i, c, x, w) to $\mathcal{O}$ which returns $(i, d_i) \leftarrow \mathsf{ShareDec}(\mathsf{pk}, c, x, w, (i, \mathsf{sk}_i))$. Only for $\mathcal{A}_2$, the oracle first checks if $c = c^*$, $x \in \{x_0, x_1\}$ and $(x, w) \in R$. If this holds, the oracle returns $(i, \perp)$ and otherwise it returns a correct decryption share.

Experiment $\mathsf{Exp}^{\mathsf{SO\text{-}DC}}_{\mathsf{TWE},\mathcal{A}}(1^\kappa)$

$\mathcal{M} \leftarrow \mathcal{A}_0(1^\kappa)$ with $|\mathcal{M}| < s$

$(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\kappa, s, n)$

$(x, c, \{(i, d_i)\}_{i\in\mathcal{S}}, \{(i, d'_i)\}_{i\in\mathcal{S}'}) \leftarrow \mathcal{A}_1^{\mathcal{O}(\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in\mathcal{M}})$

  where $\mathcal{S}, \mathcal{S}' \subseteq [n] \wedge |\mathcal{S}| = s = |\mathcal{S}'|$

$m \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, x, \{(i, d_i)\}_{i\in\mathcal{S}})$

$m' \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, x, \{(i, d'_i)\}_{i\in\mathcal{S}'})$

if $\forall i \in \mathcal{S} : \mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, x, (i, d_i)) = \mathsf{true}$

  $\wedge \forall i \in \mathcal{S}' : \mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, x, (i, d'_i)) = \mathsf{true}$

  $\wedge \bot \neq m \neq m' \neq \bot$

  return 1

else

  return 0

Here, the adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ corrupts the decryption servers in $\mathcal{M}$ and $\mathcal{A}_1$ can use the decryption oracle $\mathcal{O}(i, c, x, w)$ that returns $(i, d_i) \leftarrow \mathsf{ShareDec}(\mathsf{pk}, c, x, w, (i, \mathsf{sk}_i))$.

Definition 4 (SO-IND-CCA Security of TWE). *A threshold witness encryption scheme* TWE *is* statement-oblivious and message-indistinuishable under chosen-ciphertext attacks *(SO-IND-CCA) secure if for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$*, there exist negligible functions* negl_0 *and* negl_1 *such that*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{SO\text{-}CCA}}_{\mathsf{TWE},\mathcal{A}}(1^\kappa) = 1] - \frac{1}{4}\right| \leq \mathsf{negl}_0(\kappa) \quad \wedge$$

$$\Pr[\mathsf{Exp}^{\mathsf{SO\text{-}DC}}_{\mathsf{TWE},\mathcal{A}}(1^\kappa) = 1] \leq \mathsf{negl}_1(\kappa).$$

Remark 1 The standard notion of witness encryption (cf. [1]) defines security without access to a decryption oracle. This is due to the fact that decryption in the standard notion can be attempted by any party locally using knowledge of the witness. In the threshold setting, decryption is performed via an interaction with a decryption committee that performs decryption in a distributed way using a secret shared trapdoor. Hence, we have to give the adversary access to a decryption oracle.

Remark 2 We note that in the context of TWE SO-IND-CCA security implies *extractability*, an additional security requirement often required from witness encryption. We provide further details to the notion of extractability for TWE and a reduction from extractability to SO-IND-CCA security in Appendix A.

4 Oblivious Threshold Tag-Based Encryption

In this section, we present the notion of *oblivious threshold tag-based encryption* (O-TTBE) which constitutes an extension of standard threshold tag-based

encryption as presented in [15]. Intuitively, a threshold tag-based encryption scheme is *oblivious* if a ciphertext hides the tag it was created with. We first state the definition of threshold tag-based encryption and present the obliviousness property as part of the security guarantees afterwards.

Definition 5 (TTBE). *A* threshold tag-based encryption scheme *(TTBE)* TTBE *consists of the following five PPT algorithms:*

1. $\mathsf{Setup}(1^\kappa, s, n)$ *takes as input the security parameter* 1^κ*, a threshold* s*, and the number of decryption servers* n *where* $1 \leq s \leq n$*. It outputs a triple* $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]})$*, where* pk *is a public key,* vk *is a verification key, and* sk_i *is the secret key share for the* i*-th decryption server.*
2. $\mathsf{Encrypt}(\mathsf{pk}, t, m)$ *takes as input a public key* pk*, a tag* t*, and a message* m*, and it outputs a ciphertext* c*.*
3. $\mathsf{ShareDec}(\mathsf{pk}, c, t, (i, \mathsf{sk}_i))$ *takes as input a public key* pk*, a ciphertext* c*, a tag* t*, and the index* i *together with the secret key share* sk_i *of the decryption server with index* i*. It outputs a decryption share* d_i *or a failure symbol* $\perp$ *together with the index* i*.*
4. $\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, t, (i, d_i))$ *takes as input a public key* pk*, a verification key* vk*, a tag* t*, and an indexed decryption share* (i, d_i)*. It outputs* false *if the decryption share is invalid and* true *if it is valid with respect to* pk*,* vk*,* c *and* t*.*
5. $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, t, \{(i, d_i)\}_{i\in\mathcal{S}})$ *takes as input a public key* pk*, a verification key* vk*, a ciphertext* c*, a tag* t*, and a set of decryption shares* $\{(i, d_i)\}_{i\in\mathcal{S}}$*. It outputs message* m *or* $\perp$*.*

We require for every security parameter $\kappa \in \mathbb{N}$*, every committee parameters* $n, s \in \mathbb{N}$ *where* $1 \leq s \leq n$*, every* $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]})$ *generated by* $\mathsf{Setup}(1^\kappa, s, n)$*, every message* m*, every tag* t *and every* $c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, t, m)$*:*

- ***Decryption share validity:*** *If* $(d_i, i) \leftarrow \mathsf{ShareDec}(\mathsf{pk}, c, t, (i, \mathsf{sk}_i))$*, then* $\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, t, (i, d_i)) = 1$*.*
- ***Correctness:*** *If* $\{(i, d_i)\}_{i\in\mathcal{S}}$ *is a set of* s *distinct decryption shares with* $(i, d_i) \leftarrow \mathsf{ShareDec}(\mathsf{pk}, c, t, (i, \mathsf{sk}_i))$ *for each* $i \in \mathcal{S}$*, then* $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, t, \{(i, d_i)\}_{i\in\mathcal{S}}) = m$*.*

Security. Security of a TTBE scheme is defined via two properties: *oblivious indistinguishable messages under chosen-ciphertext attacks* (IND-CCA) and *decryption consistency under chosen-ciphertext attacks* (DC-CCA). The intuition for these properties is analog to the ones of threshold witness encryption. The IND-CCA property states that ciphertexts created using two different messages and two different tags cannot be distinguished. The DC-CCA property states that an adversary cannot produce two sets of valid decryption shares that are combined to two different messages unequal $\perp$. To formalize these properties, we design the following security games:

Experiment $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE},\mathcal{A}}(\kappa)$

$\mathcal{M} \leftarrow \mathcal{A}_0(1^\kappa)$ with $|\mathcal{M}| < s$

$\alpha, \beta \in_R \{0,1\}$

$(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\kappa, s, n)$

$(t_0, t_1, m_0, m_1) \leftarrow \mathcal{A}_1^{\mathcal{O}(\cdot,\cdot,\cdot)}(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in\mathcal{M}})$

$c^* \leftarrow \mathsf{Encrypt}(pk, t_\alpha, m_\beta)$

$(\alpha', \beta') \leftarrow \mathcal{A}_2^{\mathcal{O}(\cdot,\cdot,\cdot)}(c^*)$

return $(\alpha, \beta) = (\alpha', \beta')$

The decryption oracle $\mathcal{O}(\cdot,\cdot,\cdot)$ takes as parameter an index i, a ciphertext c and a tag t, and computes $(i, d_i) \leftarrow \mathsf{ShareDec}(\mathsf{pk}, c, t, (i, \mathsf{sk}_i))$. If $(c, t) \in \{(c^*, t_0), (c^*, t_1)\}$ it returns $(i, \perp)$, otherwise it returns (i, d_i).

Experiment $\mathsf{Exp}^{\mathsf{O\text{-}DC}}_{\mathsf{TTBE},\mathcal{A}}(\kappa)$

$\mathcal{M} \leftarrow \mathcal{A}_0(1^\kappa)$ with $|\mathcal{M}| < s$

$(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\kappa, s, n)$

$(t, c, \{(i, d_i)\}_{i\in\mathcal{S}}, \{(i, d'_i)\}_{i\in\mathcal{S}'}) \leftarrow \mathcal{A}_1^{\mathcal{O}(\cdot,\cdot,\cdot)}(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in\mathcal{M}})$

  where $\mathcal{S}, \mathcal{S}' \subseteq [n] \wedge |\mathcal{S}| = s = |\mathcal{S}'|$

$m \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, t, \{(i, d_i)\}_{i\in\mathcal{S}})$

$m' \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, t, \{(i, d'_i)\}_{i\in\mathcal{S}'})$

if $\forall i \in \mathcal{S} : \mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, t, (i, d_i)) = \mathsf{true}$

  $\wedge\, \forall i \in \mathcal{S}' : \mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, t, (i, d'_i)) = \mathsf{true}$

  $\wedge\, \perp \neq m \neq m' \neq \perp$

  return 1

else

  return 0

The decryption oracle $\mathcal{O}(\cdot,\cdot,\cdot)$ takes as parameter an index i, a ciphertext c and a tag t , and returns $\mathsf{ShareDec}(\mathsf{pk}, c, t, (i, \mathsf{sk}_i))$.

Definition 6. *A TTBE scheme* TTBE *is OB-IND-CCA secure if for every PPT adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$*, there exists negligible functions* negl_0 *and* negl_1 *such that*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE},\mathcal{A}}(\kappa) = 1] - \frac{1}{4}\right| \leq \mathsf{negl}_0(\kappa) \quad \wedge$$
$$\Pr[\mathsf{Exp}^{\mathsf{O\text{-}DC}}_{\mathsf{OTTBE},\mathcal{A}}(\kappa) = 1] \leq \mathsf{negl}_1(\kappa).$$

We use the notation of *oblivious TTBE* in short for referring to an OB-IND-CCA secure TTBE.

5 Constructing Statement-Oblivious TWE

In this section, we present a construction for statement-oblivious threshold witness encryption (SO-TWE) from oblivious threshold tag-based encryption (O-TTBE).

Construction 1: $\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}}$

Public parameters:
The scheme is defined for a language $\mathcal{L}$ with relation R. The number of committee members is denoted by n and the threshold parameter is s. We make use of a oblivious tag-based encryption scheme OTTBE and a collision-resistant hash function $H : \mathbb{X} \to \mathbb{T}$, where $\mathbb{X}$ is the statement space of language $\mathcal{L}$ and $\mathbb{T}$ is the tag space of OTTBE.

$\mathsf{Setup}(1^\kappa, s, n)$**:**
Output $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]}) := \mathsf{OTTBE.Setup}(1^\kappa, s, n)$.

$\mathsf{Encrypt}(\mathsf{pk}, x, m)$**:**
Output $c := \mathsf{OTTBE.Encrypt}(\mathsf{pk}, H(x), m)$.

$\mathsf{ShareDec}(\mathsf{pk}, c, x, w, (i, \mathsf{sk}_i))$**:**
If $(x, w) \in R$, output $\mathsf{OTTBE.ShareDec}(\mathsf{pk}, c, H(x), (i, \mathsf{sk}_i))$. Otherwise, output $(i, \perp)$.

$\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, x, (i, d_i))$**:**
If $d_i = \perp$, output false. Otherwise output $\mathsf{OTTBE.ShareVf}(\mathsf{pk}, \mathsf{vk}, c, H(x), (i, d_i))$.

$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, x, \{(i, d_i)\}_{i\in\mathcal{S}})$**:**
Output $\mathsf{OTTBE.Combine}(\mathsf{pk}, \mathsf{vk}, c, H(x), \{(i, d_i)\}_{i\in\mathcal{S}})$.

Theorem 1. *Let* OTTBE *be a* threshold tag-based encryption scheme *that is* OB-IND-CCA *secure and* H *be a collision-resistant hash function. Then, the scheme* $\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}}$ *is a* SO-IND-CCA *secure* threshold witness encryption *scheme.*

The security proof is presented in Appendix B.

Confidential witnesses and decryptions. We can add the support of confidential witnesses and decryptions to our construction by applying techniques from [8]. To ensure confidentiality of witnesses, clients send non-interactive zero-knowledge proofs of knowledge of the witness to the decryption severs. Then, as part of the decryption algorithm the servers check the validity of the proof against the submitted statement, instead of checking the witness relation directly. To achieve confidentiality of decryptions, the decryption servers encrypt decryption shares under the public key of the client as part of the decryption algorithm. We ensure that decryption requests cannot be replayed with different public keys by applying the witness confidentiality approach and labeling the zero-knowledge proof with the submitted public key.

6 O-TTBE from Bilinear Mappings and Random Oracles

In this section, we present the construction of a concretely efficient oblivious threshold tag-based encryption scheme. Our construction is based on bilinear maps and random oracles and its security relies on the Decision Bilinear Diffie-Hellman assumption (cf. Section 2.1). Before we present the formal specification of the construction, we give an intuition about the challenges of designing an O-TTBE scheme and how they are addressed by our construction.

Common approaches towards CCA security in the threshold setting incorporate ciphertext validation before decryption [19, 18, 20, 21, 22]. The validation ensures that each decryption request is either declined or yields exactly the original plaintext created by some client. This feature is the common way to prevent the adversary from executing *ciphertext-reuse*. Under this term, we understand reusing and potentially adapting ciphertext components in maliciously created ciphertext with the goal to extract decryptions for valid ciphertexts from the decryptions of maliciously created ones.

In an *oblivious threshold* scheme, declining decryptions is not possible since a single decryption server must not detect if the provided tag is valid. This is due to the fact that some servers can get corrupted in the threshold setting. If a single server was able to check the validity of a tag, the adversary would be able to exploit corrupted servers to break obliviousness. It follows that we have to apply a less strict ciphertext validation allowing for multiple decryptions, with different tags, of the same ciphertext. However, decryptions with invalid tags must not leak any information about the encrypted plaintext or the tag used for encryption. Consequently, we cannot follow the approaches of previous work.

Instead, we have to take one step back and address the challenge of achieving CCA security independent of previous work. It turns out that the discussed ciphertext validation is necessary but not sufficient to prove CCA security. In particular, when constructing a CCA secure encryption scheme it is not sufficient to take a CPA secure scheme and add a zero-knowledge proof of correct encryption to the ciphertexts. Proving security via a reduction to a number theoretic assumption is typically done by building a simulator that uses a concrete adversary on the scheme to break the underlying assumption. Even if a ciphertext is proven to be created correctly, the simulator needs to be capable of answering decryption queries of the adversary without actually knowing the secret key. This challenge is typically addressed by incorporating an additional trapdoor into the construction. It follows that for achieving CCA security we need both, (i) a way to prevent ciphertext reuse and (ii) a trapdoor to enable the simulator to answer decryption queries. In addition, for tag obliviousness, we have to achieve the former while (iii) still allowing multiple decryptions, with different tags, for the same ciphertext.

We propose a new construction deploying a single extension together with a simple zero-knowledge proof of correct encryption to a standard threshold variant of CPA secure Hash-ElGamal. The extension provides both, (iii) tag obliviousness and (ii) a trapdoor for decryption, such that a simple zero-knwoledge proof

of correct encryption is sufficient to decline invalid ciphertexts, and hence, (i) prevent ciphertext reuse.

We start by briefly recalling Hash-ElGamal. In Hash-ElGamal, the encryption of a message m samples a random exponent a and outputs two elements $A = g^a$ and $M = m \oplus H(X^a)$ for a group generator g, a random oracle H, and a public key $X = g^x$. In the threshold setting, the secret key x is secret shared among the decryption servers. The decryption shares of the servers are calculated as $d_i := A^{x_i}$, where x_i is the share of the i-th server.

Our extension is to apply a random offset T to A for both, encryption and decryption, using a bilinear map e. This offset is unique for each ciphertext-tag pair. Precisely, we compute $M := H(e(T, X^a)) \oplus m$ for encryption and $d_i := e(T, A^{x_i})$ for decryption. As our notion requires each encrypting party and decryption server to be capable of generating the offset independently, we generate the offset using a random oracle. In particular, we compute the offset T by applying the random oracle to the tag t and the ciphertext component A, i.e., we compute $T = H_2(t, A)$. Finally, we add a non-interactive Schnorr zero-knowledge argument of knowledge of a [25] to the ciphertext, which we bind to the ciphertext component M. The binding is done by incorporating M into the generation of the challenge in the Fiat-Shamir transformation [26]. In addition, we add a Chaum-Pedersen zero-knowledge argument [27] of correct decryption to decryption shares.

The random offset T adds a random exponent $\log_g(T)$ to decryptions with invalid tags, and hence, ensures that invalid decryptions do not give any information about the encrypted message (cf. (iii)). Further it provides a backdoor that can be exploited by the simulator to answer decryption queries without knowledge of x_i (cf. (ii)). In particular, the simulator can simulate the random oracle such that the simulator learns $k = \log_g(T)$ for each T generated by H_2. This way, the simulator can calculate the combined decryption shares $D = e(X^k, A) = e(T, A^x)$ which can again be used to interpolate decryption shares of individual parties. Finally, the zero-knowledge argument of knowledge of a ensures that a component A cannot be re-used for different ciphertexts, and hence, prevents ciphertext reuse (cf. (i)). Further, the zero-knowledge argument of correct decryption ensures that malicious servers cannot trick honest clients into excepting incorrect decryptions.

Our construction yields a concretely efficient scheme. The ciphertexts consist of a bitstring with length equal to the message length, a group element of the bilinear mapping's base group and two exponents (in $\mathbb{Z}_q$, where q is the bilinear group's order). Decryption shares consist of one group element in the mapping's target group and two exponents. Encryption requires a single evaluation and decryption three evaluations of the bilinear map.

We continue by presenting the concrete construction:

Construction 2: $\mathsf{TTBE}_{\mathsf{pROM}}$

Public parameters:
The scheme is defined over a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with groups $\mathbb{G}$ and $\mathbb{G}_T$ where each group is of order q. The number of committee members is denoted by n and the threshold parameter is s. The message and tag length is defined as l. We make use of random oracles $H_1 : \mathbb{G}_T \to \{0,1\}^l$, $H_2 : \{0,1\}^l \times \mathbb{G} \to \mathbb{G}$, $H_3 : \{0,1\}^l \times \mathbb{G}^2 \to \mathbb{Z}_q$, $H_4 : \mathbb{G}^3 \to \mathbb{Z}_q$.

$\mathsf{Setup}(1^\kappa, s, n)$**:**
Sample a generator, g, of $\mathbb{G}$, a secret key $x \in_R \mathbb{Z}_q$ and a sharing polynomial F of degree $s-1$ over $\mathbb{Z}_q$ such that $F(0) = x$. Set $\mathsf{pk} = (g, X := g^x)$, $\mathsf{vk} := \{g^{F(i)}\}_{i\in[n]}$ and $\mathsf{sk}_i := F(i) = x_i$ for each $i \in [n]$. Output $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]})$.

$\mathsf{Encrypt}(\mathsf{pk}, t, m)$**:**
Sample $a, r \in_R \mathbb{Z}_q$ and calculate:

$$A := g^a,\ T = H_2(t, A),\ \widetilde{M} := e(T, X^a),\ M := H_1(\widetilde{M}) \oplus m$$
$$U := g^r,\ w = H_3(M, A, U),\ f = r + aw,\ \pi := (w, f)$$

Return $c = (M, A, \pi)$.
Note that π constitutes a zero knowledge argument of knowledge of $\log_g(A)$.

$\mathsf{ValidateCT}(c)$**:**
Parse $c = (M, A, \pi = (w, f))$ and return true iff

$$w = H_3(M, A, U) \text{ for } U = \frac{g^f}{A^w}.$$

$\mathsf{ShareDec}(pk, c, t', (i, sk_i))$**:**
If $\mathsf{ValidateCT}(c) = \mathsf{false}$ return $(i, \perp)$. Otherwise choose $r_i \in_R \mathbb{Z}_q$ and compute,

$$T' := H_2(t', A), D_i := e(T', A^{x_i})$$
$$U_i := e(T', A^{r_i}), V_i := e(T', g^{r_i})$$
$$w_i := H_4(D_i, U_i, V_i), f_i := r_i + x_i \cdot w_i$$
$$\pi_i := (w_i, f_i)$$

and return $d_i := (i, D_i, \pi_i)$.
Note that π_i constitutes a zero knowledge argument that $(e(T', A), e(T', \mathsf{vk}_i), D_i)$ is a Diffie-Hellmann triple.

$\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}_i, c, t', d_i)$**:**
Parse $d_i = (i, D_i, \pi_i = (w_i, f_i))$, $c = (\cdot, A, \cdot)$, calculate $T' := H_2(t', A)$ and return true iff

$$w_i = H_4(D_i, U_i, V_i) \text{ for } U_i = \frac{e(T', A)^{f_i}}{D_i^{w_i}}, V_i = \frac{e(T', g)^{f_i}}{e(T', \mathsf{vk}_i)^{w_i}}.$$

$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, t', \{(d_i)\}_{i\in\mathcal{S}})$**:**
Return $m = M \oplus H_1(\prod_{i\in\mathcal{S}} (D_i)^{\lambda^{\mathcal{S}}_{0,i}})$.

Correctness of the scheme can be shown as follows:

$$\begin{aligned} m &= M \oplus H_1(\prod_{i\in\mathcal{S}}(D_i)^{\lambda^{\mathcal{S}}_{0,i}}) = M \oplus H_1(\prod_{i\in\mathcal{S}}(e(T', A^{x_i}))^{\lambda^{\mathcal{S}}_{0,i}}) \\ &= M \oplus H_1(\prod_{i\in\mathcal{S}} e(T', A)^{x_i\cdot\lambda^{\mathcal{S}}_{0,i}}) = M \oplus H_1(e(T', A)^{x}) \\ &= m \oplus H_1(e(T, g^{ax})) \oplus H_1(e(T', g^{ax})) = m, \end{aligned}$$

where $t = t'$ yields $H_2(t, A) = T = T' = H_2(t', A)$).

For security, we state the following theorem:

Theorem 2. *Let* BGen *be a Bilinear Group Generator,* $(e, \mathbb{G}, \mathbb{G}_T, q) \leftarrow_R \mathsf{BGen}(\kappa)$ *be a bilinear group in which the Decisional Bilinear Diffie-Hellman (DBDH) assumption holds, and* H_1, H_2, H_3, H_4 *be programmable random oracles. Then, the scheme* $\mathsf{TTBE}_{\mathsf{pROM}}$ *is an* OB-IND-CCA *secure oblivious threshold tag-based encryption scheme.*

We will provide an intuition of our proof for indistinguishable messages under chosen-ciphertext attacks, here, and defer the formal security proof for both indistinguishable messages under chosen-ciphertext attacks (defined via $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE}_{\mathsf{pROM}},\mathcal{A}}$) and decryption consistency under chosen-ciphertext attacks (defined via $\mathsf{Exp}^{\mathsf{O\text{-}DC}}_{\mathsf{TTBE}_{\mathsf{pROM}},\mathcal{A}}$) to Appendix C.

Proof intuition. We prove indistinguishable messages via a reduction to the DBDH assumption. Hence, we build a distinguisher $\mathcal{D}$ that receives a tuple $(\bar{g}, \bar{h}, \alpha = \bar{g}^x, \beta = \bar{g}^y, \gamma)$ and decides if the received tuple is a DBDH tuple, i.e., if $\gamma = e(\bar{h}, \bar{g})^{xy}$. $\mathcal{D}$ has access to an adversary $\mathcal{A}$ on the experiment $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE}_{\mathsf{pROM}},\mathcal{A}}$.

The reduction is based on the observation that, in order to win in $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE}_{\mathsf{pROM}},\mathcal{A}}$, adversary $\mathcal{A}$ when receiving a challenge ciphertext $(M^*, A^*, \cdot)$ has to query H_1 at either $P_0 = e(H_2(t_0, A^*), A^*)^x$ or $P_1 = e(H_2(t_1, A^*), A^*)^x$. If $\mathcal{D}$ defines public parameters $g = \bar{g}$ and $\mathsf{pk} = X = \alpha$ and challenge ciphertext components $H_2(t_{1-b}, A^*) \leftarrow \bar{h}$ (via programming of the random oracle) and $A^* = \beta$ for some $b \in_R \{0, 1\}$, it follows that $P_{1-b} = \gamma$ iff the received tuple is a DBDH tuple. Hence, we can distinguish DBDH tuples from random tuples based on the event that γ has been queried by $\mathcal{A}$. However, setting $M^* = m_{b_m} \oplus H_1(\gamma)$ for $b_m \in_R \{0, 1\}$ does not yield a valid ciphertext if the tuple is no DBDH tuple, a fact that makes the reduction distinguishable from a real experiment. While there are techniques to deal with this problem (cf. [19]), this distinguishability makes the argumentation more long-winded. Instead, we make use of the fact that we are in the tag-based setting, i.e., there are two possible keys at which H_1 can be queried to decide which tag or message has been used for encryption. In particular, we create M^* such that it is a correct encryption of m_{b_m} under tag t_b, i.e., $M^* = m \oplus H_1(P_b)$. At the same time, we program H_2 such that $P_{1-b} = \gamma$ iff the received tuple is a DBDH tuple, i.e., by programming $H_2(t_{1-b}, A^*) \leftarrow \bar{h}$. Hence, we can distinguish based on the event that γ has been queried while still creating a valid ciphertext.

The next question is how to actually compute P_b without knowing x nor $y = \log_g(A^*)$. Here we make use of the fact that $\mathcal{D}$ simulates the random oracle H_2 that is used to compute the tag-dependent offset. In particular, whenever the random oracle H_2 is supposed to sample a random value in $\mathbb{G}$, it samples a random exponent $k \in_R \mathbb{Z}_q$ instead and returns $\bar{g}^k$. The output is still uniformly random distributed in $\mathbb{G}$ but $\mathcal{D}$ learns the discrete logarithm of every value sampled by H_2. This way, $\mathcal{D}$ can restore $k = \log_g(H_2(t_b, A^*))$ and compute $P_b = e(\alpha, \beta^k) = e(H_2(t_b, A^*), A^*)^x$.

As explained above, the major challenge is to answer decryption queries without having access to the private key x. However, this problem can be solved the same way as computing P_b. In particular, $\mathcal{D}$ answers decryption queries for ciphertext $c = (\cdot, A, \cdot)$ and tag t by restoring $k = \log_g(H_2(t, A))$ and computing $e(\beta, A^k) = e(H_2(t, A), A^x)$. The only keys for which the restoring of the exponent k does not work are $(t_{1-b}, A^*) = (t_{1-b}, \beta)$ for which $\mathcal{D}$ programmed the random oracle to $\bar{h}$ without knowing $\log_g(\bar{h})$. However, in consistency with the original security game, $\mathcal{D}$ declines decryptions for (c, t) if $(c, t) \in \{(c^*, t_0), (c^*, t_1)\}$. Hence, $\mathcal{D}$ only fails to answer decryption queries if $\mathcal{A}$ sends a valid ciphertext $c \neq c^*$ such that $c = (\cdot, A^*, \pi)$. However, to do so, the adversary needs to be capable of generating a valid zero-knowledge argument π of knowledge of $y = \log_g(A^*)$ without actually knowing y. In fact, not even $\mathcal{D}$ has knowledge of y. The probability is negligible for a computationally bounded adversary to find such a proof. It follows that $\mathcal{D}$ is capable of answering all decryption queries, expect with negligible probability.

7 Oblivious TTBE from Anonymous TIBE

This section presents a general transformation from an anonymous threshold identity-based encryption scheme, a one-time signature scheme and a collision-resistant hash functions to an oblivious threshold tag-based encryption scheme. The scheme depicts an extension of [18].

Construction 3: $\mathsf{TTBE}_{\mathsf{IBE}}$

Public parameters:
The number of committee members is denoted by n and the threshold parameter is s. We make use of a one-time signature scheme OTS, an anonymous threshold identity-based encryption scheme TIBE, and a collision-resistant hash function $H : \mathbb{T} \times \mathbb{K} \to \mathbb{I}$, where $\mathbb{T}$, $\mathbb{K}$, $\mathbb{I}$ is the tag space, the verification key space of OTS, and the identity space of TIBE.

$\mathsf{Setup}(1^\kappa, s, n)$**:**
Run $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{TIBE.Setup}(1^\kappa, s, n)$ and output the keys $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]})$.

$\mathsf{Encrypt}(\mathsf{pk}, t, m)$**:**

Generate a signature key pair $(\mathsf{SigK}, \mathsf{VerK}) \leftarrow \mathsf{OTS.KeyGen}(1^\kappa)$, calculate $\mathsf{id} := H(t, \mathsf{VerK})$, $c_0 := \mathsf{TIBE.Encrypt}(\mathsf{pk}, \mathsf{id}, m)$, and $\sigma := \mathsf{OTS.Sign}(\mathsf{SigK}, c_0)$. Output $c := (c_0, \mathsf{VerK}, \sigma)$.

$\mathsf{ShareDec}(\mathsf{pk}, c, t, (i, \mathsf{sk}_i))$**:**
Parse c to $(c_0, \mathsf{VerK}, \sigma)$ and check that $\mathsf{OTS.Verify}(\mathsf{VerK}, \sigma, c_0) = \mathsf{true}$. If the check fails, output $(i, \perp)$. Otherwise, calculate $\mathsf{id} := H(t, \mathsf{VerK})$ and output an identity key share $(i, \mathsf{ik}_i) \leftarrow \mathsf{TIBE.ShareKeyGen}(\mathsf{pk}, i, \mathsf{sk}_i, \mathsf{id})$ as d_i.

$\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, t, (i, d_i))$**:**
Parse c to $(c_0, \mathsf{VerK}, \sigma)$ and output true iff $\mathsf{OTS.Verify}(\mathsf{VerK}, \sigma, c_0) = \mathsf{true}$ and $\mathsf{TIBE.ShareVf}(\mathsf{pk}, \mathsf{vk}, H(t, \mathsf{VerK}), i, d_i) = \mathsf{true}$.

$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, t, \{(i, d_i)\}_{i \in \mathcal{S}})$**:**
Parse c to $(c_0, \mathsf{VerK}, \sigma)$, calculate $\mathsf{id} = H(t, \mathsf{VerK})$ and $\mathsf{ik} := \mathsf{TIBE.Combine}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, \{(d_i)\}_{i \in \mathcal{S}})$. If $\mathsf{ik} = \perp$, output $\perp$. Otherwise, output $m := \mathsf{TIBE.Decrypt}(\mathsf{pk}, \mathsf{id}, \mathsf{ik}, c_0)$.

Correctness of the scheme is easy to see. If the same tag is used for decryption and encryption, the encryption contains a ciphertext under the same identity for which the decryption algorithm creates the identity key. Next, we show security.

Theorem 3. *Let* TIBE *be an anonymous threshold identity-based encryption scheme that is* ANON-IND-ID-CPA *secure, H be collision-resistant hash function, and* OTS *an existentially unforgeable one-time signature scheme. Then, the scheme* $\mathsf{TTBE}_{\mathsf{IBE}}$ *is a* OB-IND-CCA *secure threshold tag-based encryption scheme.*

The security proof is presented in Appendix F.

8 Constructing Anonymous TIBE

In this section, we construct an anonymous threshold identity-based encryption scheme (TIBE) from an anonymous non-threshold verifiable identity-based encryption scheme (VIBE) and an homomorphic secret sharing scheme (HSS) with linear decoding. A VIBE extends the definition of an identity-based encryption scheme with a verification algorithm that allows to check if an identity key was generated correctly. An HSS scheme allows to secret share some value and to perform operations on the shares such that the result of the combination yields the output of a function applied directly to the value. We state the definitions for VIBE and HSS in Appendix D.4 and D.5 respectively. The HSS scheme is used to execute the Extract algorithm of the VIBE scheme in a distributed way. The operations that need to be supported by the HSS scheme depend on the concrete VIBE scheme, i.e., how the output shares of its Extract algorithm can be computed. While we use the HSS scheme in a black-box way, it is an interesting open question to provide concrete instantiations of the following black-box transformation. In Appendix H we discuss potential pathways to obtain an anonymous

threshold IBE scheme from the concrete anonymous IBE scheme by Boyen and Waters [16].

Construction 4: Anonymous TIBE

Public parameters:
The number of committee members is denoted by n and the threshold parameter is s. This construction uses an ANON-IND-ID-CPA secure VIBE scheme $\mathsf{VIBE} = (\mathsf{VIBE.Setup}, \mathsf{VIBE.Extract}, \mathsf{VIBE.Verify}, \mathsf{VIBE.Encrypt}, \mathsf{VIBE.Decrypt})$ and a linear decoding HSS scheme $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval}, \mathsf{HSS.Dec})$ for the function $y := (\mathsf{ik}_z, \rho_z) \leftarrow \mathsf{VIBE.Extract}(\mathsf{pk}, x, z)$ with public input $z = \mathsf{id}$ and shared private input $x = \mathsf{msk}$, as building blocks.

$\mathsf{Setup}(1^\kappa, s, n)$:
- $(\mathsf{pk}_{\mathsf{VIBE}}, \mathsf{vk}_{\mathsf{VIBE}}, \mathsf{msk}) \leftarrow \mathsf{VIBE.Setup}(1^\kappa)$
- $(\mathsf{msk}_1, \ldots, \mathsf{msk}_n) \leftarrow \mathsf{HSS.Share}(1^\kappa, \mathsf{msk})$
- **return** $(\mathsf{pk}, \mathsf{vk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)) := (\mathsf{pk}_{\mathsf{VIBE}}, \mathsf{vk}_{\mathsf{VIBE}}, (\mathsf{msk}_1, \ldots, \mathsf{msk}_n))$

$\mathsf{ShareKeyGen}(\mathsf{pk}, i, \mathsf{sk}_i, \mathsf{id})$:
- **return** (i, ik_i), where $\mathsf{ik}_i := y_i \leftarrow \mathsf{HSS.Eval}(i, \mathsf{id}, \mathsf{sk}_i)$

$\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, i, \mathsf{ik}_i)$:
- **return** true

$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, \{(i, \mathsf{ik}_i)\}_{i \in \mathcal{S}})$:
- $y \leftarrow \mathsf{HSS.Dec}(\{\mathsf{ik}_i\}_{i \in \mathcal{S}})$
- Parse $y := (\mathsf{ik}, \rho)$
- **if** $\mathsf{VIBE.Verify}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, \mathsf{ik}, \rho) = 1$ **return** ik
- **else return** $\perp$

$\mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, m)$:
- **return** $\mathsf{VIBE.Encrypt}(\mathsf{pk}, \mathsf{id}, m)$

$\mathsf{Decrypt}(\mathsf{pk}, \mathsf{id}, \mathsf{ik}, c)$:
- **return** $\mathsf{VIBE.Decrypt}(\mathsf{pk}, \mathsf{ik}, c)$

We first show that our construction satisfies the correctness properties, in particular share consistency and decryption correctness. Then, we prove the security property, ANON-IND-ID-CPA security.

The share consistency property states that for all correctly generated identity key shares, the ShareVf algorithm outputs true. Since the ShareVf algorithm of our construction always outputs true, the property is apparently satisfied. Decryption correctness is easy to see as well. Let, for any $\kappa, n \in \mathbb{N}$ and $1 \leq s \leq n$, $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i \in [n]}) \leftarrow \mathsf{Setup}(1^\kappa, s, n)$. Note that $\mathsf{sk}_i := \mathsf{msk}_i$ where msk_i is the i-th share obtained using $\mathsf{HSS.Share}(1^\kappa, \mathsf{msk})$ for a master secret key of the non-threshold VIBE scheme VIBE. Then, for any id, $\mathsf{ShareKeyGen}(\mathsf{pk}, i, \mathsf{sk}_i, \mathsf{id})$ returns an output share of $\mathsf{HSS.Eval}(i, \mathsf{id}, \mathsf{msk}_i)$ which equals a share of $\mathsf{VIBE.Extract}(\mathsf{pk}, \mathsf{msk}, \mathsf{id})$. Given any set of s identity key shares, the Combine algorithm first decodes the shares to (ik, ρ) and outputs ik if

$\mathsf{VIBE.Verify}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, \mathsf{ik}, \rho) = 1$. Due to the correctness property of the (s, n)-HSS scheme, (ik, ρ) is exactly the output of $\mathsf{VIBE.Extract}(\mathsf{pk}, \mathsf{msk}, \mathsf{id})$. Now, due to the correctness property of VIBE, it follows that $\mathsf{Decrypt}(\mathsf{pk}, \mathsf{ik}, \mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, m)) = m$ holds for any message m.

Finally, we show that the scheme TIBE is ANON-IND-ID-CPA secure. Formally, we state the following theorem.

Theorem 4. *Let* VIBE *be an ANON-IND-ID-CPA secure VIBE scheme satisfying soundness and let* HSS *be a linear decoding (s, n)-HSS scheme satisfying correctness and computational security. Then,* TIBE *defined in Construction 4 is an ANON-IND-ID-CPA secure (n, s)-TIBE scheme.*

The security proof is presented in Appendix G.

9 Applications

Statement-oblivious threshold witness encryption (SO-TWE) is interesting whenever use-cases of classical witness encryption, e.g., the ones presented in [1, 8], should be extended by an additional privacy property, i.e., if the statement used for encryption is required to stay hidden until the decryption is successful. A straightforward example is witness encryption based time-lock encryption, as proposed by [6], with a hidden release time. In simplified settings, in which the decryption servers have access to public data (e.g., timestamps), our intermediate notion of oblivious threshold tag-based encryption (O-TTBE) is often sufficient. However, in more sophisticated scenarios, e.g., if the decryption servers need to rely on external authorities to provide authenticated public data, it is necessary to use SO-TWE. We present use-cases and briefly explain how they can be realized using our primitives; one of the use-case is provided in this section and others are deferred to Appendix J. The use-cases are partial extensions to the ones presented by [8] for their notion of eWEB.

Price-dependent transaction execution with hidden price. Imaging a user that wants to buy some asset at a Decentralized Finance (DeFi) trading platform once the share price reaches a certain value. Since the user does not know when this event happens, it does not want to stay online all the time. The user's goal is to keep the transaction and the desired share price private until the price hits the intended value. Privacy is an important aspect in this scenario, since revealing information, e.g., the intended purchase price, could lead to financial disadvantage, e.g., due to insider trading. In the DeFi space, oracle services are widely deployed and commonly used. These services provide signed information about real-world data such as share prices. However, achieving the user's goal requires additional techniques. To support the described scenario, the user can exploit SO-TWE.

In more detail, suppose there is a committee holding the secret key shares of a SO-TWE scheme with public key pk for language $\mathcal{L}$ with associated relation R. $\mathcal{L}$ is defined such that a statement x specifies the intended share price as well as the public key of the oracle service and $(x, w) \in R$ if the witness

w contains a proof that the current share price equals the specified one signed by the oracle. Initially, the user creates a transaction tx containing the trade description and encrypts it using the public key of the SO-TWE scheme, i.e., $c = \mathsf{Encrypt}(pk, x, \mathsf{tx})$, where the statement is from the specified language. The user sends the ciphertext c to its broker. Next, the broker regularly requests the current share price together with a proof from the oracle and provides this information as the witness w together with the ciphertext c to the decryption committee. Each committee member performs $\mathsf{ShareDec}(\mathsf{pk}, c, x, w, (i, \mathsf{sk}_i))$ to obtain a decryption share (i, d_i). After obtaining s valid shares from the committee, the broker executes $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, x\{(i, d_i)\}_{i \in \mathcal{S}})$. If decryption was executed with the intended share price, the result is tx. In this case, the broker executes the transaction which effectively performs the trade. Otherwise, the output of the $\mathsf{Combine}$-algorithm does not constitute a valid transaction.

The statement-obliviousness property guarantees that no party, not even the broker or the oracles, gets to know anything about the trade, neither the asset, the amount or the specified price, until the transaction is successfully decrypted and the trade can be executed. This way, we prevent insider trading. To incentivize the broker to execute the trade reliably and timely, users can rely on multiple brokers rewarding the one executing the trade first.

Acknowledgments

The first, third, and fourth authors were supported by the German Federal Ministry of Education and Research (BMBF) *iBlockchain project* (grant nr. 16KIS0902), by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) *SFB 1119 – 236615297 (CROSSING Project S7)*, and by the BMBF and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the *National Research Center for Applied Cybersecurity ATHENE*. The second author was supported by ISF grant No. 1316/18 and by the Algorand Centres of Excellence programme managed by Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Algorand Foundation.

References

1. S. Garg, C. Gentry, A. Sahai, and B. Waters, "Witness encryption and its applications," in *STOC*, 2013.
2. A. R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers, "Fairness in an unfair world: Fair multiparty computation from public bulletin boards," in *CCS*, 2017.
3. S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "How to run turing machines on encrypted data," in *CRYPTO*, 2013.
4. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," in *FOCS*, 2013.

5. C. Gentry, A. B. Lewko, and B. Waters, "Witness encryption from instance independent assumptions," in *CRYPTO*, 2014.
6. J. Liu, T. Jager, S. A. Kakvi, and B. Warinschi, "How to build time-lock encryption," *Des. Codes Cryptogr.*, 2018.
7. D. Boneh, D. B. Glass, D. Krashen, K. E. Lauter, S. Sharif, A. Silverberg, M. Tibouchi, and M. Zhandry, "Multiparty non-interactive key exchange and more from isogenies on elliptic curves," *J. Math. Cryptol.*, 2020.
8. V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song, "Fast batched dpss and its applications," in *PKC*, 2022.
9. A. Erwig, S. Faust, and S. Riahi, "Large-scale non-interactive threshold cryptosystems through anonymity," *IACR Cryptol. ePrint Arch.*, 2021.
10. M. Campanelli, B. David, H. Khoshakhlagh, A. K. Kristensen, and J. B. Nielsen, "Encryption to the future: A paradigm for sending secret messages to future (anonymous) committees," *IACR Cryptol. ePrint Arch.*, 2021.
11. V. Goyal, E. Masserova, B. Parno, and Y. Song, "Blockchains enable non-interactive MPC," in *TCC*, 2021.
12. G. Almashaqbeh, F. Benhamouda, S. Han, D. Jaroslawicz, T. Malkin, A. Nicita, T. Rabin, A. Shah, and E. Tromer, "Gage MPC: bypassing residual function leakage for non-interactive MPC," *Proc. Priv. Enhancing Technol.*, 2021.
13. F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin, "Can a public blockchain keep a secret?" in *TCC*, 2020.
14. C. Gentry, S. Halevi, B. Magri, J. B. Nielsen, and S. Yakoubov, "Random-index PIR and applications," in *TCC*, 2021.
15. S. Arita and K. Tsurudome, "Construction of threshold public-key encryptions through tag-based encryptions," in *ACNS*, 2009.
16. X. Boyen and B. Waters, "Anonymous hierarchical identity-based encryption (without random oracles)," in *CRYPTO*, 2006.
17. C. Gentry, "Practical identity-based encryption without random oracles," in *EUROCRYPT*, 2006.
18. D. Boneh, X. Boyen, and S. Halevi, "Chosen ciphertext secure public key threshold encryption without random oracles," in *CT-RSA*, 2006.
19. V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," in *EUROCRYPT*, 1998.
20. P. Mol and S. Yilek, "Chosen-ciphertext security from slightly lossy trapdoor functions," in *PKC*, 2010.
21. B. Libert and M. Yung, "Non-interactive cca-secure threshold cryptosystems with adaptive security: New framework and constructions," in *TCC*, 2012.
22. V. Koppula and B. Waters, "Realizing chosen ciphertext security generically in attribute-based encryption and predicate encryption," in *CRYPTO*, 2019.
23. T. E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *CRYPTO*, 1984.
24. D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *CRYPTO*, 2001.
25. C. Schnorr, "Efficient identification and signatures for smart cards," in *CRYPTO*, 1989.
26. A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986.
27. D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *CRYPTO*, 1992.
28. D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *EUROCRYPT*, 1996.

29. A. Shamir, "Identity-based cryptosystems and signature schemes," in *CRYPTO*, 1984.
30. E. Boyle, N. Gilboa, Y. Ishai, H. Lin, and S. Tessaro, "Foundations of homomorphic secret sharing," in *ITCS*, 2018.
31. R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," Massachusetts Institute of Technology. Laboratory for Computer Science, Tech. Rep., 1996.

Appendix

A Reduction: Extractability to SO-IND-CCA Security

This section provides further details to the notion of extractable threshold witness encryption and presents the reduction from extractability to SO-IND-CCA security in the threshold setting.

Intuitively, the original notion of extractable witness encryption states that any adversary that is able to obtain non-trivial information about a plaintext is also able to provide the witness for the corresponding ciphertext. Formally, this is defined by allowing the adversary to win the security game with non-negligible advantage but requiring that such an adversary can be used to extract the witness for the challenged plaintext. It is natural to translate this notion from witness encryption to our context, the one of threshold witness encryption, by defining extrability via the same security game as the one of SO-IND-CCA security, $\mathsf{Exp}^{\mathsf{SO\text{-}CCA}}$, with the only difference that the adversary is allowed to query the decryption oracle $\mathcal{O}(\cdot,\cdot,\cdot,\cdot,\cdot)$ with any ciphertext-witness pair while the oracle in the original experiment returns $\perp$ if $c = c^*$, $x \in \{x_0, x_1\}$ and $(x,(w_s,w_p)) \in R$. We call this game $\mathsf{Exp}^{\mathsf{SO\text{-}Ext}}_{\mathsf{TWE},\mathcal{A}}$, when played with an adversary $\mathcal{A}$ for a scheme TWE.

Extractability now requires that if the adversary has a non-negligible advantage in the security game, then it is possible to construct a witness extractor that extracts a valid witness with non-negligible probability.

Definition 7 (Extractability of SO-TWE). *Let $\mathcal{A}$ be a PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ such that the following holds: for every pk generated by Setup, for every x_0, x_1, m_0, m_1 and every auxiliary information $z \in \{0,1\}^{\mathsf{poly}(\kappa)}$:*

$$\Pr[\mathsf{Exp}^{\mathsf{SO\text{-}Ext}}_{\mathsf{TWE},\mathcal{A}}(1^\kappa) = 1] \geq \frac{1}{4} + \frac{1}{\mathsf{poly}(\kappa)}.$$

Then there exists a PPT extractor $\mathcal{E}$ such that:

$$\Pr[(b,w) = \mathcal{E}(1^\kappa, x_0, x_1, z) : (x_b, w) \in R] \geq \frac{1}{\mathsf{poly}(\kappa)}.$$

We state the following theorem:

Theorem 5. *Any statement-oblivious threshold witness encryption scheme TWE, that is SO-IND-CCA secure, is also* extractable.

Proof. Assume an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ that breaks extractability of TWE. This means that $\mathcal{A}$ is able to win game $\mathsf{Exp}^{\mathsf{SO\text{-}Ext}}_{\mathsf{TWE},\mathcal{A}}(1^\kappa)$ with a non-negligible advantage and there is no extractor $\mathcal{E}$.

From the fact that there is no extractor, we can derive that $\mathcal{A}$ does not query the oracle with input $(\cdot,\cdot,x_b,w_s,w_p)$ such that $x_b \in \{x_0,x_1\}$ and $R(x_b,(w_s,w_p)) = \mathsf{true}$; otherwise, there would exist the trivial extractor $\mathcal{E}_{\mathsf{triv}}$ that equals $\mathcal{A}$ up to this query and then outputs $(x_b,(w_s,w_p)$.

As the adversary $\mathcal{A}$ does not make such queries, it can be used in the SO-IND-CCA game without any modifications and will still win with non-neglgible probability.

B Security Proof: Statement-Oblivious Threshold Witness Encryption

This section presents the security proof for Construction 1 in Section 5; the generic transformation from a threshold tag-based encryption scheme TTBE that is OB-IND-CCA secure and a collision-resistant hash function H to a threshold witness encryption scheme $\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}}$ that is SO-IND-CCA secure. Formally, we prove Theorem 1.

Proof. We will show security via reductions to the underlying primitives, TTBE and H. For simplicity, we do not deal with the *bad events* of collisions in the hash functions via dedicated game-hops, but incorporate those bad events into the reduction to the TTBE scheme.

In particular, we construct a O-TTBE adversary $\mathcal{D}$ playing the experiment $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE},\mathcal{D}}$ with a challenger $\mathcal{C}$ that makes black-box use of a SO-TWE adversary $\mathcal{A}$. In this experiment, $\mathcal{D}$ has access to a decryption oracle $\mathcal{O}$ as defined in Section 4. We will show that $\mathcal{A}$ can only have a non-negligible advantage in game $\mathsf{Exp}^{\mathsf{SO\text{-}CCA}}_{\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}},\mathcal{A}}$, if $\mathcal{D}$ is able to find a hash collision or win the O-TTBE game $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE},\mathcal{D}}$ with non-negligible advantage.

For every adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ on the security game $\mathsf{Exp}^{\mathsf{SO\text{-}CCA}}_{\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}},\mathcal{A}}$, we define the SO-TWE adversary $\mathcal{D}$ as follows:

- When invoked with 1^κ, $\mathcal{D}$ returns $\mathcal{M} \leftarrow \mathcal{A}_0(1^\kappa)$ to $\mathcal{C}$.
- When receiving $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in\mathcal{M}})$, $\mathcal{D}$ calls $(x_0, x_1, m_0, m_1) \leftarrow \mathcal{A}_1(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in\mathcal{M}})$.
- During the execution of $\mathcal{A}_1$, $\mathcal{D}$ answers decryption queries of the form (i, c, x, w) from $\mathcal{A}$ as follows:
  - If $(x, w) \in R = \mathsf{false}$, $\mathcal{D}$ returns $(i, \perp)$.
  - Otherwise, $\mathcal{D}$ returns $(i, d_i) := \mathcal{O}(c, H(x), i)$.
- $\mathcal{D}$ calculates $t_0^* = H(x_0)$ and $t_1^* = H(x_1)$ and sends (t_0^*, t_1^*, m_0, m_1) to $\mathcal{C}$. When receiving c^* from $\mathcal{C}$, $\mathcal{D}$ calls $(\alpha^*, \beta^*) \leftarrow \mathcal{A}_2(c^*)$ and returns guess bits (α^*, β^*) to $\mathcal{C}$.
  Note that if $\mathcal{C}$ selects challenge bits (α, β), this corresponds to $\mathcal{D}$ selecting challenge bits $(\alpha^{\mathsf{TWE}}, \beta^{\mathsf{TWE}}) = (\alpha, \beta)$ in the SO-TWE security game.
- During the execution of $\mathcal{A}_2$, $\mathcal{D}$ answers decryption queries of the form (i, c, x, w) from $\mathcal{A}$ as follows:
  - If $(x, w) \in R$ or $(c, x) \in \{(c^*, x_j^*)\}_{j\in\{0,1\}}$, $\mathcal{D}$ returns $(i, \perp)$.
  - If $\exists j \in \{0, 1\}$, such that $x \neq x_j^*$ but $H(x) = t_j^*$, $\mathcal{D}$ ends the experiment with $(\mathsf{hash\text{-}collision}, x, x_j^*)$ and guesses bits $(0, 0)$.
  - Otherwise, $\mathcal{D}$ queries $\mathcal{O}$ with $(c, H(x), i)$ and returns the received decryption share (i, d_i) to $\mathcal{A}$. Note that the query to $\mathcal{O}$ is always valid, as $\mathcal{D}$ does not query with (c^*, t_0^*) or (c^*, t_1^*).

Let us start by assuming that there is no bad event. Observe that in this case it holds that (a) $\mathcal{D}$'s simulation of the challenger in game $\mathsf{Exp}^{\mathsf{SO\text{-}CCA}}_{\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}},\mathcal{A}}$ is perfect and (b) $\mathcal{D}$ wins $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE},\mathcal{D}}$ iff $\mathcal{A}$ wins $\mathsf{Exp}^{\mathsf{SO\text{-}CCA}}_{\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}},\mathcal{A}}$. Hence, it follows that the difference in $\mathcal{D}$'s advantage, $\mathsf{Adv}_{\mathcal{D}}$, and $\mathcal{A}$'s advantage, $\mathsf{Adv}_{\mathcal{A}}$, in the respective games is smaller than the probability of a bad event $\mathsf{Pr}[\mathsf{bad}]$, i.e.,

$$|\mathsf{Adv}_{\mathcal{D}} - \mathsf{Adv}_{\mathcal{A}}| < \mathsf{Pr}[\mathsf{bad}] \tag{1}$$

which implies that

$$\mathsf{Adv}_{\mathcal{D}} + \mathsf{Pr}[\mathsf{bad}] > \mathsf{Adv}_{\mathcal{A}}. \tag{2}$$

The only possible bad event is $(\mathsf{hash\text{-}collision}, x, x_j^*)$ which directly yields a hash collision. It follows that

$$\mathsf{Adv}_{\mathcal{D}} + \mathsf{Pr}[\mathsf{hash\text{-}collision}] > \mathsf{Adv}_{\mathcal{A}} \tag{3}$$

Therefore, if $\mathsf{Adv}_{\mathcal{A}}$ is a non-neglgible function, then at least one of $\mathsf{Adv}_{\mathcal{D}}$ and $\mathsf{Pr}[\mathsf{hash\text{-}collision}]$ must also be non-negligible which contradicts our assumptions.

Next, we show decryption consistency under chosen ciphertext attacks. We do so via a reduction to the decryption consistency TTBE scheme. In particular, we construct an TTBE-adversary $\bar{\mathcal{D}}$ playing the experiment $\mathsf{Exp}^{\mathsf{O\text{-}DC}}_{\mathsf{TTBE},\bar{\mathcal{D}}}$ with a challenger $\bar{\mathcal{C}}$ that makes black-box use of a SO-TWE adversary $\bar{\mathcal{A}}$. In this experiment, $\bar{\mathcal{D}}$ has again access to the decryption share oracle $\mathcal{O}$ as defined in Section 4. For every adversary $\bar{\mathcal{A}} = (\bar{\mathcal{A}}_0, \bar{\mathcal{A}}_1)$ on the security game $\mathsf{Exp}^{\mathsf{SO\text{-}DC}}_{\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}},\bar{\mathcal{A}}}$, we define the O-TTBE adversary $\bar{\mathcal{D}}$ as follows:

- When invoked with 1^κ, $\bar{\mathcal{D}}$ returns $(\mathcal{M}) \leftarrow \bar{\mathcal{A}}_0(1^\kappa)$ to $\bar{\mathcal{C}}$.
- When receiving $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in\mathcal{M}})$, $\bar{\mathcal{D}}$ calls $(x, c, \{(i, d_i)\}_{i\in\mathcal{S}}, \{(i, d_i')\}_{i\in\mathcal{S}'}) \leftarrow \bar{\mathcal{A}}_1(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in\mathcal{M}})$ and returns $(H(x), c, \{(i, d_i)\}_{i\in\mathcal{S}}, \{(i, d_i')\}_{i\in\mathcal{S}'})$.
- During the execution of $\bar{\mathcal{A}}_1$, $\bar{\mathcal{D}}$ answers decryption queries of the form (i, c, x, w) from $\bar{\mathcal{A}}$ as follows:
  - If $(x, w) \in R$, return $(i, \perp)$.
  - Otherwise, $\bar{\mathcal{D}}$ returns $(\imath, d_i) \leftarrow \mathcal{O}(c, H(x), i)$.

Obviously, $\mathcal{D}$ perfectly simulates the game $\mathsf{Exp}^{\mathsf{SO\text{-}DC}}_{\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}},\bar{\mathcal{A}}}$. We complete the proof by showing that $\bar{\mathcal{D}}$ wins security game $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE},\bar{\mathcal{D}}}$ if $\bar{\mathcal{A}}$ wins security game $\mathsf{Exp}^{\mathsf{SO\text{-}DC}}_{\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}},\bar{\mathcal{A}}}$.

In order to win, $\bar{\mathcal{A}}$ has to sent a tuple

$$(x, c, \{(i, d_i)\}_{i\in\mathcal{S}}, \{(i, d_i')\}_{i\in\mathcal{S}'})$$

for which the following holds:

$$\mathcal{S}, \mathcal{S}' \subseteq [n] \text{ and } |\mathcal{S}| = |\mathcal{S}'| = s \tag{4}$$

$$\begin{aligned} \forall i \in \mathcal{S} :& \mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, x, (i, d_i)) = \mathsf{true} \\ \text{and } \forall i \in \mathcal{S}' :& \mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, x, (i, d_i')) = \mathsf{true} \end{aligned} \tag{5}$$

$$\begin{aligned} \perp \neq m \neq m' \neq \perp \text{ for } m :=& \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, x, \{(i, d_i)\}_{i\in\mathcal{S}}) \\ \text{and } m' :=& \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, x, \{(i, d_i')\}_{i\in\mathcal{S}'}) \end{aligned} \tag{6}$$

In scheme $\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}}$ this implies that:
From (5):

$$\begin{aligned} &\forall i \in \mathcal{S} : \mathsf{TTBE.ShareVf}(\mathsf{pk}, \mathsf{vk}, H(x), (i, d_i)) = \mathsf{true} \\ &\text{and } \forall i \in \mathcal{S}' : \mathsf{TTBE.ShareVf}(\mathsf{pk}, \mathsf{vk}, H(x), (i, d'_i)) = \mathsf{true} \end{aligned} \tag{7}$$

From (6):

$$\begin{aligned} &\bot \neq m \neq m' \neq \bot \\ &\text{for } m := \mathsf{TTBE.Combine}(\mathsf{pk}, \mathsf{vk}, c, H(x), \{(i, d_i)\}_{i \in \mathcal{S}}) \\ &\text{and } m' := \mathsf{TTBE.Combine}(\mathsf{pk}, \mathsf{vk}, c, H(x), \{(i, d'_i)\}_{i \in \mathcal{S}'}) \end{aligned} \tag{8}$$

In the reduction, $\bar{\mathcal{D}}$ sends the tuple $(H(x), c, \{(i, d_i)\}_{i \in \mathcal{S}}, \{(i, d'_i)\}_{i \in \mathcal{S}'})$. Note, that properties (4), (7), (8), are exactly what is required in order to win the security game $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE},\bar{\mathcal{D}}}$. It follows that $\bar{\mathcal{D}}$ wins security game $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE},\bar{\mathcal{D}}}$ if $\bar{\mathcal{A}}$ wins security game $\mathsf{Exp}^{\mathsf{SO\text{-}DC}}_{\mathsf{SO\text{-}TWE}_{\mathsf{OTTBE}},\bar{\mathcal{A}}}$ which implies that $\bar{\mathcal{D}}$'s advantage, $\mathsf{Adv}_{\bar{\mathcal{D}}}$, is larger or equal to $\bar{\mathcal{A}}$'s advantage, $\mathsf{Adv}_{\bar{\mathcal{A}}}$, in the respective games:

$$\mathsf{Adv}_{\bar{\mathcal{D}}} \leq \mathsf{Adv}_{\bar{\mathcal{A}}} \tag{9}$$

Therefore, if $\mathsf{Adv}_{\bar{\mathcal{A}}}$ is a non-neglgible function, then $\mathsf{Adv}_{\bar{\mathcal{D}}}$ must also be non-negligible which contradicts our assumptions. This concludes the security proof.

C Security Proof: O-TTBE from Bilinear Mappings and Random Oracles

In this section, we present the formal security proof of Construction 2 in Section 6, i.e., we show security of our O-TTBE construction from bilinear mappings in the random oracle model. Formally, we show Theorem 2.

Proof. We start showing oblivious indistinguishable message under chosen-ciphertext attacks and continue with proving decryption consistency under chosen-ciphertext attacks.

Oblivious indistinguishable messages. We prove oblivious indistinguishable messages under chosen-ciphertext attacks via a reduction to the DBDH assumption. In particular, we build a DBDH distinguisher $\mathcal{D}$ that makes use of an adversary $\mathcal{A}$ on the CCA experiment $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE},\mathcal{A}}$. The distinguisher is defined as follows:

- $\mathcal{D}$ receives a tuple $(\bar{h}, \bar{g}, \alpha, \beta, \gamma)$ for public parameters $(e, \mathbb{G}, \mathbb{G}_T, q)$. It holds that $\alpha = \bar{g}^x$ and $\beta = \bar{g}^y$ for random unknown (x, y) and $\gamma = e(\bar{h}, \bar{g})^z$ with z either being xy or random. $\mathcal{D}$ starts by calling $\mathcal{M} \leftarrow \mathcal{A}_0$.
- $\mathcal{D}$ answers random oracle queries as follows:
  - For H_1, H_2, and H_3, $\mathcal{D}$ behaves like a standard random oracle with the difference that $\mathcal{D}$ can program keys that have not been queried yet.

- For H_2, $\mathcal{D}$ samples random $k \in_R \mathbb{Z}_q$ and returns $T = \bar{g}^k$. This allows $\mathcal{D}$ to extract $\log_{\bar{g}}(T)$ for any T sampled with H_2. We write $H_2[t, A]$ for this extraction. In addition, $\mathcal{D}$ can program keys that have not been queried, yet. For programmed keys, $H_2[t, A]$ returns $\perp$.

– $\mathcal{D}$ simulates the setup by sampling $x_i \in_R \mathbb{Z}_q$ for $i \in \mathcal{M}$ and defining

$$
\begin{aligned}
&g := \bar{g},\ X := \alpha,\ \mathcal{M}' = \mathcal{M} \cup \{0\},\\
&\text{For } i \in \mathcal{M}:\ \mathsf{vk}_i := g^{x_i},\ \mathsf{sk}_i := x_i\\
&\text{For } j \in [n] \setminus \mathcal{M} : \mathsf{vk}_j := \alpha^{\lambda_{j,0}^{\mathcal{M}'}} \cdot \prod_{i \in \mathcal{M}} \mathsf{vk}_i^{\lambda_{j,i}^{\mathcal{M}'}}\\
&\mathsf{pk} := (g, X),\ \mathsf{vk} := \{\mathsf{vk}_i\}_{i \in [n]}.
\end{aligned}
$$

– $\mathcal{D}$ calls $(t_0, t_1, m_0, m_1) \leftarrow \mathcal{A}_1(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i \in \mathcal{M}})$. During this call, $\mathcal{D}$ answers decryption queries of the form (i, c, t) to the decryption oracle $\mathcal{O}$ as follows:
  - If $i \in \mathcal{M}$, return $\mathsf{ShareDec}(\mathsf{pk}, c, t, (i, \mathsf{sk}_i))$.
  - If $i \notin \mathcal{M}$ and $\mathsf{ValidateCT}(c) = \mathsf{false}$, return $(i, \perp)$.
  - If $i \notin \mathcal{M}$ and $\mathsf{ValidateCT}(c) = \mathsf{true}$, parse $c = (\cdot, A, \cdot)$, sample $w_i, f_i \in_R \mathbb{Z}_q$ and calculate

$$
\begin{aligned}
T &:= H_2(t, A),\ k := H_2[t, A] = \log_g(T)\\
D_i &:= e(\mathsf{vk}_i, A^k) = e(T, A^{x_i})\\
U_i &:= \frac{e(T', A)^{f_i}}{D_i^{w_i}},\ V_i = \frac{e(T', g)^{f_i}}{e(T', \mathsf{vk}_i)^{w_i}}.
\end{aligned}
$$

    Then, program $H_4(D_i, U_i, V_i) \leftarrow w_i$ and return $(i, D_i, \pi_i := (w_i, f_i))$. If H_4 cannot be programmed emit event (failed-programming) and output 0. If $k = \perp$ emit event (oracle-collision) and output 0.

– To create the challenge ciphertext, $\mathcal{D}$ samples a uniform tag choice bit b and creates a valid ciphertext for tag t_b. Then, $\mathcal{D}$ programs the random oracle H_2 such that a hypothetical encryption (and hence decryption) for tag t_{1-b} would query H_1 with key γ iff the received tuple is a DBDH tuple. In particular, $\mathcal{D}$ creates the challenge ciphertext $c^* = (M^*, A^*, \pi^*)$ by sampling random bits $b, b_m \in_R \{0, 1\}$ and exponents $w, f \in_R \mathbb{Z}_q$, defining

$$
\begin{aligned}
A^* &:= \beta = g^y\\
\widetilde{T} &:= H_2(t_b, A^*),\ \tilde{k} := H_2[t_b, A^*] = \log_g(\widetilde{T})\\
\tilde{\gamma} &:= e(X^{\tilde{k}}, A^*) = e(\widetilde{T}, A^*)^x\\
M^* &:= m_{b_m} \oplus H_1(\tilde{\gamma})\\
U^* &:= \frac{g^f}{(A^*)^w},\ \pi^* = (w, f)
\end{aligned}
$$

  and programming $H_2(t_{1-b}, A^*) \leftarrow \bar{h}$ and $H_3(M^*, A^*, U^*) \leftarrow w$. If H_2 or H_3 have already been queried at said keys, emit event (failed-programming) and output 0.

– $\mathcal{D}$ calls $(\cdot, \cdot) \leftarrow \mathcal{A}_2(c^*)$. During this call, $\mathcal{D}$ answers decryption queries as above with the only difference that queries in $\{(\cdot, A^*, t_0), (\cdot, A^*, t_1)\}$ are answered with $(i, \perp)$.

– If the adversary queried H_1 at γ, $\mathcal{D}$ outputs 1; otherwise, if the adversary terminates without querying H_1 at γ, $\mathcal{D}$ outputs 0.

The reduction is based on the fact that the only value the adversary receives that is dependent on the tag or the message is M^*. As M^* is calcu-

lated based on the random oracle H_1, it is obvious that an adversary breaking the security of the scheme when receiving a challenge ciphertext $c^* = (A^*, M^*, \pi^*)$, needs to query the random oracle H_1 at least at one of $\{P_0, P_1\} := \{e(H_2(t_0, A^*), A^*)^x, e(H_2(t_1, A^*), A^*)^x\}$. By assuming that $\mathcal{A}$ has a non-negligible advantage in the CCA game $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE},\mathcal{A}}$, we can conclude that $\mathcal{A}$ queries H_1 at P_0 or P_1 (or both) with non-negligible probability $\mathsf{Advantage}_{\mathcal{A}}$. The distinguisher $\mathcal{D}$ simulates the experiment such that, for a randomly chose bit b, it creates a valid ciphertext based on P_b but ensures that P_{1-b} equals γ iff the received tuples is a DBDH tuple. This way, $\mathcal{D}$ can identify a BDDH tuple based on the event that H_1 has been queried at γ.

We start by showing that the view of an adversary interacting with $\mathcal{D}$ is computationally indistinguishable from the view of the adversary when playing the CCA game $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE},\mathcal{A}}$. As $\mathcal{D}$ programs H_1, H_3, H_4 only with randomly sampled elements and simulates them, besides the programming, as standard random oracles, it follows that H_1's, H_3's and H_4's replies are identically distributed in the reduction and in the original game. For the simulation of H_2, $\mathcal{D}$ samples $k \in_R \mathbb{G}$ and returns $\bar{g}^k$. Since k is random and $\bar{g}$ is a generator of $\mathbb{G}$, returning $\bar{g}^k$ yields the same distribution as sampling and returning $K \in_R \mathbb{G}$ directly. Further, $\mathcal{D}$ only programs H_2, once, with a random element $\bar{h}$. It follows that H_2's replies are identically distributed in the reduction and the original game. Regarding the setup, $\mathcal{D}$ defines a public key $\mathsf{pk} := (g, \alpha = \bar{g}^x)$ for randomly sampled $\bar{g}$ and x (but without actually knowing x). By sampling s random secret key shares x_i, $\mathcal{D}$ implicitly defines a random sharing polynomial F of degree $s-1$ for $F(0) = x$. As $\bar{g}$, x and F are sampled uniformly random (under the condition that $F(0) = x$) and it holds that $\mathsf{pk} := (g, \bar{g}^x)$, $\mathsf{sk}_i = F(i)$ and $\mathsf{vk}_i = g^{F(i)}$, we can conclude that the distribution of the setup is identically distributed in the reduction and the original game. Regarding the challenge ciphertext c^*, the adversary receives $(M^*, A^*, (w^*, f^*))$ with random A^*, $M^* = H_1(e(H_2(t_b, A^*), A^*)^x) \oplus m_{b_m}$ (for random bits b, b_m) and random exponents w^*, f^* subject to the condition that $w^* = H_3(M^*, A^*, \frac{g^{f^*}}{A^{w^*}})$. This is exactly what an adversary would see in the real experiment. It follows that the challenge ciphertext c^* is identically distributed in the reduction and the original game (expect the bad event which we analyze below). Regarding the decryption queries (of the form (i, c, t)), each reply constitutes a valid decryption, as the adversary receives $(i, \perp)$ if $(c, t) \in \{(c^*, t_0), (c^*, t_1)\}$ or $\mathsf{ValidateCT}(c) = \mathsf{false}$, and $(i, e(H_2(t, A^*), (A^*)^{F(i)}), \pi_i = (w_i, f_i))$ for randomly sampled w_i, f_i subject to the condition that $w_i = H_4(D_i, \frac{e(H_2(t,A^*),A)^{f_i}}{D_i^{w_i}}, \frac{e(H_2(t,A^*),g)^{f_i}}{e(H_2(t,A^*),\mathsf{vk}_i)^{w_i}})$. It follows that the decryption queries are identically distributed in the reduction and the original game (expect the bad event which we analyze below). Having shown that all messages sent from $\mathcal{D}$ to $\mathcal{A}$ are distributed identically in the reduction and the real experiment, it remains to show that the bad events happen with at most negligible probability. First, when answering decryption queries, there are two different kinds of bad events, $\mathsf{failed\text{-}programming}$ and $\mathsf{oracle\text{-}collision}$. The former happens if H_4 has been queried at (D_i, U_i, V_i) before the decryption query. As U_i and V_i are calculated based on freshly and randomly sampled w_i and f_i, this

event happens with at most negligible probability. The latter bad event happens if $k = H_2[t, A] = \perp$. This happens only for keys (t_{1-b}, A^*) as H_2 is only programmed for those keys. However, the probability that the keys (t_{1-b}, A^*) are queried within a decryption query are negligible, as we argue next. If the adversary queries these values before the challenge ciphertext c^* is created, there would not be an event oracle-collision as the random oracle has not been programmed, yet. However, the creation of the challenge ciphertext c^* would fail with event failed-programming as $\mathcal{D}$ cannot execute $H_2(t_{1-b}, A^*) \leftarrow \bar{h}$. If the adversary submits a decryption query (t, c) containing (t_{1-b}, A^*) after the challenge ciphertext c^* is created, $\mathcal{D}$ declines decryption if $(t, c) \in \{(t_0, c^*), (t_1, c^*)\}$ (as in the original experiment). Hence, $\mathcal{D}$ queries (t_{1-b}, A^*) only if $c \neq c^*$ but $A = A^*$. In order to create such a ciphertext $c \neq c^*$ containing A^*, the adversary needs to forge the zero knowledge argument. In detail, the adversary needs to come up with $\pi' = (w', f')$ such that $g^{f'} = g^{r'} \cdot A^{*w'}$. Since w' is the output of a random oracle where at least one input must be different than the inputs used to compute the value w in the challenge ciphertext and the adversary has to fix r' before obtaining w', there is only one value for which the adversary can hope to create a valid proof. The probability for this is $1/q$, where q is the group order. Although the adversary can try polynomial many values for w' the probability of forging the zero knowledge argument is still negligible. Hence, A^* can only be used as part of c^* except with negligible probability. It follows that the probability of the event oracle-collision when answering decryption queries is negligible. Second, when creating the challenge ciphertext, there is another bad event failed-programming that happens either if H_2 has already been queried at keys (t_{1-b}, A^*) or H_3 has been queried at keys (M^*, A^*, U^*). In both cases, $A^* = \beta$ is a uniform random element in $\mathbb{G}$ which implies that the probability of an adversary querying H_2 or H_3 at the given keys, before the challenge ciphertext is created, is negligible. Having shown that all messages sent from $\mathcal{D}$ to $\mathcal{A}$ are distributed identically in the reduction and the real experiment and that the probability of a bad event is at most negligible, we can conclude that the view of the adversary in the reduction is computationally indistinguishable from a view of the adversary in the real experiment.

To analyze the success probability of $\mathcal{D}$, we start with the case that the received tuple is a DBDH tuple. In this case, it holds that $P_{1-b} = e(H_2(t_{1-b}, A^*), A^*)^x = e(\bar{h}, A^*)^x = e(\bar{h}, g^{xy}) = \gamma$; remember that $\mathcal{D}$ programs $H_2(t_{1-b}, A^*) \leftarrow \bar{h}$. As b is sampled uniformly random, we conclude that $\mathcal{A}$ queries H_1 at γ at least with probability $\frac{1}{2} \cdot \mathsf{Advantage}_{\mathcal{A}} - \mathsf{negl}$. As $\mathcal{D}$ guesses 1 (the tuple is a DBDH tuple) only if the adversary queries γ, the success probability of $\mathcal{D}$ in the case that the received tuple is a DBDH tuple, is

$$\Pr[\mathsf{success} \mid \text{DBDH tuple}] \geq \frac{1}{2} \cdot \mathsf{Advantage}_{\mathcal{A}} - \mathsf{negl}.$$

We have to include the $(-\mathsf{negl})$ *as the experiment can fail with negligible probability, as discussed above, which leads to a* 0 *guess. Further, the probability is larger or equal ("*$\geq$*") and a not equal ("*$=$*") to the given term, as the adversary*

could also query both P_0 and P_1 instead of the worst case in which it queries just one of them.

Next, we continue with the case that the received tuple is no DBDH tuple. Recall that $\mathcal{D}$ guesses 1 only iff $\mathcal{A}$ queries H_1 at key γ. As γ is a uniform random element in $\mathbb{G}_3$ it holds that $\mathcal{A}$ queries H_2 at key γ with only negligible probability. It follows that the success probability of $\mathcal{D}$ in case that the received tuple is no DBDH tuple, is

$$\Pr[\mathsf{success} \mid \text{no DBDH-tuple}] = 1 - \mathsf{negl}.$$

For randomly chosen challenge tuples it holds that $\mathcal{D}$ can distinguish between DBDH tuples and random tuples with probability:

$$\begin{aligned}\Pr[\mathsf{success}] \geq & \frac{1}{2} \cdot (\Pr[\mathsf{success} \mid \text{no DBDH-tuple}] \\ & + \Pr[\mathsf{success} \mid \text{ DBDH-tuple}]) \\ \geq & \frac{1}{2} \cdot (1 - \mathsf{negl} + \frac{1}{2}\mathsf{Advantage}_{\mathcal{A}} - \mathsf{negl}) \\ > & \frac{1}{2} + \mathsf{negl}\end{aligned}$$

This concludes the reduction and shows CCA security of the construction.

Decryption consistency. Next, we show decryption consistency. In particular, we show that an attacker $\mathcal{A}$ wins the security game $\mathsf{Exp}^{\mathsf{O\text{-}DC}}_{\mathsf{TTBE}_{\mathsf{pROM}},\mathcal{A}}$ with at most negligible probability. We show this by contradiction, i.e., we start by assuming $\mathcal{A}$ wins the game with non-negligible probability and show a contradiction to this assumption.

In the security game, the adversary obtains a public key pk, verification keys $\mathsf{vk} = \{\mathsf{vk}_i\}_{i\in[n]} = \{g^{x_i}\}_{i\in[n]}$ and the secret keys for the corrupted parties $\{\mathsf{sk}_i\}_{i\in\mathcal{M}} = \{x_i\}_{i\in\mathcal{M}}$. Next, $\mathcal{A}$ has to provide a tag t, a ciphertext $c = (M, A, \pi)$ and two sets of decryption shares $\{(i, d_i)\}_{i\in\mathcal{S}}$ and $\{(i, d'_i)\}_{i\in\mathcal{S}'}$, where $d_i = (i, D_i, \pi_i = (w_i, f_i))$ for $i \in \mathcal{S}$ and $d'_i = (i, D'_i, \pi'_i = (w'_i, f'_i))$ for $i \in \mathcal{S}'$. For a successful execution, all the checks in $\mathsf{Exp}^{\mathsf{O\text{-}DC}}_{\mathsf{TTBE}_{\mathsf{pROM}},\mathcal{A}}$ need to hold, i.e., all shares are valid and the combined messages are different.

Due to the correctness property of $\mathsf{TTBE}_{\mathsf{pROM}}$, there must be at least one j such that $D_j \neq e(T, A^{x_j})$ or $D'_j \neq e(T, A^{x_j})$ for $T := H_2(t, A)$ in order to get two different message $m \neq m'$. Wlog., we assume $D_j = e(T, A^{x^*_j}) \neq e(T, A^{x_j})$. Since $\mathcal{A}$ wins the game, $\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}_j, c, t, d_j) = \mathsf{true}$. In particular, $w_j = H_4(D_j, U_j, V_j)$ for

$$e(T, A)^{f_j} = U_j \cdot D_j^{w_j} \text{ and } e(T, g)^{f_j} = V_j \cdot e(T, \mathsf{vk}_j)^{w_j}, \tag{10}$$

where $T := H_2(t, A)$. Equation (10) implies

$$\begin{aligned}
&\log_{e(T,A)}(U_j \cdot D_j^{w_j}) = \log_{e(T,g)}(V_j \cdot e(T, \mathsf{vk}_j)^{w_j}) \\
\Leftrightarrow\ &\log_{e(T,A)}(e(T,A)^{r_j} \cdot e(T,A)^{x_j^* w_j}) = \\
&\qquad\qquad \log_{e(T,g)}(e(T,g)^{r'_j} \cdot e(T,g)^{x_j w_j}) \\
\Leftrightarrow\ &r_j + x_j^* w_j = r'_j + x_j w_j \\
\Leftrightarrow\ &(r_j - r'_j) + (x_j^* - x_j) w_j = 0.
\end{aligned}$$

Since $x_j^* - x_j \neq 0$, there is at most one challenge value w_j for which the adversary may hope that the equation holds and the proof is valid. Since w_j is the output of a random oracle, and hence, at random, the probability for w_j being the required value is at most $1/q$ where q is the order of $\mathbb{G}_T$. Although the adversary may try polynomial often to compute the challenge value with different r_j and r'_j, the probability is still negligible. This follows from the standard argument about the Fiat-Shamir heuristic [28]. Since this contradicts our initial assumption, it concludes the consistency proof.

D Further Definitions

D.1 Collision Resistance of Hash Functions

The *collision resistance* property of hash functions states that any PPT adversary can find two values x, x' such that $x \neq x'$ and $H(x) = H(x')$ only with negligible probability.

D.2 Security Properties of Digital Signatures

We assume digital signatures to satisfy *consistency* and *existential unforgeability against chosen-message attacks*. The *consistency* property states that for all $\kappa \in \mathbb{N}$, for all $(\mathsf{SigK}, \mathsf{VerK}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and for every $m \in \mathbb{M}$ it holds $\Pr[\mathsf{Verify}(\mathsf{VerK}, m, \mathsf{Sign}(\mathsf{SigK}, m))] = 1$.

We define *existential unforgeability against chosen-message attacks* via the following game

Experiment $\mathsf{Exp}^{\mathsf{EX\text{-}UNF}}_{\mathsf{SIG},\mathcal{A}}(\kappa)$
$(\mathsf{SigK}, \mathsf{VerK}) \leftarrow \mathsf{KeyGen}(1^\kappa)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{VerK})$
if $\mathsf{Verify}(\mathsf{VerK}, m^*, \sigma^*) = 1$ **then return** 1
else return 0

where the adversary may ask its oracle $\mathcal{O}$ on a message $m \in \mathbb{M}$ and gets back the signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{SigK}, m)$. The pair (m^*, σ^*) output by $\mathcal{A}$ must be different to any (m, σ) obtained by the oracle.

Definition 8. *A signature scheme* SIG *is* existentially unforgeable against chosen-message attacks *if for every* $\kappa \in \mathbb{N}$ *and every PPT adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr[\mathsf{Exp}^{\mathsf{EX\text{-}UNF}}_{\mathsf{SIG},\mathcal{A}}(\kappa) = 1] \leq \mathsf{negl}(\kappa).$$

Definition 9 (OTS). *A signature scheme* $\mathsf{OTS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is called* one-time signature *scheme with* existential unforgeability against chosen-message attacks*, if for every* $\kappa \in \mathbb{N}$ *and every PPT adversary* $\mathcal{A}'$ *that makes as most one oracle query there exists a negligible function* negl *such that*

$$\Pr[\mathsf{Exp}^{\mathsf{EX\text{-}UNF}}_{\mathsf{OTS},\mathcal{A}'}(\kappa) = 1] \leq \mathsf{negl}(\kappa).$$

D.3 Identity-Based Encryption

Identity-based encryption was first introduced by Shamir in 1984 [29]. We state the definition following Boneh and Franklin [24].

Definition 10 (IBE). *An* identity-based encryption *scheme* IBE *consists of four probabilistic polynomial-time algorithms:*

$$\mathsf{IBE} = (\mathsf{Setup}, \mathsf{Extract}, \mathsf{Encrypt}, \mathsf{Decrypt}).$$

such that

1. $\mathsf{Setup}(1^\kappa)$ *takes as input a security parameter* 1^κ *and outputs a public key* pk *including public parameters and a master key* msk*.* pk *include a description of the finite message space* $\mathcal{M}$ *and the finite ciphertext space* $\mathcal{C}$*.*
2. $\mathsf{Extract}(\mathsf{pk}, \mathsf{msk}, \mathsf{id})$ *takes as input the public key* pk*, the master key* msk *and an identity* $\mathsf{id} \in \{0,1\}^*$*. It outputs an identity secret key* $\mathsf{ik}_{\mathsf{id}}$*.*
3. $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, m)$ *takes as input the public key* pk*, an identity* id *and a message* m*. It outputs a ciphertext* c *encrypted under identity* id*.*
4. $\mathsf{Decrypt}(\mathsf{pk}, c, \mathsf{ik}_{\mathsf{id}})$ *takes as input the public key* pk*, a ciphertext* c *and an identity secret key* $\mathsf{ik}_{\mathsf{id}}$*. It outputs a message* m*.*

We require these algorithms to fulfill the consistency *property, namely for every* $\kappa \in \mathbb{N}$*, every output* $(\mathsf{pk}, \mathsf{msk})$ *of* $\mathsf{Setup}(1^\kappa)$*, every* $\mathsf{id} \in \{0,1\}^*$*, every* $\mathsf{ik}_{\mathsf{id}} \leftarrow \mathsf{Extract}(\mathsf{pk}, \mathsf{msk}, \mathsf{id})$ *and every* $m \in \mathcal{M}$*:*

$$\mathsf{Decrypt}(\mathsf{pk}, \mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, m), \mathsf{ik}_{\mathsf{id}}) = m.$$

Security of an IBE scheme can be defined as the anonymity and message indistiguishability against chosen-plaintext attacks properties for verifiable IBE in Definition 13.

D.4 Verifiable IBE

We state a definition for verifiable identity-based encryption as an extension of identity-based encryption presented by Boneh and Franklin [24]. In particular, the primitive contains a verification algorithm that allows to check if an identity key is generated correctly. For completeness, we provide the definition of plain identity-based encryption in Appendix D.3.

Definition 11 (VIBE). *A* verifiable identity-based encryption *scheme* VIBE *consists of five probabilistic polynomial-time algorithms:*

1. $\mathsf{Setup}(1^\kappa)$ *takes as input a security parameter* 1^κ *and outputs a public key* pk *including public parameters, a verification key* vk *and a master key* msk. pk *include a description of the finite message space* $\mathbb{M}$ *and the finite ciphertext space* $\mathbb{C}$.
2. $\mathsf{Extract}(\mathsf{pk}, \mathsf{msk}, \mathsf{id})$ *takes as input the public key* pk*, the master key* msk *and an identity* $\mathsf{id} \in \{0,1\}^*$*. It outputs an identity secret key* ik_id *together with a proof* ρ_id *stating that* ik_id *was computed correctly.*
3. $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, \mathsf{ik}_\mathsf{id}, \rho_\mathsf{id})$ *takes as input the public key* pk*, the verification key* vk*, an identity* id*, an identity key* ik_id*, and a proof* ρ_id*. It outputs 1 if* ik_id *is a valid identity key for identity* id *and 0 otherwise.*
4. $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, m)$ *takes as input the public key* pk*, an identity* id *and a message* m*. It outputs a ciphertext* c *encrypted under identity* id.
5. $\mathsf{Decrypt}(\mathsf{pk}, \mathsf{ik}_\mathsf{id}, c)$ *takes as input the public key* pk*, an identity secret key* ik_id *and a ciphertext* c*. It outputs a message* m.

We require these algorithms to fulfill the following correctness and verifiability *properties for all* $\kappa \in \mathbb{N}$*:*

- ***Correctness:*** *For every* $(\mathsf{pk}, \mathsf{vk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa)$*, every* $\mathsf{id} \in \{0,1\}^*$*, every* $(\mathsf{ik}_\mathsf{id}, \cdot) \leftarrow \mathsf{Extract}(\mathsf{pk}, \mathsf{msk}, \mathsf{id})$ *and every* $m \in \mathbb{M}$*:*

$$\mathsf{Decrypt}(\mathsf{pk}, \mathsf{ik}_\mathsf{id}, \mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, m)) = m.$$

- ***Verifiability:*** *For every* $(\mathsf{pk}, \mathsf{vk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa)$*, every* $\mathsf{id} \in \{0,1\}^*$ *and every* $(\mathsf{ik}_\mathsf{id}, \rho_\mathsf{id}) \leftarrow \mathsf{Extract}(\mathsf{pk}, \mathsf{msk}, \mathsf{id})$

$$\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, \mathsf{ik}_\mathsf{id}, \rho_\mathsf{id}) = 1.$$

We define security by three properties: *soundness*, *anonymity* and *security against chosen-plaintext attacks*. We start defining the soundness property. Informally, soundness means that an adversary cannot come up with two different but valid identity keys that decrypt a chosen ciphertext to two different plaintexts. Formally, we define the soundness property via the following game.

$$\begin{array}{l}\text{Experiment } \mathsf{Exp}^{\mathsf{SOUND}}_{\mathsf{VIBE},\mathcal{A}}(\kappa) \\ \hline (\mathsf{pk},\mathsf{vk},\mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa) \\ (\mathsf{ID}, c, (\mathsf{ik}_{\mathsf{ID}}, \rho_{\mathsf{ID}}), (\mathsf{ik}'_{\mathsf{ID}}, \rho'_{\mathsf{ID}})) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(\mathsf{pk},\mathsf{vk}) \\ \textbf{if } \mathsf{Verify}(\mathsf{pk},\mathsf{vk},\mathsf{ID},\mathsf{ik}_{\mathsf{ID}},\rho_{\mathsf{ID}}) = 1 \\ \quad \wedge\, \mathsf{Verify}(\mathsf{pk},\mathsf{vk},\mathsf{ID},\mathsf{ik}'_{\mathsf{ID}},\rho'_{\mathsf{ID}}) = 1 \\ \quad \wedge\, \mathsf{Decrypt}(\mathsf{pk},\mathsf{ik}_{\mathsf{ID}},c) \neq \mathsf{Decrypt}(\mathsf{pk},\mathsf{ik}'_{\mathsf{ID}},c) \\ \quad \textbf{return } 1 \\ \textbf{else} \\ \quad \textbf{return } 0 \end{array}$$

The adversary can use its oracle $\mathcal{O}(\cdot)$ to make identity key queries. More precisely, upon receiving id from $\mathcal{A}$ the oracle returns $(\mathsf{ik}_{\mathsf{id}}, \rho_{\mathsf{id}}) \leftarrow \mathsf{Extract}(\mathsf{pk},\mathsf{msk},\mathsf{id})$ for any $\mathsf{id} \in \{0,1\}^*$.

Definition 12 (Soundness). *A verifiable identity-based encryption scheme* VIBE *satisfies* soundness *if for all $\kappa \in \mathbb{N}$ and all PPT adversary*

$$\Pr[\mathsf{Exp}^{\mathsf{SOUND}}_{\mathsf{VIBE},\mathcal{A}}(\kappa) = 1] \leq \mathsf{negl}(\kappa).$$

We next move on to the anonymity and security against chosen-plaintext attacks. The anonymity property of a VIBE scheme informally states that an adversary cannot learn the associated identity from a ciphertext, while the security against chosen-plaintext attacks states that an adversary cannot distinguish two ciphertexts over different messages. We combine both properties following Gentry [17] and define security via the following game.

$$\begin{array}{l}\text{Experiment } \mathsf{Exp}^{\mathsf{A\text{-}V\text{-}CPA}}_{\mathsf{VIBE},\mathcal{A}}(\kappa) \\ \hline (\mathsf{pk},\mathsf{vk},\mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa) \\ (\mathsf{ID}_0, \mathsf{ID}_1, m_0, m_1) \leftarrow \mathcal{A}_0^{\mathcal{O}}(\mathsf{pk},\mathsf{vk}) \\ \alpha, \beta \in_R \{0,1\} \\ c^* \leftarrow \mathsf{Encrypt}(\mathsf{pk},\mathsf{ID}_\alpha, m_\beta) \\ (\alpha', \beta') \leftarrow \mathcal{A}_1^{\mathcal{O}}(c^*) \\ \textbf{return } (\alpha,\beta) = (\alpha',\beta') \end{array}$$

In the game $\mathsf{Exp}^{\mathsf{A\text{-}V\text{-}CPA}}_{\mathsf{VIBE},\mathcal{A}}$, the adversary can use its oracle $\mathcal{O}$ to make key generation queries. Upon receiving id, $\mathcal{O}$ returns $\mathsf{Extract}(\mathsf{pk},\mathsf{msk},\mathsf{id})$ if $\mathsf{id} \notin \{\mathsf{ID}_0, \mathsf{ID}_1\}$ and $\perp$ otherwise.

Definition 13 (ANON-IND-ID-CPA). *A VIBE scheme* VIBE *is* ANON-IND-ID-CPA *secure if for all PPT adversary $\mathcal{A}$ in game $\mathsf{Exp}^{\mathsf{A\text{-}V\text{-}CPA}}_{\mathsf{VIBE},\mathcal{A}}$, there exists a negligible function* negl *such that*

$$\left| \Pr[\mathsf{Exp}^{\mathsf{A\text{-}V\text{-}CPA}}_{\mathsf{VIBE},\mathcal{A}}(\kappa) = 1] - \frac{1}{4} \right| \leq \mathsf{negl}(\kappa).$$

In Appendix E, we show how to construct a VIBE scheme from a standard identity-based encryption scheme IBE combined with an existentially unforgeable signature scheme SIG. Assuming IBE satisfies ANON-IND-ID-CPA security, the VIBE construction satisfies soundness and ANON-IND-ID-CPA security.

D.5 Homomorphic Secret Sharing

We follow the definition of Boyle et al. [30] for homomorphic secret sharing (HSS) schemes but state a simplified version that fits our application. In particular, we consider only a single input HSS and incorporate robust decoding in our definition where only s output shares are required for correct decoding. Additionally, we use the notation of s-out-of-n HSS to denote an n-server $(s-1)$-secure HSS according to the definition of Boyle et al.

In Section 8, we utilize an HSS to transform a VIBE scheme into a threshold IBE scheme. In particular, the identity key generation will be executed in a distributed fashion, i.e., the $\mathsf{Extract}$ algorithm of the non-threshold scheme. The homomorphic operations that need to be supported by the HSS depend on the concrete VIBE construction. Since we present a black-box construction in Section 8, we consider a generalized homomorphic secret sharing scheme.

Definition 14 (HSS). *An s-out-of-n* homomorphic secret sharing *scheme HSS for a function $F : (\{0,1\}^*)^2 \to \{0,1\}^*$, or (s,n)-HSS in short, consists of three PPT algorithms:*

1. $\mathsf{Share}(1^\kappa, x)$ *takes as input a security parameter 1^κ and a user input x. It outputs n shares $(x_1, \ldots, x_n)$, where server i gets share x_i.*
2. $\mathsf{Eval}(i, z, x_i)$ *takes as input a server index i, a public input z and the i-th share x_i. It outputs $y_i \in \{0,1\}^*$, corresponding to server i's share of $F(z;x)$.*
3. $\mathsf{Dec}(\{y_i\}_{i\in\mathcal{S}})$ *takes as input a set of output shares and outputs the final output $y \in \{0,1\}^*$.*

We require the following correctness and security properties for every $\kappa \in \mathbb{N}$:

- ***Correctness:*** *For any input $z, x \in \{0,1\}^*$ and any set of shares $(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(1^\kappa, x)$. Let $\forall i \in [n]$ $y_i \leftarrow \mathsf{Eval}(i, z, x_i)$, then for any set $\mathcal{S} \subseteq [n]$ of size s it holds that*
$$\mathsf{Dec}(\{y_i\}_{i\in\mathcal{S}}) = F(z;x).$$
- ***Computational security:*** *Security of an HSS HSS is defined via the experiment $\mathsf{Exp}^{\mathsf{HSS}}_{\mathsf{HSS},\mathcal{A},I}$ where the adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ corrupts a set $\mathcal{M} \subset [n]$ of $s-1$ servers. Then, we require*
$$\left|\Pr[\mathsf{Exp}^{\mathsf{HSS}}_{\mathsf{HSS},\mathcal{A},\mathcal{M}}(\kappa) = 1] - \frac{1}{2}\right| \leq \mathsf{negl}(\kappa),$$
where the experiment is defined as follows.

Experiment $\mathsf{Exp}^{\mathsf{HSS}}_{\mathsf{HSS},\mathcal{A},I}(\kappa)$

$(x_0, x_1) \leftarrow \mathcal{A}_0(1^\kappa)$, *where* $|x_0| = |x_1|$.
$b \in_R \{0,1\}$
$(\hat{x}_1, \ldots, \hat{x}_n) \leftarrow \mathsf{Share}(1^\kappa, x_b)$
$b' \leftarrow \mathcal{A}_1(\{\hat{x}_i\}_{i \in I})$
return $b = b'$

A trivial construction of the Eval algorithm is the identity function. Then, the Dec algorithm first reconstructs x and computes $F(z; x)$ next. As described above, we utilize an HSS to perform the Extract algorithm of a VIBE scheme in a distributed way. In this scenario, the Eval algorithm being the identity function means that the party that should learn the identity key also learns the master secret key. Since this is an undesired effect, we impose an additional requirement on the decoding algorithm. We define a *linear decoding* HSS as a slightly weakening of an additive HSS as defined by Boyle et al. [30]. Intuitively, a linear decoding HSS requires the decoding to be a linear combination of the output shares. In contrast to an additive HSS, a linear decoding HSS enables a decoding algorithm whose output depends on the set of servers from which shares are obtained. In particular, the coefficients depend on the servers' indices that computed the shares. This notion allows to capture any s-out-of-n Shamir's secret sharing.

Definition 15 (Linear Decoding HSS). *An (s, n)-HSS scheme* $\mathsf{HSS} = (\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ *is called* linear decoding *if* Dec *works as follows:*

Let $\{y_1, \ldots, y_n\}$ be a set of output shares. Then, for any set $\mathcal{S} \subseteq [n]$ of size s, there exists a set of s coefficient $\{a_{\mathcal{S},i}\}_{i \in \mathcal{S}}$ such that

$$\mathsf{Dec}(\{y_i\}_{i \in \mathcal{S}}) = \sum_{i \in \mathcal{S}} a_{\mathcal{S},i} \cdot y_i.$$

D.6 Threshold IBE

In this section, we state the formal security games for threshold identity-based encryption schemes (TIBE). The notation for TIBE is given in Section 2.3. We first define the security game for anonymity and security against chosen-identity attacks.

Experiment $\mathsf{Exp}^{\mathsf{A\text{-}T\text{-}CPA}}_{\mathsf{TIBE},\mathcal{A}}(1^\kappa)$

$\mathcal{M} \leftarrow \mathcal{A}_0(1^\kappa)$, where $|\mathcal{M}| < s$
$\alpha, \beta \leftarrow \{0,1\}$
$(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i \in [n]}) \leftarrow \mathsf{Setup}(1^\kappa, s, n)$
$(\mathsf{ID}_0, \mathsf{ID}_1, m_0, m_1) \leftarrow \mathcal{A}_1^{\mathcal{O}(\cdot,\cdot)}(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i \in \mathcal{M}})$
$c^* \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathsf{ID}_\alpha, m_\beta)$
$(\alpha', \beta') \leftarrow \mathcal{A}_2^{\mathcal{O}(\cdot,\cdot)}(c^*)$
return $(\alpha, \beta) = (\alpha', \beta')$

The adversary can use its oracle $\mathcal{O}(\cdot,\cdot)$ to make key generation queries. To do so, the adversary sends (i, id) to $\mathcal{O}$ and receives $(i, \mathsf{ik}_i) \leftarrow \mathsf{ShareKeyGen}(\mathsf{pk}, i, \mathsf{sk}_i, \mathsf{id})$. In the game, we require that ID_0 and ID_1 was not used in any oracle query of $\mathcal{A}_1$ before or after providing the identities and messages.

Next, we define the game for key generation consistency.

$$
\begin{array}{l}
\text{Experiment } \mathsf{Exp}^{\mathsf{KC\text{-}CPA}}_{\mathsf{TIBE},\mathcal{A}}(1^\kappa) \\
\hline
\mathcal{M} \leftarrow \mathcal{A}_0(1^\kappa), \text{ where } |\mathcal{M}| < s \\
(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\kappa, s, n) \\
(\mathsf{ID}, c, \{(i, \mathsf{ik}_i)\}_{i\in\mathcal{S}}, \{(i, \mathsf{ik}'_i)\}_{i\in\mathcal{S}'}) \leftarrow \mathcal{A}_1^{\mathcal{O}(\cdot,\cdot)}(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in\mathcal{M}}), \\
\quad \text{where } \mathcal{S}, \mathcal{S}' \subseteq [n] \wedge |\mathcal{S}| = s = |\mathcal{S}'| \\
\textbf{if } \forall i \in \mathcal{S} : \mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, \mathsf{ID}, i, \mathsf{ik}_i) = \mathsf{true} \\
\quad \wedge \forall i \in \mathcal{S}' : \mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, \mathsf{ID}, i, \mathsf{ik}'_i) = \mathsf{true} \\
\quad \wedge \mathsf{ik} = \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, \mathsf{ID}, \{\mathsf{ik}_i\}_{i\in\mathcal{S}}) \\
\quad \wedge \mathsf{ik}' = \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, \mathsf{ID}, \{\mathsf{ik}'_i\}_{i\in\mathcal{S}'}) \\
\quad \wedge \mathsf{ik}, \mathsf{ik}' \neq \bot \\
\quad \wedge \mathsf{Decrypt}(\mathsf{pk}, \mathsf{ID}, \mathsf{ik}, c) \neq \mathsf{Decrypt}(\mathsf{pk}, \mathsf{ID}, \mathsf{ik}', c) \\
\quad \textbf{return } 1 \\
\textbf{else} \\
\quad \textbf{return } 0
\end{array}
$$

The adversary can use its oracle $\mathcal{O}(\cdot,\cdot)$ in the same way as described above without any restrictions on the queried identities.

Definition 16 (ANON-IND-ID-CPA). *A TIBE scheme* TIBE *is* ANON-IND-ID-CPA *secure if for every* $\kappa, n \in \mathbb{N}$*, every* $1 \leq s \leq n$ *and for every PPT adversary* $\mathcal{A} := (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ *there exist two negligible function* negl_0 *and* negl_1 *such that*

$$
\left|\Pr[\mathsf{Exp}^{\mathsf{A\text{-}T\text{-}CPA}}_{\mathsf{TIBE},\mathcal{A}}(\kappa) = 1] - \frac{1}{4}\right| \leq \mathsf{negl}_0(\kappa) \quad \wedge
$$
$$
\Pr[\mathsf{Exp}^{\mathsf{KC\text{-}CPA}}_{\mathsf{TIBE},\mathcal{A}}(\kappa) = 1] \leq \mathsf{negl}_1(\kappa).
$$

E Construction of Verifiable IBE

In this section, we show how to construct a verifiable identity-based encryption scheme (VIBE) from a standard identity-based encryption scheme and a signature scheme.

Construction 5: VIBE

This construction uses an IBE scheme $\mathsf{IBE} = (\mathsf{IBE.Setup}, \mathsf{IBE.Extract}, \mathsf{IBE.Encrypt}, \mathsf{IBE.Decrypt})$ and a signanture scheme $\mathsf{SIG} = (\mathsf{SIG.KeyGen}, \mathsf{SIG.Sign}, \mathsf{SIG.Verify})$ as building blocks.

$\mathsf{Setup}(1^\kappa)$:

- $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{IBE.Setup}(1^\kappa)$
- $(\mathsf{SigK}, \mathsf{VerK}) \leftarrow \mathsf{SIG.KeyGen}(1^\kappa)$
- **return** $(\mathsf{pk}, \mathsf{VerK}, (\mathsf{msk}, \mathsf{SigK}))$

$\mathsf{Extract}(\mathsf{pk}, (\mathsf{msk}, \mathsf{SigK}), \mathsf{id})$:

- $\mathsf{ik}_{\mathsf{id}} \leftarrow \mathsf{IBE.Extract}(\mathsf{pk}, \mathsf{msk}, \mathsf{id})$
- $\sigma_{\mathsf{id}} \leftarrow \mathsf{SIG.Sign}(\mathsf{SigK}, \mathsf{ik}_{\mathsf{id}})$
- **return** $(\mathsf{ik}_{\mathsf{id}}, \sigma_{\mathsf{id}})$

$\mathsf{Verify}(\mathsf{pk}, \mathsf{VerK}, \mathsf{id}, \mathsf{ik}_{\mathsf{id}}, \sigma_{\mathsf{id}})$:

- **return** $\mathsf{SIG.Verify}(\mathsf{VerK}, \mathsf{ik}_{\mathsf{id}}, \sigma_{\mathsf{id}})$

$\mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, m)$:

- $\mathsf{IBE.Encrypt}(\mathsf{pk}, \mathsf{id}, m)$

$\mathsf{Decrypt}(\mathsf{pk}, \mathsf{id}, \mathsf{ik}_{\mathsf{id}}, c)$:

- $\mathsf{IBE.Decrypt}(\mathsf{pk}, c, \mathsf{ik}_{\mathsf{id}})$

Let VIBE be the construction presented above. The correctness property of VIBE follows directly from the consistency property of IBE and the verifiability property follows from the consistency property of SIG. Moreover, given SIG is existentially unforgeable against chosen-message attacks, one can show via a reduction that VIBE satisfies soundness. Finally, if IBE is ANON-IND-ID-CPA secure, it can be shown again via reduction that the VIBE construction is ANON-IND-ID-CPA secure as well.

F Security Proof: Oblivious TTBE from Anonymous TIBE

This section presents the security proof for Construction 3 in Section 7; the construction of an oblivious tag-based threshold encryption scheme from an anonymous threshold identity-based encryption scheme, a collision-resistant hash function and one-time signatures. Formally, we prove Theorem 3.

Proof. We show security via reductions to the underlying primitives, TIBE, OTS, and H. We start by showing oblivious indistinguishable messages under chosen-ciphertext attacks. For simplicity, we do not deal with the *bad events* of forged signatures and collisions in the hash functions via dedicated game-hops, but incorporate those bad events into the reduction to the TIBE scheme.

In particular, we construct a TIBE-adversary $\mathcal{D}$ playing the experiment $\mathsf{Exp}^{\mathsf{A\text{-}T\text{-}CPA}}_{\mathsf{TIBE},\mathcal{D}}$ with a challenger $\mathcal{C}$ that makes black-box use of a O-TTBE adversary $\mathcal{A}$. In this experiment, $\mathcal{D}$ has access to an identity key oracle $\mathcal{O}$ as defined in Section 2.3. We show that $\mathcal{A}$ can only have a non-negligible advantage in game $\mathsf{Exp}^{\mathsf{O\text{-}CCA}}_{\mathsf{TTBE}_{\mathsf{IBE}},\mathcal{A}}$,

if $\mathcal{D}$ is able to find a hash collision, forge a signature or win the TIBE-game $\mathsf{Exp}^{\text{A-T-CPA}}_{\mathsf{TIBE},\mathcal{D}}$ with non-negligible advantage.

For every adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ on the security game $\mathsf{Exp}^{\text{O-CCA}}_{\mathsf{TTBE}_{\mathsf{IBE}},\mathcal{A}}$, we define the TIBE-adversary $\mathcal{D}$ as follows:

- When invoked with 1^κ, $\mathcal{D}$ initializes sets $\mathcal{I} := \emptyset$ and $\mathcal{V} := \emptyset$, and returns $(\{i_1, \ldots, i_{s-1}\}) \leftarrow \mathcal{A}_0(1^\kappa)$ to $\mathcal{C}$.
- When receiving $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_{i_1}, \ldots \mathsf{sk}_{i_{s-1}}\})$, $\mathcal{D}$ calls $(t_0, t_1, m_0, m_1) \leftarrow \mathcal{A}_1(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_{i_1}, \ldots \mathsf{sk}_{i_{s-1}}\})$.
- During the execution of $\mathcal{A}_1$, $\mathcal{D}$ answers decryption queries of the form $c = (i, (c_0, \mathsf{VerK}, \sigma), t)$ from $\mathcal{A}$ as follows:
  - If $\mathsf{OTS.Verify}(\mathsf{VerK}, \sigma, c_0) = \mathsf{false}$, $\mathcal{D}$ returns $(i, \perp)$.
  - Otherwise, $\mathcal{D}$ sets $\mathsf{id} := H(t, \mathsf{VerK})$ and queries $\mathcal{C}$'s identity key oracle $\mathcal{O}$ with (id, i) to receive key share (i, ik_i).
  - $\mathcal{D}$ adds $(\mathsf{id}, \mathsf{VerK}, t)$ to $\mathcal{I}$, VerK to $\mathcal{V}$, and returns (i, ik_i).
- $\mathcal{D}$ samples $(\mathsf{SigK}^*, \mathsf{VerK}^*) := \mathsf{OTS.KeyGen}(1^\kappa)$, calculates $\mathsf{ID}_0 := H(t_0, \mathsf{VerK}^*)$ and $\mathsf{ID}_1 := H(t_1, \mathsf{VerK}^*)$ and performs the following checks:
  - If $\mathsf{VerK}^* \in \mathcal{V}$, $\mathcal{D}$ ends the experiment with the bad event (bad-sample) and guesses bits $(0, 0)$
  - If $\mathsf{VerK}^* \notin \mathcal{V}$ and $\exists(\mathsf{id}, \mathsf{VerK}, t) \in \mathcal{I}$ and $j \in \{0, 1\}$ such that $\mathsf{id} = \mathsf{ID}_j$, $\mathcal{D}$ ends the experiment with the bad event $(\mathsf{hash\text{-}collision} : (t, \mathsf{VerK}), (t_j, \mathsf{VerK}^*))$ and guesses bits $(0, 0)$.
- $\mathcal{D}$ sends $(\mathsf{ID}_0, \mathsf{ID}_1, m_0, m_1)$ to $\mathcal{C}$. When receiving c^*, $\mathcal{D}$ calculates $\sigma^* := \mathsf{OTS.Sign}(\mathsf{SigK}, c^*)$, calls $(\alpha^*, \beta^*) \leftarrow \mathcal{A}_2((c^*, \mathsf{VerK}^*, \sigma^*))$ and returns guess bits (α^*, β^*) to $\mathcal{C}$.
  Note that if $\mathcal{C}$ selects challenge bits (α, β), this corresponds to $\mathcal{D}$ selecting challenge bits $(\alpha^{\mathsf{TTBE}}, \beta^{\mathsf{TTBE}}) = (\alpha, \beta)$ in the simulated OTTBE security game.
- During the execution of $\mathcal{A}_2$, $\mathcal{D}$ answers decryption queries as follows:
  - If $\mathsf{OTS.Verify}(\mathsf{VerK}, \sigma, c_0) = \mathsf{false}$, $\mathcal{D}$ returns $(i, \perp)$.
  - If $(c_0, \mathsf{VerK}, \sigma, t) \in \{(c^*, \mathsf{VerK}^*, \sigma^*, t_j)\}_{j \in \{0,1\}}$, $\mathcal{D}$ returns $(i, \perp)$.
  - If $(\mathsf{VerK}, t) \in \{(\mathsf{VerK}^*, t_j)\}_{j \in \{0,1\}}$ but $\sigma \neq \sigma^*$, $\mathcal{D}$ ends the experiment with (forged-sig) and guesses bits $(0, 0)$.
  - If $\exists j \in \{0, 1\}$, s.t. $H(t, \mathsf{VerK}) = H(t_j, \mathsf{VerK}^*)$, $\mathcal{D}$ ends the experiment with $(\mathsf{hash\text{-}collision} : (t, \mathsf{VerK}), (t_j, \mathsf{VerK}^*))$ and guesses bits $(0, 0)$.
  - Otherwise, $\mathcal{D}$ calculates $\mathsf{id} := H(t, \mathsf{VerK})$, queries $\mathcal{O}$ with (id, i) to receive key share (i, ik_i), and returns (i, ik_i) to $\mathcal{A}$. Note that the query to $\mathcal{O}'$ is always valid, as $\mathcal{D}$ does not query with $\mathsf{id} \in \{\mathsf{ID}_j\}_{j \in \{0,1\}}$.

Let us start by assuming that there is no bad event. Observe that in this case it holds that (a) $\mathcal{D}$'s simulation of the challenger in game $\mathsf{Exp}^{\text{O-CCA}}_{\mathsf{TTBE}_{\mathsf{IBE}},\mathcal{A}}$ is perfect and (b) $\mathcal{D}$ wins $\mathsf{Exp}^{\text{A-T-CPA}}_{\mathsf{TIBE},\mathcal{D}}$ iff $\mathcal{A}$ wins $\mathsf{Exp}^{\text{O-CCA}}_{\mathsf{TTBE}_{\mathsf{IBE}},\mathcal{A}}$. Hence, it follows that the difference in $\mathcal{D}$'s advantage, $\mathsf{Adv}_{\mathcal{D}}$, and $\mathcal{A}$'s advantage, $\mathsf{Adv}_{\mathcal{A}}$, in the respective games is smaller than the probability of a bad event $\mathsf{Pr}[\mathsf{bad}]$, i.e.,

$$|\mathsf{Adv}_{\mathcal{D}} - \mathsf{Adv}_{\mathcal{A}}| < \mathsf{Pr}[\mathsf{bad}]$$

which implies that

$$\mathsf{Adv}_{\mathcal{D}} + \mathsf{Pr}[\mathsf{bad}] > \mathsf{Adv}_{\mathcal{A}}.$$

As each bad event terminates the execution of the experiment it follows that the events are exclusive and the probability of the experiment ending with any bad event ($\mathsf{Pr}[\mathsf{bad}]$) is the sum of the probabilities for the experiment ending with a particular bad event.

$$\mathsf{Pr}[\mathsf{bad}] = \mathsf{Pr}[\mathsf{bad\text{-}sample}] + \mathsf{Pr}[\mathsf{hash\text{-}collision}] + \mathsf{Pr}[\mathsf{forged\text{-}sig}]$$

As $\mathcal{D}$ randomly samples the key pair $(\mathsf{SigK}^*, \mathsf{VerK}^*)$ from the key space, the probability is negligible in the security parameter κ for the adversary submitting a decryption query including a verification key $\mathsf{VerK} = \mathsf{VerK}^*$. It follows that

$$\mathsf{Adv}_{\mathcal{D}} + \mathsf{negl}(\kappa) + \mathsf{Pr}[\mathsf{hash\text{-}collision}] + \mathsf{Pr}[\mathsf{forged\text{-}sig}] > \mathsf{Adv}_{\mathcal{A}}$$

Therefore, if $\mathsf{Adv}_{\mathcal{A}}$ is a non-neglgible function, then at least one of $\mathsf{Adv}_{\mathcal{D}}$, $\mathsf{Pr}[\mathsf{hash\text{-}collision}]$, and $\mathsf{Pr}[\mathsf{forged\text{-}sig}]$ must also be non-negligible which contradicts our assumptions.

Next, we show decryption consistency under chosen-ciphertext attacks. We do so via a reduction to the identity key share consistency property of the TIBE scheme. In particular, we construct a TIBE-adversary $\bar{\mathcal{D}}$ playing the experiment $\mathsf{Exp}^{\mathsf{KC\text{-}CPA}}_{\mathsf{TIBE},\bar{\mathcal{D}}}$ with a challenger $\bar{\mathcal{C}}$ that makes black-box use of a O-TTBE adversary $\bar{\mathcal{A}}$. In this experiment, $\bar{\mathcal{D}}$ has again access to an identity key oracle $\mathcal{O}$ as defined in Section 2.3. For every adversary $\bar{\mathcal{A}} = (\bar{\mathcal{A}}_0, \bar{\mathcal{A}}_1)$ on the security game $\mathsf{Exp}^{\mathsf{O\text{-}DC}}_{\mathsf{TTBE}_{\mathsf{IBE}},\bar{\mathcal{A}}}$, we define the TIBE-adversary $\bar{\mathcal{D}}$ as follows:

- When invoked with 1^κ, $\bar{\mathcal{D}}$ returns $(\{i_1, \ldots, i_{s-1}\}) \leftarrow \bar{\mathcal{A}}_0(1^\kappa)$ to $\bar{\mathcal{C}}$.
- When receiving $(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_{i_1}, \ldots \mathsf{sk}_{i_{s-1}}\})$, $\bar{\mathcal{D}}$ calls $(t, (c_0^*, \mathsf{VerK}^*, \cdot), \{(i, d_i)\}_{i\in\mathcal{S}}, \{(i, d_i')\}_{i\in\mathcal{S}'}) \leftarrow \bar{\mathcal{A}}_1(\mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_{i_1}, \ldots \mathsf{sk}_{i_{s-1}}\})$.
- During the execution of $\bar{\mathcal{A}}_1$, $\bar{\mathcal{D}}$ answers decryption queries of the form $(i, c = (c_0, \mathsf{VerK}, \sigma), t)$ from $\bar{\mathcal{A}}$ as follows:
  - If $\mathsf{OTS.Verify}(\mathsf{VerK}, \sigma, c_0) = \mathsf{false}$, $\bar{\mathcal{D}}$ returns $(i, \perp)$.
  - Otherwise, $\bar{\mathcal{D}}$ returns $(i, \mathsf{ik}_i) \leftarrow \mathcal{O}(H(t, \mathsf{VerK}), i)$.
- $\bar{\mathcal{D}}$ calculates $\mathsf{ID}^* := H(t, \mathsf{VerK}^*)$ and sends $(\mathsf{ID}^*, c_0^*, \{(i, d_i)\}_{i\in\mathcal{S}}, \{(i, d_i')\}_{i\in\mathcal{S}'})$ to $\bar{\mathcal{C}}$.

Obviously, $\bar{\mathcal{D}}$ perfectly simulates the game $\mathsf{Exp}^{\mathsf{O\text{-}DC}}_{\mathsf{TTBE}_{\mathsf{IBE}},\bar{\mathcal{A}}}$. We complete the proof by showing that $\bar{\mathcal{D}}$ wins security game $\mathsf{Exp}^{\mathsf{KC\text{-}CPA}}_{\mathsf{TIBE},\bar{\mathcal{D}}}$ if $\bar{\mathcal{A}}$ wins security game $\mathsf{Exp}^{\mathsf{O\text{-}DC}}_{\mathsf{TTBE}_{\mathsf{IBE}},\bar{\mathcal{A}}}$. In order to win, $\bar{\mathcal{A}}$ has to sent a tuple

$$(t, c^* = (c_0^*, \mathsf{VerK}^*, \sigma^*), \{(i, d_i)\}_{i\in\mathcal{S}}, \{(i, d_i')\}_{i\in\mathcal{S}'})$$

for which the following holds:

$$\mathcal{S}, \mathcal{S}' \subseteq [n] \text{ and } |\mathcal{S}| = |\mathcal{S}'| = s \tag{11}$$

$$\begin{aligned} \forall i \in \mathcal{S} :& \mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, t, (i, d_i)) = \mathsf{true} \\ \text{and } \forall i \in \mathcal{S}' :& \mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, c, t, (i, d_i')) = \mathsf{true} \end{aligned} \tag{12}$$

$$\begin{aligned} \perp \neq m \neq m' \neq \perp \text{ for } m :=& \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, t, \{(i, d_i)\}_{i\in\mathcal{S}}) \\ \text{and } m' :=& \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, c, t, \{(i, d_i')\}_{i\in\mathcal{S}'}) \end{aligned} \tag{13}$$

In the concrete scheme $\mathsf{TTBE}_{\mathsf{IBE}}$ this implies that:
From (12):

$$\begin{aligned} &\forall i \in \mathcal{S} : \mathsf{TIBE.ShareVf}(\mathsf{pk}, \mathsf{vk}, H(t, \mathsf{VerK}^*), i, d_i) = \mathsf{true} \\ &\text{and } \forall i \in \mathcal{S}' : \mathsf{TIBE.ShareVf}(\mathsf{pk}, \mathsf{vk}, H(t, \mathsf{VerK}^*), i, d_i') = \mathsf{true} \end{aligned} \tag{14}$$

From (13):

$$\begin{aligned} &\mathsf{TTBE.Decrypt}(\mathsf{pk}, \mathsf{id}, \mathsf{ik}, c_0^*) \neq \mathsf{TTBE.Decrypt}(\mathsf{pk}, \mathsf{id}, \mathsf{ik}', c_0^*) \\ &\quad \text{and } \mathsf{ik}, \mathsf{ik}' \neq \perp \text{ for } \mathsf{id} = H(t, \mathsf{VerK}^*) \\ &\quad \text{and } \mathsf{ik} = \mathsf{TTBE.Combine}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, \{(i, d_i)\}_{i \in \mathcal{S}}) \neq \perp \\ &\quad \text{and } \mathsf{ik}' = \mathsf{TTBE.Combine}(\mathsf{pk}, \mathsf{vk}, \mathsf{id}, \{(i, d_i')\}_{i \in \mathcal{S}'}) \neq \perp \end{aligned} \tag{15}$$

In the reduction, $\bar{\mathcal{D}}$ sends the tuple $(H(t, \mathsf{VerK}^*), c_0^*, \{(i, d_i)\}_{i \in \mathcal{S}}, \{(i, d_i')\}_{i \in \mathcal{S}'})$. Note, that properties (11), (14), (15), are exactly what is required in order to win the security game $\mathsf{Exp}^{\mathsf{KC\text{-}CPA}}_{\mathsf{TIBE}, \bar{\mathcal{D}}}$. It follows that $\bar{\mathcal{D}}$ wins security game $\mathsf{Exp}^{\mathsf{KC\text{-}CPA}}_{\mathsf{TIBE}, \bar{\mathcal{D}}}$ if $\bar{\mathcal{A}}$ wins security game $\mathsf{Exp}^{\mathsf{O\text{-}DC}}_{\mathsf{TTBE}_{\mathsf{IBE}}, \bar{\mathcal{A}}}$, which implies that $\bar{\mathcal{D}}$'s advantage, $\mathsf{Adv}_{\bar{\mathcal{D}}}$, is larger or equal to $\bar{\mathcal{A}}$'s advantage, $\mathsf{Adv}_{\bar{\mathcal{A}}}$, in the respective games:

$$\mathsf{Adv}_{\bar{\mathcal{D}}} \leq \mathsf{Adv}_{\bar{\mathcal{A}}}$$

Therefore, if $\mathsf{Adv}_{\bar{\mathcal{A}}}$ is a non-neglgible function, then $\mathsf{Adv}_{\bar{\mathcal{D}}}$ must also be non-negligible which contradicts our assumptions. This concludes the security proof.

G Security Proof: Anonymous Threshold IBE

This section presents the security proof for Construction 4 in Section 8; the construction of an anonymous threshold identity-based encryption scheme from an anonymous non-threshold verifiable identity-based encryption scheme (VIBE) and an homomorphic secret sharing scheme with (HSS) with linear decoding. Formally, we prove Theorem 4.

Proof. For proving security, we need to show anonymity and security against chosen-identity attacks as well as key generation consistency. We split the analysis and show that an adversary playing the security games has only negligible advantage over randomly guessing in both games. We start showing anonymity and security against chosen-identity attacks and prove key generation consistency afterwards.

Anonymity and security against chosen-identity attacks. We prove anonymity and security against chosen-identity attacks using a sequence of hybrid games and show that the difference between every two hybrids is negligible via a reduction to the security properties of VIBE and HSS. We eventually end in a hybrid where we can make a statement about the success probability of the adversary.

Hybrid 0: We start with the original ANON-IND-ID-CPA security game for threshold IBE schemes between a challenger $\mathcal{C}$ and an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$.

Upon receiving a set $\mathcal{M}$ of $s-1$ indices of the corrupted parties from $\mathcal{A}_0$, the challenger performs the setup algorithm, i.e., it first computes $(\mathsf{pk}_{\mathsf{VIBE}}, \mathsf{vk}, \mathsf{msk}) \leftarrow \mathsf{VIBE.Setup}(1^\kappa)$ and then $(\mathsf{msk}_1, \ldots, \mathsf{msk}_n) \leftarrow \mathsf{HSS.Share}(1^\kappa, \mathsf{msk})$. Next, the challenger provides $\mathsf{pk} := \mathsf{pk}_{\mathsf{VIBE}}, \mathsf{vk}$ and $\{\mathsf{msk}_i\}_{i\in\mathcal{M}}$ to $\mathcal{A}_1$. Upon receiving a pair (id, i) as a key generation query from $\mathcal{A}_1$, $\mathcal{C}$ answers with (i, y_i), where $y_i \leftarrow \mathsf{HSS.Eval}(i, \mathsf{id}, \mathsf{msk}_i)$. Recall, that the Eval-algorithm computes a share of $y := (\mathsf{ik}, \rho) \leftarrow \mathsf{VIBE.Extract}(\mathsf{pk}_{\mathsf{VIBE}}, \mathsf{msk}, \mathsf{id})$. $\mathcal{A}_1$ outputs $(\mathsf{ID}_0, \mathsf{ID}_2, m_0, m_1)$. The challenger samples two random bits $\alpha, \beta \in_R \{0,1\}$ and computes $c^* \leftarrow \mathsf{VIBE.Encrypt}(\mathsf{pk}_{\mathsf{VIBE}}, \mathsf{ID}_\alpha, m_\beta)$. In the final stage, $\mathcal{A}_2$ gets as input the challenge ciphertext c^* and outputs two bits (α', β'). $\mathcal{A}_2$ can make the same key generation queries as before except for queries (id, i), where $\mathsf{id} \in \{\mathsf{ID}_0, \mathsf{ID}_1\}$, for which the challenger returns $(i, \perp)$. The adversary wins if $(\alpha, \beta) = (\alpha', \beta')$.

Hybrid 1: In Hybrid 1, we slightly modify how the challenger answers the key generation queries (id, i), where $i \notin I$. Instead of computing $y_i \leftarrow \mathsf{HSS.Eval}(i, \mathsf{id}, \mathsf{msk}_i)$ directly, $\mathcal{C}$ first computes $y := (\mathsf{ik}, \rho) \leftarrow \mathsf{VIBE.Extract}(\mathsf{pk}_{\mathsf{VIBE}}, \mathsf{msk}, \mathsf{id})$ and second obtains for all $j \in \mathcal{M}$ $y_i \leftarrow \mathsf{HSS.Eval}(j, \mathsf{id}, \mathsf{msk}_j)$. Then, given the set $\mathcal{S} = \mathcal{M} \cup \{i\}$ of s values, the challenger can interpolate the identity key share y_i for any $i \notin \mathcal{M}$. In particular, since HSS is linear decoding $y_i := \frac{y - \sum_{j\in\mathcal{S}\setminus\{i\}} a_{\mathcal{S},j} \cdot y_j}{a_{\mathcal{S},i}}$, where the coefficients depend only on $\mathcal{S}$ and the specific indices $i \in \mathcal{S}$.

Since we have a linear decoding HSS that satisfies correctness, we can write the decoding $\mathsf{HSS.Dec}(\{y_i\}_{i\in\mathcal{S}})$ for any set $\mathcal{S}$ of s identity key shares as $y := \sum_{i\in\mathcal{S}} a_{\mathcal{S},i} \cdot y_i$. In particular, for the identity key shares $\{y_j\}_{j\in\mathcal{M}}$ obtained as $y_i \leftarrow \mathsf{HSS.Eval}(j, \mathsf{id}, \mathsf{msk}_j)$ and any $i \notin \mathcal{M}$ obtained as $y_i \leftarrow \mathsf{HSS.Eval}(j, \mathsf{id}, \mathsf{msk}_i)$, it must hold that $y := \sum_{j\in\mathcal{M}} a_{\mathcal{S},j} \cdot y_j + a_{\mathcal{S},i} \cdot y_i$. It follows that $y_i := \frac{y - \sum_{j\in\mathcal{S}\setminus\{i\}} a_{\mathcal{S},j} \cdot y_j}{a_{\mathcal{S},i}}$.

The view of the adversary in Hybrid 0 and Hybrid 1 is identical and hence the success probability is the same.

Hybrid 2: In Hybrid 2, we modify the challenger such that it computes a second master secret key $(\cdot, \cdot, \mathsf{msk}') \leftarrow \mathsf{VIBE.Setup}(1^\kappa)$ and a second set of shares $(\mathsf{msk}'_1, \ldots, \mathsf{msk}'_n) \leftarrow \mathsf{HSS.Share}(1^\kappa, \mathsf{msk}')$ during the setup. While $\mathcal{C}$ still provides $\mathsf{pk} := \mathsf{pk}_{\mathsf{VIBE}}$ and vk to $\mathcal{A}_1$, the secret key shares given to $\mathcal{A}_1$ are $\{\mathsf{msk}'_i\}_{i\in\mathcal{M}}$, i.e., from the second set of master secret key shares. In order to make the responses to key generation queries consistent with the shares msk'_i, the challenger acts as in Hybrid 1 but uses the set $\{\mathsf{msk}'_i\}_{i\in\mathcal{M}}$ instead of $\{\mathsf{msk}_i\}_{i\in\mathcal{M}}$. In full detail, $\mathcal{C}$ acts as follows upon receiving a query (id, i). If $i \in \mathcal{M}$, it answers with (i, y_i), where $y_i \leftarrow \mathsf{HSS.Eval}(i, \mathsf{id}, \mathsf{msk}'_i)$. Otherwise, if $i \notin \mathcal{M}$, $\mathcal{C}$ computes $y := (\mathsf{ik}, \rho) \leftarrow \mathsf{VIBE.Extract}(\mathsf{pk}_{\mathsf{IBE}}, \mathsf{msk}, \mathsf{id})$ and for all $j \in \mathcal{M}$ $y_j \leftarrow \mathsf{HSS.Eval}(j, \mathsf{id}, \mathsf{msk}'_j)$, and returns the interpolated identity key share y_i as described above. To summarize, the difference between Hybrid 1 and Hybrid 2 is the usage of the second master secret key msk' for generating the shares of the corrupted parties as well as during the key generation queries.

We show computational indistinguishability between Hybrid 1 and Hybrid 2 via a reduction to the computational security of the HSS scheme HSS. To this end, we assume a distinguisher $\mathcal{D}_1$ that distinguishes with non-negligible probability between Hybrid 1 and Hybrid 2 and construct an adversary $\mathcal{A}_{\mathsf{HSS}}$

playing in game $\mathsf{Exp}^{\mathsf{HSS}}_{\mathsf{HSS},\mathcal{A}_{\mathsf{HSS}},\mathcal{M}}$. $\mathcal{A}_{\mathsf{HSS}}$ acts like the challenger in Hybrid 2, i.e. it generates two master secret keys msk and msk'. It then provides these two values to the HSS security game which randomly shares one of these and returns the shares $\{\hat{\mathsf{msk}}_j\}_{j\in\mathcal{M}}$. $\mathcal{A}_{\mathsf{HSS}}$ forwards $\{\hat{\mathsf{msk}}_j\}_{j\in\mathcal{M}}$ to the adversary playing Hybrid 1 or Hybrid 2. Note that if msk was shared, the set $\{\hat{\mathsf{msk}}_j\}_{j\in\mathcal{M}} = \{\mathsf{msk}_j\}_{j\in\mathcal{M}}$ and if msk' was shared $\{\hat{\mathsf{msk}}_j\}_{j\in\mathcal{M}} = \{\mathsf{msk}'_j\}_{j\in\mathcal{M}}$. The first case is as in Hybrid 1 and the second is as in Hybrid 2. Upon receiving key generation queries, $\mathcal{A}_{\mathsf{HSS}}$ acts like the challenger in Hybrid 1 resp. Hybrid 2 but it uses $\{\hat{\mathsf{msk}}_j\}_{j\in\mathcal{M}}$ for computing $\{\mathsf{ik}_j\}_{j\in\mathcal{M}}$. Again, if msk was shared, the computation is the same as in Hybrid 1, if msk' was shared, the computation equals Hybrid 2. We showed that $\mathcal{A}_{\mathsf{HSS}}$ is able to fully simulate the view of the adversary in Hybrid 1 or Hybrid 2. Hence, assuming there is a distinguisher $\mathcal{D}_1$ that outputs 0 if the view was generated by Hybrid 1 and 1 if it was generated by Hybrid 2, then $\mathcal{A}_{\mathsf{HSS}}$ outputs this bit. If $\mathcal{D}_1$ distinguishs with non-negligible probability, the adversary $\mathcal{A}_{\mathsf{HSS}}$ wins the security game of the HSS scheme with non-negligible probability too. Since this is a contradiction to our assumption of HSS satisfying correctness, we showed that Hybrid 1 and Hybrid 2 are computational indistinguishable.

We finally are in a hybrid, where we can make a statement about the success probability of the adversary. In particular, we make a reduction to the ANON-IND-ID-CPA security of the non-threshold VIBE scheme VIBE. That means, we show how to construct a successful adversary against the VIBE scheme $\mathcal{A}^{\mathsf{VIBE}} = (\mathcal{A}_0^{\mathsf{VIBE}}, \mathcal{A}_1^{\mathsf{VIBE}})$ based on an successful adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ in Hybrid 2.

Upon initialization with input $\mathsf{pk}_{\mathsf{VIBE}}$ and $\mathsf{vk}_{\mathsf{VIBE}}$, $\mathcal{A}_0^{\mathsf{VIBE}}$ acts like the challenger in Hybrid 2, i.e., it computes msk' and calls $\mathcal{A}_1$ on $(\mathsf{pk} := \mathsf{pk}_{\mathsf{IBE}}, \mathsf{vk} := \mathsf{vk}_{\mathsf{VIBE}}, \{\mathsf{msk}'_j\}_{j\in\mathcal{M}})$, where $\mathcal{M}$ is obtained from $\mathcal{A}_0$ before. All key generation queries are answered like the challenger in Hybrid 2 except that $y := (\mathsf{ik}, \rho)$ is obtained from $\mathcal{A}_0^{\mathsf{VIBE}}$'s oracle instead of computing it directly. In this reduction, it is not possible to compute it inside $\mathcal{A}_0^{\mathsf{VIBE}}$ because $\mathcal{A}_0^{\mathsf{VIBE}}$ does not know the master secret key msk. The output $(\mathsf{ID}_0, \mathsf{ID}_1, m_0, m_1)$ of $\mathcal{A}_0$ is directly forwarded to the challenger in the VIBE security game. Next, $\mathcal{A}_1^{\mathsf{VIBE}}$ is called with the challenge ciphertext c^* which is passed to $\mathcal{A}_1$. The oracle queries in this stage are handled like before. Note that it is no problem that $\mathcal{A}_1^{\mathsf{VIBE}}$ is not allowed to query identity keys for ID_0 and ID_1 since $\mathcal{A}_2$ is also not allowed to query any key shares for these two identities. Finally, $\mathcal{A}_1^{\mathsf{VIBE}}$ outputs exactly the two bits returned by $\mathcal{A}_2$.

It is easy to see that success probability of $\mathcal{A}^{\mathsf{VIBE}}$ is the same as the success probability of $\mathcal{A}$. Via proof by contradiction, it follows that $\mathcal{A}$ wins in Hybrid 2 only with probability $\frac{1}{4} + \mathsf{negl}$ and hence, our construction is ANON-IND-ID-CPA secure.

Key generation consistency. Similar to above, we show that the success probability of an adversary $\mathcal{A} := (\mathcal{A}_0, \mathcal{A}_1)$ in the security game $\mathsf{Exp}^{\mathsf{KC\text{-}CPA}}_{\mathsf{TIBE},\mathcal{A}}$ is negligible via a sequence of hybrids.

Hybrid 0: We start in the original security game $\mathsf{Exp}^{\mathsf{KC\text{-}CPA}}_{\mathsf{TIBE},\mathcal{A}}$. Upon intialization, $\mathcal{A}_0$ gives a set of $s-1$ indices denoting the corrupted parties. Next, the challenger $\mathcal{C}$ performs the setup and gives the public key, the verification key and the set of master secret key shares of the corrupted parties to $\mathcal{A}_1$. The oracle queries are answered as in Hybrid 0 of the previous proof, i.e., upon receiving a pair (id, i) from $\mathcal{A}_1$, $\mathcal{C}$ answers with (i, y_i), where $y_i \leftarrow \mathsf{HSS.Eval}(i, \mathsf{id}, \mathsf{msk}_i)$. Finally, $\mathcal{A}_1$ outputs an identity ID, a ciphertext c and two set of each s identity key shares $\{(i, y_i)\}_{i\in\mathcal{S}}$ and $\{(i, y'_i)\}_{i\in\mathcal{S}'}$, where $\mathcal{S}, \mathcal{S}' \subseteq [n]$. The adversary wins the game if the a set of requirements hold. We leave out the checks $\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, \mathsf{ID}, i, y_i)) = \mathsf{true}$ and $\mathsf{ShareVf}(\mathsf{pk}, \mathsf{vk}, \mathsf{ID}, i, y'_i)) = \mathsf{true}$, since the $\mathsf{ShareVf}$ algorithm of the construction always returns true and hence the checks trivially hold. The following checks remain:

(1) for $\mathsf{ik} = \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, \mathsf{ID}, \{y_i\}_{i\in\mathcal{S}})$ and $\mathsf{ik}' = \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, \mathsf{ID}, \{y'_i\}_{i\in\mathcal{S}'})$ it holds that $\mathsf{ik} \neq \perp$ and $\mathsf{ik}' \neq \perp$ and
(2) $\mathsf{Decrypt}(\mathsf{pk}, \mathsf{ID}, \mathsf{ik}, c) \neq \mathsf{Decrypt}(\mathsf{pk}, \mathsf{ID}, \mathsf{ik}', c)$.

If any of these checks fail, the adversary loose the game.

Hybrid 1 and Hybrid 2: We apply the same transformations as in Hybrid 1 and Hybrid 2 of the previous proof. That means, we first answer oracle queries for honest parties via the interpolated value and in the second transformation, we use a fresh master secret key whose shares are given to the adversary. The oracle queries are answered in the same way as described in the previous proof. Indistinguishability between Hybrid 0 and Hybrid 1 as well as between Hybrid 1 and Hybrid 2 can be shown via reduction to the correctness and security of HSS, respectively. We elaborate the details in the previous proof. We end up in a hybrid where the challenger $\mathcal{C}$ uses the real master secret key only for computing the output of $\mathsf{Extract}(\mathsf{pk}, \mathsf{msk}, \mathsf{id})$ upon oracle queries.

We conclude our proof by making a reduction to the soundness property of the VIBE scheme. We take the adversary $\mathcal{A}^2 := (\mathcal{A}^2_0, \mathcal{A}^2_1)$ in Hybrid 2 and construct an adversary $\mathcal{A}$ for the soundness game $\mathsf{Exp}^{\mathsf{SOUND}}_{\mathsf{VIBE},\mathcal{A}}$. Upon being initialized with $\mathsf{pk}_{\mathsf{VIBE}}$ and $\mathsf{vk}_{\mathsf{VIBE}}$, $\mathcal{A}$ acts like the challenger in Hybrid 2. This means, $\mathcal{A}$ computes a fresh master secret key and computes shares $\{\hat{\mathsf{msk}}_i\}_{i\in[n]}\}$ of it. Next, $\mathcal{A}$ calls $\mathcal{A}^2_1(\mathsf{pk}_{\mathsf{VIBE}}, \mathsf{vk}_{\mathsf{VIBE}}, \{\mathsf{msk}_i\}_{i\in\mathcal{M}})$, where $\mathcal{M}$ was obtained from $\mathcal{A}^2_0$. Every key share oracle query by $\mathcal{A}^2_1$ is answered as described above, i.e., $\mathcal{A}$ asks its oracle $\mathcal{O}$ on the same identity and then the required share is interpolated while using $\{\mathsf{msk}_i\}_{i\in\mathcal{M}}$. This is identical to the behavior of the challenger in Hybrid 2. Upon receiving the output $(\mathsf{ID}, c, \{(i, \mathsf{ik}_i)\}_{i\in\mathcal{S}}, \{(i, \mathsf{ik}'_i)\}_{i\in\mathcal{S}'})$ from $\mathcal{A}^2_1$, $\mathcal{A}$ uses the $\mathsf{HSS.Dec}$ algorithm to obtain both pairs $(\mathsf{ik}_{\mathsf{ID}}, \rho_{\mathsf{ID}})$ and $(\mathsf{ik}'_{\mathsf{ID}}, \rho'_{\mathsf{ID}})$. These two pairs together with ID and c is returned to the challenger of the soundness game.

We note that adversary $\mathcal{A}$ simulates the view of $\mathcal{A}^2$ as in Hybrid 2. It remains to show that $\mathcal{A}$ wins with the same probability as $\mathcal{A}^2$.

Note that, if $\mathcal{A}^2$ wins, checks (1) and (2) hold. (1) implies that $\mathsf{VIBE.Verify}(\mathsf{pk}, \mathsf{vk}, \mathsf{ID}, \mathsf{ik}_I D, \rho_{\mathsf{ID}}) = 1$ and $\mathsf{VIBE.Verify}(\mathsf{pk}, \mathsf{vk}, \mathsf{ID}, \mathsf{ik}'_I D, \rho'_{\mathsf{ID}}) = 1$ because otherwise the $\mathsf{Combine}$ algorithm would return $\perp$. Check (2) directly implies $\mathsf{VIBE.Decrypt}(\mathsf{pk}, \mathsf{ik}_{\mathsf{ID}}, c) \neq \mathsf{VIBE.Decrypt}(\mathsf{pk}, \mathsf{ik}'_{\mathsf{ID}}, c)$. It follows that $\mathcal{A}$

wins in the soundness game if $\mathcal{A}^2$ wins in Hybrid 2. This leads to a contradiction to the assumption that VIBE satisfies the soundness property. Hence, $\mathcal{A}^2$ in Hybrid 2 has only negligible success probability.

H Concrete Anonymous TIBE Construction

In this section, we discuss how concrete anonymous TIBE scheme can be constructed. To this end, we analyze the anonymous non-threshold IBE scheme from Boyen and Waters [16]. For the sake of completeness, we state the Boyen-Waters construction in Appendix I. The naive approach to transform the scheme into a threshold variant is to use Shamir's secret sharing to share all components of the master secret key over multiple parties and executing the algorithm generating the identity keys, the Extract algorithm, in a distributed way. The problem using this approach is that the Extract algorithm of the original scheme performs several multiplications of master secret key components. Suppose these values are shared using Shamir's secret sharing, then each server gets a share of a polynomial of degree $s-1$. Performing a single multiplications of the shares results in a shared polynomial of degree $2(s-1)$; the degree increases with every further multiplication by a factor of 2. While there are at most two multiplications of secret key components in the Extract algorithm, there is another multiplication with randomly sampled values. Security of the scheme relies on the fact that these random values, which are sampled per identity key, remain hidden. Hence, it is necessary that each decryption server uses a secret share of the random values. Thus, multiplication with a shared random value increases the degree again by factor 2. Since the total number of decryption servers needs to be bigger as the degree of the shared polynomial, every multiplication over shared values effectively lowers the number of corrupted parties that can be tolerated. In case we require robustness, a property that states that the honest parties alone are sufficient to execute decryption, the degree of the shared polynomial needs to be even smaller than the number of honest committee members. Hence, we propose a number of steps to transform the scheme into a threshold variant while keeping the security threshold as high as possible.

As in the naive approach, we start by secret sharing the master secret key components, i.e., ω, t_1, t_2, t_3 and t_4, using Shamir's secret sharing. In addition, we compute additive shares of the multiplications of secret key components, i.e., of $t_1t_2, t_3t_4, \omega t_1$ and ωt_2, in the Setup algorithm using Shamir's secret sharing. Now, the computations performed in the Extract algorithm contain at most a single multiplication between a component of the shared master secret key and a random value sampled in a decentralized way. That means, we already reduced the blow up in the polynomial degree to a factor of 2. At this point, the identity key shares constitute shares of a polynomial of degree $2(s-1)$. In order to guarantee that enough shares are available, we need that $n > 2s-2$ (for robustness $n - s > 2s - s$). This holds for a threshold of $s \leq \frac{n}{2}$ (for robustness $\leq \frac{n}{3}$).

Alternatively, the Setup algorithm can pre-compute the multiplications by sampling a set of random pairs $\{(r_{i,1}, r_{i,2})\}_{i\in[K]}$ for some large $K \in \mathbb{N}$ and

generating shares of the multiplications $r_{i,1}t_1t_2$, $r_{i,2}t_3t_4$, $r_{i,1}t_2$, $r_{i,1}t_1$, $r_{i,2}t_4$ and $r_{i,2}t_3$. The master secret key shares are extended by the new multiplication shares. The committee, then uses one of these pre-computations in the Extract algorithm which allows the generation of identity keys without any multiplications of shared values. Hence, we can set the security threshold to the optimum of $s < n$ (for robustness $\frac{n}{2}$). However, it is necessary that each pre-computation is used at most once. This means that the size of the master secret keys increases with the number of identity keys K that should be extracted later on and K needs to be fixed during setup. The later can be circumvented by a refreshing phase that precomputes new multiplication shares if required.

I Boyen Waters Anonymous IBE Scheme

In this section, we present the Boyen and Waters anonymous identity-based encryption (IBE) scheme [16]. The scheme is semantically secure and anonymous under the DBDH and the decisional linear assumption (DLIN). A detailed description and security analysis can be found in [16]. For the sake of completeness, we first present the decisional linear assumption and then, outline the construction.

Definition 17 (DLIN). *The Decision Linear assumption (DLIN) states that for every algorithm $\mathcal{D}$ running in time polynomial in security parameter κ it holds that*

$$\left|\Pr[\mathcal{D}(u, v, h, u^a, v^b, h^{a+b})] - \Pr[\mathcal{D}(u, v, h, u^a, v^b, h^c)]\right| \leq \mathsf{negl}(\kappa) \qquad (16)$$

where $\mathbb{G}$ is a cyclic group of prime order q, u, v, h are random generators of $\mathbb{G}$, and $a, b, c \in_R \mathbb{Z}_q$. The randomness is taken over the random choices of the group elements $u, v, h,$, the values a, b, c, and the random bits of $\mathcal{D}$.

Construction 6: BW-IBE

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups of prime order q and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ a bilinear mapping.

Setup(1^κ):

- choose random generator $g \in_R \mathbb{G}$ and random group elements $g_0, g_1 \in_R \mathbb{G}$
- sample random exponents $\omega, t_1, t_2, t_3, t_4 \in \mathbb{Z}_q$
- $\mathsf{msk} = (\omega, t_1, t_2, t_3, t_4)$
- $\mathsf{pk} = (\Omega = e(g, g)^{t_1 t_2 \omega}, g, g_0, g_1, v_1 = g^{t_1}, v_2 = g^{t_2}, v_3 = g^{t_3}, v_4 = g^{t_4})$
- **return** $(\mathsf{pk}, \mathsf{msk})$

Extract($\mathsf{pk}, \mathsf{msk}, \mathsf{id}$):

- $r_1, r_2 \in_R \mathbb{Z}_q$

– **return** $\mathsf{ik}_{\mathsf{id}} = (d_0, d_1, d_2, d_3, d_4)$ where

$$
\begin{aligned}
d_0 &= g^{r_1 t_1 t_2 + r_2 t_3 t_4} \\
d_1 &= g^{-\omega t_2} (g_0 g_1^{\mathsf{id}})^{-r_1 t_2} \\
d_2 &= g^{-\omega t_1} (g_0 g_1^{\mathsf{id}})^{-r_1 t_1} \\
d_3 &= (g_0 g_1^{\mathsf{id}})^{-r_2 t_4} \\
d_4 &= (g_0 g_1^{\mathsf{id}})^{-r_2 t_3}
\end{aligned}
$$

$\mathsf{Encrypt}(\mathsf{pk}, \mathsf{id}, m)$**:**

– $s, s_1, s_2 \in \mathbb{Z}_q$
– **return** $c = (c', c_0, c_1, c_2, c_3, c_4) = (\omega^s m, (g_0 g_1^{\mathsf{id}})^s, v_1^{s-s_1}, v_2^{s_1}, v_3^{s-s_2}, v_4^{s_2})$

$\mathsf{Decrypt}(\mathsf{pk}, c, \mathsf{ik}_{\mathsf{id}})$**:**

– **return** $m = c \cdot e(c_0, d_0) \cdot e(c_1, d_1) \cdot e(c_2, d_2) \cdot e(c_3, d_3) \cdot e(c_4, d_4)$

J Further Applications

In this section, we present further use-cases for our SO-TWE and T-TTBE primitives.

J.1 Time-lock encryption with a hidden release time.

Time-lock encryption, as introduced by Rivest et al. [31], allows a party to encrypt a message in a way that it can only be decrypted after a certain deadline. Liu et al. [6] propose an extension based on witness encryption in which the parties interested in decryption do not have to invest a large amount of sequential computation. However, being based on standard witness encryption, the deadline respectively the lock period is public. This limitation can be overcome by our primitives. In particular, we can construct a time-lock encryption scheme for which the release time remains hidden until successful decryption. Suppose there is a committee holding the secret key shares of an O-TTBE scheme with public key pk. An encrypting party that wants to encrypt a message m such that it is released at a specific date d, creates a ciphertext $\mathsf{OTTBE.Encrypt}(\mathsf{pk}, d, m)$ and sends it to the decryption committee with the instruction to decrypt every day with the current date as tag. It follows from the indistinguishable message and tag property of the OTTBE scheme that both the message and the realease time are hidden until successful decryption. In case the decryption committee does not have access to a source of time, we can still realize this notion of time-lock encryption with hidden release time by making use of SO-TWE in combination with either a trusted time provider or a blockchain. The encrypting party will encrypt the message such that the witness is a signature of the trusted time provider on a date larger than d or a valid blockchain that is (a) a successor of

the blockchain state known during encryption and (b) has a confirmed block with timestamp larger than d. Parties that want to decrypt, retrieve the corresponding witness from the time provider respectively the blockchain network and submit the witness together with the current date to the decryption committee, every day.

J.2 Dead-Man Switch for an Unknown Party

A dead-man switch is designed to trigger a particular action once a linked human becomes incapacitated. In the digital world, a dead-man switch typically releases private data that is valuable by itself or can be used to trigger further actions. The release condition can be diverse, e.g., not logging in to a particular service for a certain duration. A dead-man switch is for example interesting for whisteblowers to publish evidence of ongoing investigations in case they get harmed during said investigation. On one hand, a dead-man switch can be used to deter from actively induced harm to the individual. On the other hand, in cases where outsiders assume that the whisteblower will certainly publish the results of the ongoing investigation once finished, it is better not to draw attention by publicly announcing the dead-man switch. This privacy property can be achieved using our notion of SO-TWE. The journalist encrypts the sensible data with her name as statement and an official death certificate as required witness and uploads it to a public database (using an anonymous channel). The journalists' association submits the names and death certificates of journalists that decease from non-natural causes to the decryption committee together with all unopened dead-man switches. Again, the message and statement indstinguishability ensures that the identity of the journalist as well as the encrypted data remain hidden until decryption which can only happen after the individual deceased.

J.3 Charity Lottery

Consider a scenario where a benefactor aims to sell a digital prize (coins or tokens) via a lottery in such a way that all the money raised is donated to selected charity organizations. By making use of O-TTBE, the benefactor can realize such a lottery without the requirement of actively participating in the lottery process as well as she can also allow customers to choose the subset of charities they want to support. To do so, the benefactor generates fresh O-TTBE decryption keys and distributes them to the charity organization. Then, she encrypts the authorization to withdraw the prize using a random tag and publishes the ciphertext. By publishing a set of possible tags, the benefactor can adjust the win rate of the lottery. Once the lottery has been set up, customers can buy decryption shares for self-selected tags from the charity organizations, hence, directly donating the money to the organizations of their choice. The party that guesses the tag correctly wins the lottery and gets the prize.

J.4 Puzzle Prizes

Consider a puzzle whose solutions are not publicly verifiable but require a game master to verify, e.g., a quiz. In scenarios, in which the puzzle is not expected to be solved within a short time period, it is desirable not to require the game master to be online for the whole duration of the game. This requirement can be circumvented by using O-TTBE in combination with a public decryption committee. The game master encrypts the prize for the correct answer with the correct solution as the tag using O-TTBE. Parties can request decryptions with their solution attempts as tag, and, if successful, obtain the prize. This use-case can further be extended with smart contracts, e.g., to handle fees for solution attempts that are (partially) included into the prize pool.