

# BIP32-Compatible Threshold Wallets

Poulami Das<sup>1</sup>      Andreas Erwig<sup>2</sup>      Sebastian Faust<sup>3</sup>  
Philipp-Florens Lehwald<sup>3</sup>      Julian Loss<sup>4</sup>      Ziyang Qu<sup>3</sup>      Siavash Riahi<sup>3</sup>

<sup>1</sup> Least Authority, Germany

poulamidas22@gmail.com

<sup>2</sup> TU Darmstadt, Germany

aerwig@posteo.de

<sup>3</sup> TU Darmstadt, Germany

firstname.lastname@tu-darmstadt.de

<sup>4</sup> CISPA Helmholtz Center of Information Security, Germany

lossjulian@gmail.com

## Abstract

Cryptographic wallets are an essential tool to securely store and maintain users' secret keys and consequently their funds in Blockchain networks. A compelling approach to construct such wallets is to share the user's secret key among several devices, such that an adversary must corrupt multiple machines to extract the entire secret key. Indeed, many leading cryptocurrency companies such as Coinbase, Binance, or ZenGo have started offering such distributed wallets to their customers. An important feature of a cryptographic wallet is its compatibility with the so-called BIP32 specification, the most widely adopted standard for cryptographic wallets. Essentially, BIP32 specifies the notion of a hierarchical deterministic wallet, which allows to create a key hierarchy in a deterministic fashion. Unfortunately, despite significant interest, no practically efficient solution for a fully distributed wallet scheme, that also follows the BIP32 standard, exists.

In this work, we show the first *concretely efficient* construction of a fully distributed wallet that is compliant with the BIP32 standard. To this end, we first provide a game-based notion of threshold signatures with rerandomizable keys and show an instantiation via the Gennaro and Goldfeder threshold ECDSA scheme (CCS'18). We then observe that one of BIP32's key derivation mechanisms, the so-called hardened derivation, cannot efficiently be translated to the threshold setting. Instead, we devise a novel and efficient hardened derivation mechanism for the threshold setting that satisfies the same properties as the original mechanism as specified by BIP32. As a final contribution, we evaluate our solution with respect to its running time and communication cost.

## 1 Introduction

Blockchain technologies have gained huge popularity in the past few years, providing a new decentralized mechanism to process payments without relying on a centralized authority. The main cryptographic building block in virtually all Blockchains is a digital signature scheme, allowing Blockchain network parties to authenticate transactions. As an example, if Alice wishes to make a payment to Bob via the Blockchain, she can sign a transaction specifying her address, which is derived from her signing public key  $\mathbf{pk}_A$ , Bob's address, derived from his public key  $\mathbf{pk}_B$  and the amount being spent. Alice can then sign the transaction  $\text{tx} := (\mathbf{pk}_A \text{ pays } c \text{ coins to } \mathbf{pk}_B)$  using her secret key  $\mathbf{sk}_A$  and sends " $(\text{tx}, \sigma)$ " to the blockchain, where  $\sigma$  is Alice's signature on  $\text{tx}$  under her secret key  $\mathbf{sk}_A$ . Observe that knowledge of Alice's secret key allows to spend all of her funds, which makes it crucial for users in a Blockchain network to protect their secret keys from attackers. To this end, users employ a so-called *cryptographic wallet*, which is typically a piece of software or special-purpose hardware that securely stores and maintains the user's signing keys. Unfortunately, designing and implementing such cryptographic wallets securely has proven inherently difficult, as demonstrated by countless devastating attacks resulting in the theft of several billions worth of USD [Ske18, Blo18, Bit18].

Motivated by these attacks many leading cryptocurrency companies such as Coinbase<sup>1</sup>, Binance<sup>2</sup>, or ZenGo<sup>3</sup> develop distributed wallets, where the user’s signing key is split among several devices. That is, each device only stores a share of the user’s signing key, such that an attacker would have to corrupt several devices to steal the user’s funds. To sign a transaction, the devices that store the shares of the signing key run a cryptographic protocol – a so-called *threshold signature scheme*. While threshold cryptography was invented more than 30 years ago [DF90], its application in cryptocurrency wallets has spurred new interest both by industry and academia in designing novel schemes that offer better efficiency and advanced features (see e.g., [DMZ<sup>+</sup>21, CCL<sup>+</sup>20, CCL<sup>+</sup>21] and many more).

An important feature of cryptocurrency wallets is their compatibility with a standard for *hierarchical deterministic wallets* called *Bitcoin Improvement Proposal 32 (BIP32)* [Wik18]. Initially put forth by the cryptocurrency community in 2012, it is nowadays widely deployed by major wallets such as Electrum [Ele13], Ledger [Led14], or Trezor [Tre14]. While the BIP32 standard has recently formally been analyzed [DEF<sup>+</sup>21], an extension to the setting of distributed wallets is missing. This is unfortunate as BIP32 offers important features that provide additional privacy and security (see below for background on BIP32 and its features). While in principle the BIP32 standard can be distributed using techniques from generic multiparty computation (MPC) [GMW87, CCD88, EGPS22, HHPV21, BDST20], this will likely lead to significant overheads in terms of communication and computation; thus, it is unlikely that such techniques are going to be used at massive scale in practice. Two recent works [Yeh23, CHL23] study the use of BIP32 in the two-party setting, i.e., where the secret key is shared between two parties; however, to date, no fully distributed solution for many parties exists. Therefore, in this work, we study the following question:

*Can we construct a concretely efficient, fully distributed wallet scheme that is compatible with the BIP32 standard?*

## 1.1 Background on the BIP32 Standard

A hierarchical deterministic wallet scheme, as specified by the BIP32 standard, is organized in a tree-like structure. The *root wallet* deterministically derives child wallets which in turn deterministically derive further child wallets. We illustrate the high-level structure of a BIP32 wallet in Figure 1. As an illustrative example, one may think of the key tree structure representing an organization, where in addition to managing its own funds, a parent wallet can create wallets for members of the organization at a lower level in the hierarchy (e.g., a group leader creating wallets for its group members).

Each wallet in the tree is identified by an ID and consists of a signing secret and public key pair  $(\text{sk}_{\text{ID}}, \text{pk}_{\text{ID}})$  as well as a so-called *chaincode*  $\text{ch}_{\text{ID}}$ . These keys can then be used to send and receive funds and together with the chaincode derive child wallets. BIP32 considers two types of child wallet key derivation: *hardened* and *non-hardened*. In the non-hardened mode, a parent wallet identified by ID derives the key pair of a non-hardened child wallet with identifier  $\text{ID}'$  as  $\text{sk}_{\text{ID}'} \leftarrow \text{sk}_{\text{ID}} + \text{H}(\text{pk}_{\text{ID}}, \text{ch}_{\text{ID}}, \text{ID}')$  and  $\text{pk}_{\text{ID}'} \leftarrow \text{pk}_{\text{ID}} \cdot g^{\text{H}(\text{pk}_{\text{ID}}, \text{ch}_{\text{ID}}, \text{ID}')}$  where  $\text{H}$  is a cryptographic hash function. In the hardened mode, keys are derived in exactly the same way, except that the hash function takes as input the parent’s secret key  $\text{sk}_{\text{ID}}$  instead of its public key  $\text{pk}_{\text{ID}}$ . The main difference between the hardened and non-hardened mode is the level of security that they can offer, which we will discuss in a moment.

Das et al. [DEF<sup>+</sup>21] were the first to formally model and analyze hierarchical deterministic wallets according to BIP32. Hierarchical deterministic wallets have to satisfy two security guarantees. *Unlinkability* guarantees that signatures created by wallets derived from the same root are unlinkable as long as the chaincode remains hidden. *Unforgeability* ensures that even if all public keys and the chaincodes leak, it is hard to forge signatures. While unlinkability is typically easy to show, proving unforgeability is more challenging since by revealing the chaincode the adversary gets to know a relation between all secret keys. In addition, the hardened key derivation allows us to further strengthen the basic unforgeability notion. To illustrate this, consider first the non-hardened key derivation. Here, if the non-hardened secret key  $\text{sk}_{\text{ID}'}$  is leaked to an adversary that knows the parent chaincode  $\text{ch}_{\text{ID}}$ , the adversary can compute the parent secret key as  $\text{sk}_{\text{ID}} \leftarrow \text{sk}_{\text{ID}'} - \text{H}(\text{pk}_{\text{ID}}, \text{ch}_{\text{ID}}, \text{ID}')$ .<sup>4</sup> This attack does not work on hardened secret keys

<sup>1</sup><https://www.coinbase.com>

<sup>2</sup><https://www.binance.com>

<sup>3</sup><https://zengo.com>

<sup>4</sup>This vulnerability of BIP32 is also known as the *privilege escalation attack*.

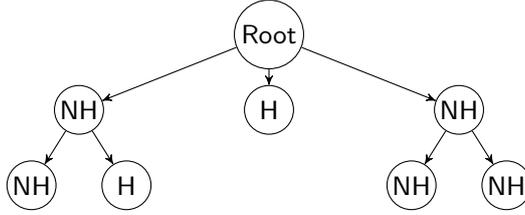


Figure 1: A hierarchical deterministic wallet scheme according to the BIP32 standard [Wik18]. H and NH identifies hardened and non-hardened nodes respectively.

as it requires knowledge of the parent secret key. One may ask why in practice a user does not derive all keys using the hardened derivation. The reason is usability, as a non-hardened public key can be derived by any party knowing  $pk_{ID}$  and  $ch_{ID}$ , i.e., without knowledge of the parent secret key, which is not the case for the hardened derivation. A key derivation where the secret key is not required for deriving the child public key is particularly useful in a setting where the secret key is stored in an offline device and therefore not accessible.

To formally analyze the security of hierarchical deterministic wallets, Das et al. [DEF+21] assume in their security model that non-hardened wallets are implemented in the hot/cold wallet setting. In this setting, a non-hardened wallet consists of one online device (the hot wallet), which stores the public key and the chaincode, and one offline device (the cold wallet), which stores the secret key and the chaincode. It is then assumed that no adversary can corrupt the cold wallet (and therefore does not learn the secret key) since it is mostly offline. With this assumption, the authors can avoid the above attack on non-hardened secret keys. Unfortunately, this assumption is rather idealized and often does not hold in practice. For instance, an adversary with physical access to the cold wallet device might be able to extract the secret key from it. Even worse, in many practical use cases cold wallets simply cannot be deployed. Consider for example a wallet holding funds of a node in the Lightning network or a wallet controlling stake in a Proof of Stake protocol. All these use cases inherently require that secret keys are permanently kept in hot storage, thus making them susceptible to attacks.

Conveniently, a distributed BIP32-compatible wallet can help to overcome the requirement of storing secret keys of non-hardened nodes in a cold wallet. More precisely, if we distribute the secret key of a non-hardened node among several (online) devices rather than implementing the node in the hot/cold setting, an adversary can only learn the node’s secret key if it successfully corrupts several devices at once. Assuming that an adversary cannot hack into several devices, which are potentially maintained and protected by different wallet provider companies, seems to be a significantly more realistic assumption than to assume that a single user-maintained offline device is incorruptible. Moreover, if the distributed wallet is sufficiently efficient, the use-cases mentioned above can benefit from our solution.

Unfortunately, constructing efficient distributed BIP32-compatible wallet is challenging, in particular due to the hardened key derivation. Recall that to derive a hardened key, we need to compute  $H(sk_{ID}, ch_{ID}, ID')$ , i.e., a hash function evaluation that receives the parent secret key as input. However, in the distributed setting, the secret key is shared among several devices, i.e., the hash function must be computed in a distributed manner. While this can be done via generic MPC techniques, such techniques are inefficient, in particular when the number of devices holding key shares grows.

## 1.2 Our Contribution

In this work, we provide the first efficient fully distributed and BIP32-compatible wallet solution. To this end, we design a novel hardened key derivation mechanism that relies on a threshold verifiable random function (TVRF) [Dod03], which offers orders of magnitude better efficiency (both in terms of computation as well as communication) than a naïve solution using generic MPC techniques. We combine the distributed key derivation with a threshold ECDSA signature scheme. Our choice for ECDSA as compared to other signature schemes is based on the following two reasons: 1) ECDSA is used by many popular cryptocurrencies such as Bitcoin or Ethereum and 2) the BIP32 standard is defined w.r.t. ECDSA and since our aim is to build a solution that is compatible with BIP32, ECDSA is a natural choice. We summarize our contribution below:

1. We introduce *threshold signature schemes with rerandomizable keys*, a novel cryptographic primitive that is essential for the construction of distributed BIP32 wallets. In a nutshell, this primitive is a threshold signature scheme, which allows the parties to non-interactively rerandomize their signing secret key shares such that the resulting shares again form a valid sharing of the scheme’s full secret key. This primitive may be of independent interest.
2. We show how we can translate the (non-)hardened derivation mechanism as specified by BIP32 to the threshold setting. Since the hardened derivation requires the distributed computation of a hash function (which is highly inefficient), we devise a tailored protocol for distributing the hardened key derivation which is efficient even for a large set of devices.
3. To substantiate our claims about the efficiency of our hardened derivation mechanism, we provide an implementation and evaluation of our mechanism and compare it to a solution using generic MPC techniques. Indeed, our evaluation shows that our distributed key derivation is more than 3.000 times more efficient than the generic MPC technique.

### 1.3 Related Work

**Cryptographic Wallets.** There has been a plethora of works on cryptographic wallets such as [MPs19, AGKK19, KMOS21]. We focus on hierarchical deterministic wallets, which have been extensively researched in the past. Gutoski and Stebila [GS15] introduced a hierarchical deterministic wallet scheme that, however, deviates from the BIP32 standard. Later, Das et al. [DFL19] gave the first formal analysis of deterministic wallets in the hot/cold setting and provided a construction based on multiplicatively rerandomizable ECDSA. The model of deterministic wallets by Das et al. [DFL19] has been extended to the post-quantum setting by Alkadri et al. [ADE+20]. Luzio et al. [LFA20] presented a hierarchical deterministic wallet scheme, which is however not compatible with Bitcoin. The most relevant work for our results is the paper by Das et al. [DEF+21] which analyzes the security of hierarchical deterministic wallets that comply with the BIP32 standard. As mentioned previously, Das et al. show that such wallets can be constructed generically from signature schemes with rerandomizable keys. Recently, Yin et al. [YLY+22] proposed a model for hierarchical deterministic wallets supporting stealth addresses. However, their construction is incompatible with Bitcoin as it relies on bilinear maps. Erwig and Riahi [ER] recently proposed deterministic wallets with support for adaptor signatures and finally, in a very recent work, Chuang et al. [CHL23] investigated how the BIP32 specification can be translated to the two-party setting. That is, Chuang et al. presented two-party protocols for the key generation and hardened key derivation as specified by BIP32, implemented these protocols and evaluated their efficiency. However, the work of Chuang et al. is restricted to the two-party setting, whereas we investigate the more general  $(t, n)$  setting.

**Threshold ECDSA.** In recent years, there has been huge interest on threshold ECDSA (e.g., [LN18, CGG+20, DMZ+21, CCL+20, CCL+21, BMP22]), especially due to the application for securing cryptocurrency wallets. For a more in-depth comparison of different threshold ECDSA schemes, we refer to the survey of Aumasson et al. [AHS20]. As mentioned above, our work is based on the threshold ECDSA scheme of Gennaro and Goldfeder [GG18]. Groth and Shoup [GS22] introduced a threshold ECDSA scheme with additive key rerandomization according to the BIP32 specification. However, the authors do not consider the derivation of hardened nodes in the threshold setting, which is the main focus of our paper. Groth and Shoup analyze their scheme in the ideal/real world setting w.r.t. an ECDSA-specific functionality, whereas we give a general game-based definition for threshold signature schemes with rerandomizable keys and show that the construction of Gennaro and Goldfeder [GG18] can be extended to satisfy our definition. Finally, their scheme is rather complex, whereas we are aiming for a simple threshold ECDSA scheme.

## 2 Overview of our Solution

In our work, we consider thresholdizing non-hardened nodes within a BIP32 tree structure, s.t. each node consists of several devices where each of them stores a *share* of the signing secret key. This design choice allows to guarantee security even if a subset of the devices are corrupted. Our idea is to instantiate non-hardened wallets with a  $(t, n)$ -threshold signature scheme such that each non-hardened wallet is

“split” into  $n$  different devices, each of which stores only a share of the signing secret key. As we are using a  $(t, n)$ -threshold signature scheme, at least  $t + 1$  devices are required to sign a message. Simultaneously, the secret key of the non-hardened wallet remains secure as long as at most  $t$  devices are corrupted. All  $n$  devices store the public key and chaincode, s.t. only a single device can derive a non-hardened child public key without having to interact with the remaining  $n - 1$  devices. In the following, we will summarize our contribution in more detail.

**Threshold Signature Schemes with Rerandomizable Keys.** Das et al. [DEF<sup>+</sup>21] showed that one can generically construct hierarchical deterministic wallets from signature schemes with rerandomizable keys. Such signature schemes allow to deterministically rerandomize the secret/public key pair of a signature scheme such that the rerandomized key pair constitutes again a valid signing key pair. In our threshold setting, we therefore require a threshold signature scheme with rerandomizable keys. To this end, we first provide a game-based definition of such a primitive, and show an instantiation based on the threshold ECDSA scheme of Gennaro and Goldfeder [GG18], denoted as GG18. Importantly, we devise public and secret key rerandomization algorithms which allow to rerandomize the respective keys *non-interactively*, i.e., parties do not have to communicate to rerandomize the scheme’s public key and their respective secret key shares. This is an important property for wallet schemes as we generally aim to minimize communication between wallet devices. We intentionally choose the threshold ECDSA scheme of GG18 for our instantiation for the following reasons: (1) the scheme is used in practice for threshold wallet solutions, e.g., by BitGo <sup>5</sup>; (2) it is relatively simple, i.e., it does not include advanced features like offline signing or proactive/adaptive security which significantly increase the complexity of other threshold ECDSA schemes; (3) several threshold ECDSA schemes directly build upon the protocol of GG18 [CGG<sup>+</sup>20, DMZ<sup>+</sup>21, CCL<sup>+</sup>20, CCL<sup>+</sup>21], improving either efficiency, functionality, or security. Since their general idea is similar to the original scheme of GG18, we believe that our results can be extended to these schemes. We leave exploring such extensions as an interesting direction for future work.

**(Non-)Hardened Node Derivation.** As a second step, we translate the (non-)hardened derivation mechanisms as specified by BIP32 to the threshold setting. To this end, we first observe that using the public and secret key rerandomization algorithms of our rerandomizable threshold ECDSA scheme we can derive non-hardened nodes according to BIP32. However, in order to securely send the rerandomized secret key shares from the parent to the child node devices, we inherently require a communication heavy protocol that re-shares the key shares from the parent to the child. We then show that the hardened node derivation cannot easily be translated to the threshold setting due to the following issue: As mentioned above, the hardened node derivation in BIP32 requires to compute a hash function evaluation on input the secret key of the parent node (and some additional inputs). In the threshold setting, however, the secret key is shared among  $n$  devices and hence, adhering to the hardened node derivation of BIP32 would require all  $n$  devices to run an MPC protocol which securely computes this hash function evaluation. However, the following two issues arise when considering the use of MPC: First, MPC protocols like MOTION [BDST20] or TurboPack [EGPS22] focus on the semi-honest setting, while the hardened node derivation should be secure against malicious adversaries. Second, even for highly optimized protocols with malicious security [Kel20, HHPV21, GS20, AHKP24] the communication overhead is still significant. In particular the computation of the SHA-512 hash function, which is used by the BIP32 standard, is inherently expensive to compute via MPC [BST21, MPC20]. Hence, instead of using MPC protocols, we provide an alternative, but concretely efficient and non-interactive hardened node derivation mechanism that still satisfies all properties of the hardened node derivation as specified by BIP32.

Our mechanism uses a (non-interactive) threshold verifiable random function (TVRF) [Dod03], which allows the  $n$  non-hardened node devices to deterministically and efficiently compute a pseudorandom value. This value can be used by the hardened node as input to the key generation algorithm of a (non-threshold) signature scheme to deterministically generate its keys. However, for this approach each non-hardened wallet instance in the tree must maintain two secret/public key pairs: one for the threshold signing scheme and one for the TVRF scheme. Similarly to the signing key pair, the TVRF keys must be deterministically derived throughout the entire tree. That is, when deriving a non-hardened node, the parent must re-share its signing *and* its TVRF keys, which introduces a significant communication overhead, considering that we essentially double the amount of required communication for *each* non-hardened node derivation. We note that the TVRF keys are only required for the derivation of hardened nodes. Yet, in practice most

---

<sup>5</sup><https://www.bitgo.de/>

non-hardened wallets never derive a hardened child node and therefore would never make use of the TVRF keys, essentially wasting the communication required to derive the keys.

Due to this reason, our idea to overcome the drawback of maintaining and deriving a second key pair is to re-use the signing key pair of non-hardened nodes for the TVRF. While it is usually not recommended to re-use the same secret key over multiple cryptographic primitives, we prove that in our concrete case, re-using the same secret key does not compromise security. This constitutes the main technical contribution of our work. To this end, we first formally define security properties for the joint threshold signature/TVRF scheme, and we then prove that the combined scheme satisfies our properties. The main challenge in our proof is that we must reduce the security of the joint scheme to the security of the underlying TVRF scheme. The difficulty here is that an adversary against the joint scheme is allowed to receive signatures under the schemes' secret key, while the reduction to the TVRF security does not obtain access to a signing oracle. The reduction therefore must simulate the signing protocol to the adversary in the joint scheme without having access to a signing oracle itself.

### 3 Preliminaries

**Notation.** We use  $s \xleftarrow{\$} H$  to denote the uniform random sampling of a value  $s$  from a set  $H$ . By  $[l]$  for an integer  $l$ , we denote the set of integers  $\{1, \dots, l\}$  and for an algorithm  $A$ , we denote by  $y \leftarrow A(x)$  the execution of  $A$  on input  $x$  that outputs  $y$ . We write  $y \in A(x)$  to denote that  $y$  is an element in the set of possible outputs of an execution of  $A$  on input  $x$ . Throughout our paper, we often avoid explicitly specifying public parameters  $\text{par}$ . Given two strings  $a$  and  $b$ , we write  $a = (b, \cdot)$  if  $b$  is a prefix of  $a$ . For a set of  $n$  parties  $\{P_1, \dots, P_n\}$  and an interactive algorithm  $\Pi$ , we denote by  $\langle \Pi(x_1), \dots, \Pi(x_n) \rangle$  the joint execution of  $\Pi$  by all parties  $P_i$  for  $i \in [n]$  with respective inputs  $x_i$ .

#### 3.1 Interactive Threshold Signatures

In the following, we recall the definition of interactive threshold signature schemes.

**Definition 3.1** (Interactive Threshold Signature Scheme). An interactive  $(t, n)$ -threshold signature scheme  $\text{TSig}$  is executed among a set of  $n$  parties  $\{P_1, \dots, P_n\}$  and consists of a tuple of procedures  $\text{TSig} = (\text{Gen}, \text{TSign}, \text{Verify})$  which are defined as follows:

- $\text{Gen}(1^\kappa, t, n)$ : The probabilistic key generation algorithm takes as input a security parameter  $\kappa$  and two integers  $t, n \in \mathbb{N}$  such that  $t < n$ . It outputs a public key  $\text{pk}$  and a set of secret key shares  $\{\text{sk}_1, \dots, \text{sk}_n\}$  such that each party  $P_i$  obtains  $\text{pk}$  and  $\text{sk}_i$ .
- $\langle \text{TSign}(\text{sk}_1, m), \dots, \text{TSign}(\text{sk}_n, m) \rangle$ : The joint execution of the probabilistic signing algorithms, where each of which takes as input a secret key share  $\text{sk}_i$  for  $i \in [n]$  and a message  $m$ , outputs either a signature  $\sigma$  or  $\perp$ .
- $\text{Verify}(\text{pk}, m, \sigma)$ : The deterministic verification algorithm takes as input a public key  $\text{pk}$ , a message  $m$  and a signature  $\sigma$  and outputs a bit  $0/1$ .

*Correctness.* An interactive  $(t, n)$ -threshold signature scheme  $\text{TSig}$  is correct if for all  $\kappa \in \mathbb{N}$ , all  $t, n \in \mathbb{N}$  with  $t < n$ , all  $(\{\text{sk}_1, \dots, \text{sk}_n\}, \text{pk}) \leftarrow \text{Gen}(1^\kappa, t, n)$ , and all  $m \in \{0, 1\}^*$ , it holds that  $\Pr[\text{Verify}(\text{pk}, m, \sigma) = 1]$ , where  $\sigma \leftarrow \langle \text{TSign}(\text{sk}_1, m), \dots, \text{TSign}(\text{sk}_n, m) \rangle = 1$ .

**Definition 3.2** (Unforgeability of interactive threshold signature schemes). An interactive  $(t, n)$ -threshold signature scheme  $\text{TSig}$  is unforgeable if no PPT adversary  $\mathcal{A}$  wins game **th-ufcma** as described below with more than negligible advantage. We define  $\mathcal{A}$ 's advantage in game **th-ufcma** $_{\text{TSig}}$  as:

$$\text{Adv}_{\text{th-ufcma}_{\text{TSig}}}^{\mathcal{A}} := \Pr[\text{th-ufcma}_{\text{TSig}}^{\mathcal{A}} = 1].$$

Game **th-ufcma** $_{\text{TSig}}$ :

- The adversary  $\mathcal{A}$  outputs a list of corrupted parties  $\mathcal{C}$ , such that  $|\mathcal{C}| \leq t$  and for all  $i \in \mathcal{C}$  it holds that  $i \in [n]$ .
- The game initializes a list  $\text{SigList} \leftarrow \{\epsilon\}$  and executes  $(\{\text{sk}_1, \dots, \text{sk}_n\}, \text{pk}) \leftarrow \text{TSig.Gen}(1^\kappa, t, n)$ . Then  $\mathcal{A}$  is run on  $\text{pk}$  and  $\{\text{sk}_i\}_{i \in \mathcal{C}}$ .

- The adversary obtains access to the following **Sign** oracle: On input message  $m$ , the oracle and the adversary jointly execute the procedure **TSig.TSign**, where the oracle runs all honest parties  $P_i$  on input  $(\text{sk}_i, m)$ . The message  $m$  is then stored in **SigList**.
- Eventually, the adversary outputs a forgery  $\sigma^*$  and a message  $m^*$ . The adversary wins the game, if the following conditions hold: (1)  $\text{TSig.Verify}(\text{pk}^*, m^*, \sigma^*) = 1$  and (2)  $m^* \notin \text{SigList}$ .

### 3.2 Signature Scheme with Honestly Rerandomizable Keys

The notion of signature schemes with rerandomizable keys has first been introduced by Fleischhacker et al. [FKM<sup>+</sup>16].

**Definition 3.3** (Signature Scheme with Perfectly Rerandomizable Keys). Let the public parameters  $\text{par}$  define a randomness space  $\mathcal{R} := \mathcal{R}(\text{par})$ . A signature scheme with perfectly rerandomizable keys is a tuple of algorithms  $\text{RSig} = (\text{Gen}, \text{Sign}, \text{Verify}, \text{RandSK}, \text{RandPK})$  where  $(\text{Gen}, \text{Sign}, \text{Verify})$  are the standard algorithms of a signature scheme. The algorithms **RandSK** and **RandPK** are defined as follows:

- **RandSK**( $\text{sk}, \rho$ ): The deterministic secret key rerandomization **RandSK** takes as input a secret key  $\text{sk}$  and randomness  $\rho \in \mathcal{R}$  and outputs a rerandomized secret key  $\text{sk}'$ .
- **RandPK**( $\text{pk}, \rho$ ): The deterministic public key rerandomization **RandPK** takes as input a public key  $\text{pk}$  and randomness  $\rho \in \mathcal{R}$  and outputs a rerandomized public key  $\text{pk}'$ .

We recall the correctness definition and the security notion of *one-per message existential unforgeability under honestly rerandomizable keys* (**uf-cma-hrk1**) for signature schemes with rerandomizable keys [DEF<sup>+</sup>21] in Appendix A.1.

### 3.3 Non-Interactive Threshold Verifiable Random Function

We recall the definition of a non-interactive threshold verifiable random function from Galindo et al. [GLOW21].<sup>6</sup>

**Definition 3.4** A non-interactive  $(t, n)$ -threshold verifiable random function (TVRF) is defined w.r.t. to a randomness space  $\text{Rand}$  and is executed among  $n$  parties  $\{P_1, \dots, P_n\}$ . It consists of a tuple of algorithms  $\text{TVRF} = (\text{Gen}, \text{PEval}, \text{Combine}, \text{Verify})$  which are defined as follows:

- **Gen**( $1^\kappa, t, n$ ): The probabilistic key generation algorithm **Gen** takes as input a security parameter  $\kappa$  and two integers  $t, n \in \mathbb{N}$  such that  $t < n$ . It outputs a public key  $\text{pk}$  and a set of secret key shares  $\{\text{sk}_1, \dots, \text{sk}_n\}$  such that each party  $P_i$  obtains  $\text{pk}$  and  $\text{sk}_i$ .
- **PEval**( $m, \text{sk}_i, \text{pk}$ ): The partial evaluation algorithm **PEval** takes as input a message  $m$ , a secret key share  $\text{sk}_i$ , and a public key  $\text{pk}$ , and it outputs an evaluation share  $\phi_i$  and a proof  $\pi_i$ .
- **Combine**( $\text{pk}, m, \mathcal{S}, \{\phi_i, \pi_i\}_{i \in \mathcal{S}}$ ): The combination algorithm **Combine** takes as input a public key  $\text{pk}$ , a message  $m$ , a set of indices  $\mathcal{S}$  with  $|\mathcal{S}| > t$ , and a set of partial evaluation shares  $\{\phi_i, \pi_i\}_{i \in \mathcal{S}}$ . It outputs either a function evaluation  $\phi \in \text{Rand}$  and a proof  $\pi$ , or  $\perp$ .
- **Verify**( $\text{pk}, m, \phi, \pi$ ): The verification algorithm **Verify** takes as input a public key  $\text{pk}$ , a message  $m$ , a function evaluation  $\phi \in \text{Rand}$ , and a proof  $\pi$ , and outputs 0 or 1.

**Definition 3.5** (Uniqueness of TVRF). A non-interactive  $(t, n)$ -threshold verifiable random function scheme **TVRF** is **th-unique**-secure if no PPT adversary  $\mathcal{A}$  wins game **th-unique** as described below with more than negligible advantage. We define  $\mathcal{A}$ 's advantage in game **th-unique**<sub>TVRF</sub> as  $\text{Adv}_{\text{th-unique-TVRF}}^{\mathcal{A}} := \Pr[\text{th-unique}_{\text{TVRF}}^{\mathcal{A}} = 1]$ .

Game **th-unique**<sub>TVRF</sub>:

---

<sup>6</sup>We note that Galindo et al. refer to the primitive in their work as *non-interactive fully distributed verifiable random function*.

- The adversary  $\mathcal{A}$  outputs a list of corrupted parties  $C$ , such that  $|C| \leq t$  and for all  $i \in C$  it holds that  $i \in [n]$ .
- The game runs  $(\text{pk}, \{\text{sk}_1, \dots, \text{sk}_n\}) \leftarrow \text{TVRF.Gen}(1^\kappa, t, n)$ .  $\mathcal{A}$  receives as input  $\text{pk}$  and  $\{\text{sk}_i\}_{i \in C}$ .
- The adversary obtains access to the following oracles:
  - **Eval**: On input message  $m$  and index  $i \in [n] \setminus C$ , the oracle executes  $(\phi_i, \pi_i) \leftarrow \text{TVRF.PEval}(m, \text{sk}_i, \text{pk})$  and returns  $(\phi_i, \pi_i)$ .
  - **KeyLeak**: On input  $i \in [n]$ , the oracle outputs  $\text{sk}_i$ .
- Eventually, the adversary outputs a message  $m^*$  and two function evaluations  $\{(\phi^{i^*}, \pi^{i^*})\}_{i \in \{0,1\}}$ . The game outputs 1 if  $\phi^{0^*} \neq \phi^{1^*}$  and  $\text{TVRF.Verify}(\text{pk}, m^*, \phi^{0^*}, \pi^{0^*}) = \text{TVRF.Verify}(\text{pk}, m^*, \phi^{1^*}, \pi^{1^*}) = 1$ . Else it outputs 0.

**Definition 3.6** (Pseudorandomness of TVRF). A non-interactive  $(t, n)$ -threshold verifiable random function scheme TVRF is **th-prand**-secure if no PPT adversary  $\mathcal{A}$  wins game **th-prand** as described below with more than negligible advantage. We define  $\mathcal{A}$ 's advantage in game **th-prand**<sub>TVRF</sub> as  $\text{Adv}_{\text{th-prand}_{\text{TVRF}}}^{\mathcal{A}} := \Pr[\text{th-prand}_{\text{TVRF}}^{\mathcal{A}} = 1] - \frac{1}{2}$ .

Game **th-prand**<sub>TVRF</sub>:

- The adversary  $\mathcal{A}$  outputs a list of corrupted parties  $C$ , such that  $|C| \leq t$  and for all  $i \in C$  it holds that  $i \in [n]$ .
- The game initializes  $\text{EvalList} \leftarrow \{\epsilon\}$  and executes  $(\text{pk}, \{\text{sk}_1, \dots, \text{sk}_n\}) \leftarrow \text{TVRF.Gen}(1^\kappa, t, n)$ .  $\mathcal{A}$  receives as input  $\text{pk}$  and  $\{\text{sk}_i\}_{i \in C}$ .
- The adversary obtains access to the following oracle:
  - **Eval**: On input message  $m$  and an index  $i \in [n] \setminus C$ , the oracle executes  $(\phi_i, \pi_i) \leftarrow \text{TVRF.PEval}(m, \text{sk}_i, \text{pk})$  and if  $(i, m) \notin \text{EvalList}$ , stores the tuple  $(i, m)$  in  $\text{EvalList}$ . The oracle returns  $(\phi_i, \pi_i)$ .
- Eventually, the adversary outputs a message  $m^*$ , a set of indices  $\mathcal{S}$  with  $|\mathcal{S}| > t$ , evaluation shares  $\{\phi_i, \pi_i\}_{i \in \mathcal{S} \cap C}$ . The game checks if there are less than  $t - |\mathcal{S} \cap C|$  tuples of the form  $(\cdot, m^*)$  in  $\text{EvalList}$  and if so, the game computes for  $j \in \mathcal{S} \setminus C$  the tuple  $(\phi_j, \pi_j) \leftarrow \text{TVRF.PEval}(m^*, \text{sk}_j, \text{pk})$  and  $(\phi, \pi) \leftarrow \text{TVRF.Combine}(\text{pk}, \mathcal{S}, \{\phi_i, \pi_i\}_{i \in \mathcal{S}})$ . If  $\phi = \perp$ , the game returns  $\phi$ . Otherwise the game chooses a bit  $b \xleftarrow{\$} \{0, 1\}$  and does the following:
  - If  $b = 0$ : Return  $\phi$ .
  - If  $b = 1$ : Sample  $\phi' \xleftarrow{\$} \text{Rand}$  and output  $\phi'$ .

The adversary then outputs a bit  $b'$  and wins if  $b = b'$ .

**Definition 3.7** (Robustness of TVRF). A non-interactive  $(t, n)$ -threshold verifiable random function scheme TVRF is **th-robust**-secure if no PPT adversary  $\mathcal{A}$  wins game **th-robust** as described below with more than negligible advantage. We define  $\mathcal{A}$ 's advantage in **th-robust**<sub>TVRF</sub> as  $\text{Adv}_{\text{th-robust}_{\text{TVRF}}}^{\mathcal{A}} := \Pr[\text{th-robust}_{\text{TVRF}}^{\mathcal{A}} = 1]$ .

Game **th-robust**<sub>TVRF</sub>:

- The game differs from game **th-prand**<sub>TVRF</sub> only by the winning condition, which we will describe below.
- The adversary outputs a message  $m^*$ , a set  $\mathcal{S}$  with  $|\mathcal{S}| > t$  and a set of evaluation shares  $\{\phi_i, \pi_i\}_{i \in \mathcal{S} \cap C}$ . The game computes  $(\phi_i, \pi_i) \leftarrow \text{TVRF.PEval}(m^*, \text{sk}_i, \text{pk})$  for all  $i \in \mathcal{S} \setminus C$ . The game finally sets  $(\phi^*, \pi^*) \leftarrow \text{TVRF.Combine}(\text{pk}, \mathcal{S}, \{\phi_i, \pi_i\}_{i \in \mathcal{S}})$ . If  $\phi^* \neq \perp$  and  $\text{TVRF.Verify}(\text{pk}, m^*, \phi^*, \pi^*) = 0$ , the game outputs 1 and 0 otherwise.

We additionally recall the TVRF Construction of Galindo et al. [GLOW21] in Appendix A.2.

## 4 Rerandomizable Interactive Threshold Signing

### 4.1 Model

In the following, we introduce the notion of interactive threshold signature schemes with rerandomizable keys. Specifically, we extend the standard notion of a threshold signature scheme by two algorithms  $\text{RandSK}$  and  $\text{RandPK}$ , which allow to derive rerandomized secret key shares and a rerandomized public key, respectively, such that the derived secret key shares form a valid  $(t, n)$ -sharing of the secret key corresponding to the derived public key.

**Definition 4.1** (Interactive Threshold Signature Scheme With Rerandomizable Keys). An interactive  $(t, n)$ -threshold signature scheme with rerandomizable keys is a tuple of procedures  $\text{RTSig} = (\text{Gen}, \text{RandSK}, \text{RandPK}, \text{TSign}, \text{Verify})$  where  $(\text{Gen}, \text{TSign}, \text{Verify})$  are defined as for interactive  $(t, n)$ -threshold signatures. We assume that the public parameters  $\text{par}$  define a randomness space  $\mathcal{R} := \mathcal{R}(\text{par})$ . The algorithms  $\text{RandSK}$  and  $\text{RandPK}$  are defined as:

- $\text{RandSK}(i, \text{sk}_i, \rho)$ : The deterministic secret key share rerandomization takes as input an index  $i \in [n]$ , a secret key share  $\text{sk}_i$  and a randomness  $\rho \in \mathcal{R}$  and it outputs a rerandomized secret key share  $\text{sk}'_i$ .
- $\text{RandPK}(\text{pk}, \rho)$ : The deterministic public key rerandomization takes as input a public key  $\text{pk}$  and a randomness  $\rho \in \mathcal{R}$  and it outputs a rerandomized public key  $\text{pk}'$ .

We require the following properties of a threshold signature scheme with rerandomizable keys:

- *Rerandomizability of public keys*: For all  $\kappa \in \mathbb{N}$ , all  $t, n \in \mathbb{N}$  with  $t < n$ , all  $(\cdot, \text{pk}) \leftarrow \text{Gen}(1^\kappa, t, n)$  and all  $\rho \xleftarrow{\$} \mathcal{R}$ , the distributions of  $\text{pk}'$  and  $\text{pk}''$  are computationally indistinguishable, where  $\text{pk}' \leftarrow \text{RandPK}(\text{pk}, \rho)$  and  $(\cdot, \text{pk}'') \leftarrow \text{Gen}(1^\kappa, t, n)$ .
- *Correctness under rerandomized keys*: For all  $\kappa \in \mathbb{N}$ , all  $t, n \in \mathbb{N}$  with  $t < n$ , all  $(\{\text{sk}_1, \dots, \text{sk}_n\}, \text{pk}) \leftarrow \text{Gen}(1^\kappa, t, n)$ , all  $\rho \xleftarrow{\$} \mathcal{R}$ , and all messages  $m \in \{0, 1\}^*$ , for the rerandomized keys  $\{\text{sk}'_i\}_{i \in [n]} \leftarrow \{\text{RandSK}(i, \text{sk}_i, \rho)\}_{i \in [n]}$  and  $\text{pk}' \leftarrow \text{RandPK}(\text{pk}, \rho)$  it holds that:  $\Pr[\text{Verify}(\text{pk}', m, \sigma)] = 1$ , with the signature  $\sigma$  as  $\sigma \leftarrow \langle \text{TSign}(\text{sk}'_1, m), \dots, \text{TSign}(\text{sk}'_n, m) \rangle$ .

We note that the property of rerandomizability of public keys is a slightly weaker notion than the perfect rerandomizability of keys of rerandomizable signature schemes (cf. Def. 3.3) which requires rerandomized public and secret keys to be identically distributed to a freshly generated key pair. However, as previously pointed out by Alkadri et al. [ADE<sup>+</sup>20], this weaker rerandomizability property is sufficient for the wallet setting. At a high level, that is because this notion is required for the wallet unlinkability property, which guarantees unlinkability of wallet *public keys*, i.e., it guarantees that a derived public key is computationally indistinguishable from freshly generated public keys.

We define the security notion of *one-per message existential unforgeability under honestly rerandomizable keys* for interactive threshold signature schemes with rerandomizable keys. That is, we define a security game **th-ufcma-hrk1** which differs from the unforgeability game **th-ufcma** (cf. Def. 3.2) of interactive threshold signatures in the following ways: (1) the adversary receives access to a **Rand** oracle, which outputs uniformly random elements from  $\mathcal{R}$ ; (2) the signing oracle **RSign** can generate signatures under the initial key set  $(\{\text{sk}_1, \dots, \text{sk}_n\}, \text{pk})$ , as well as under key sets rerandomized with an element output by the **Rand** oracle; (3) the signing oracle returns at most one signature for each key set/message pair; and (4) the adversary can win the game with a valid forgery under *any* key set rerandomized with an output of the **Rand** oracle. We note that the notion of one-per message unforgeability is weaker than standard unforgeability, however, as remarked by Das et al. [DEF<sup>+</sup>21] this weaker notion is sufficient for the wallet setting.

**Definition 4.2** (One-per message unforgeability of interactive threshold signature schemes with honestly rerandomizable keys). An interactive  $(t, n)$ -threshold signature scheme with rerandomizable keys  $\text{RTSig} = (\text{Gen}, \text{RandSK}, \text{RandPK}, \text{TSign}, \text{Verify})$  is secure under **th-ufcma-hrk1** if no PPT adversary  $\mathcal{A}$  wins game **th-ufcma-hrk1** as described below with more than negligible probability in the security parameter  $\kappa$ .

Game **th-ufcma-hrk1**<sub>RTSig</sub>:

- The adversary  $\mathcal{A}$  outputs a list of corrupted parties  $C$ , such that  $|C| \leq t$  and for all  $i \in C$  it holds that  $i \in [n]$ .
- The game initializes two lists  $RList \leftarrow \{\epsilon\}$  and  $SigList \leftarrow \{\epsilon\}$  and executes  $(\{sk_1, \dots, sk_n\}, pk) \leftarrow \text{RTSig.Gen}(1^\kappa, t, n)$ . Then,  $\mathcal{A}$  is run on input  $pk$  and  $\{sk_i\}_{i \in C}$ .
- The adversary obtains access to the following two oracles:
  - **Rand**: This oracle, upon a query, samples  $\rho \xleftarrow{\$} \mathcal{R}$ , stores  $\rho$  in  $RList$  and outputs  $\rho$  to  $\mathcal{A}$ .
  - **RSig**: On input message  $m$  and a randomness  $\rho$ , the oracle checks whether  $\rho \notin RList$  and if so outputs  $\perp$ . Otherwise, it derives a public key and secret key shares for honest parties with the randomness  $\rho$ , i.e., it computes  $pk' \leftarrow \text{RTSig.RandPK}(pk, \rho)$  and  $sk'_i \leftarrow \text{RTSig.RandSK}(i, sk_i, \rho)$  for all  $i \in \{1, \dots, n\} \setminus C$ . If  $(pk', m) \in SigList$  then the oracle returns  $\perp$ . Otherwise, the oracle and the adversary jointly execute the procedure  $\text{RTSig.TSign}$ , where the oracle runs all honest parties  $P_i$  on input  $(sk'_i, m)$ . The oracle then stores the tuple  $(pk', m)$  in  $SigList$ .
- Eventually, the adversary outputs a forgery  $\sigma^*$ , a message  $m^*$  and a public key  $pk^* \leftarrow \text{RTSig.RandPK}(pk, \rho^*)$ . The adversary wins the game, if the following conditions hold: (1)  $\rho^* \in RList$ , (2)  $(pk^*, m^*) \notin SigList$ , and (3)  $\text{RTSig.Verify}(pk^*, m^*, \sigma^*) = 1$ .

## 4.2 Construction

We show how to extend the interactive threshold ECDSA scheme as proposed by GG18 [GG18] (which we denote by  $\text{GG}[H_0]$  and recall in detail in Appendix B) to an interactive threshold ECDSA scheme with rerandomizable keys (which we denote by  $\text{rGG}[H_0]$ ). In Figure 2, we describe our  $\text{rGG}[H_0]$  scheme w.r.t. the  $\text{GG}[H_0]$  scheme. Recall that the ECDSA signature scheme is defined w.r.t. a cyclic group  $\mathbb{G} = \langle g \rangle$  of prime order  $q$  and that an ECDSA key pair  $(pk, sk)$  is simply computed as  $sk \xleftarrow{\$} \mathbb{Z}_q$  and  $pk \leftarrow g^{sk}$ . In the  $\text{GG}[H_0]$  scheme, the secret key is shared such that each party  $P_i$  holds a secret key share  $sk_i$  and a public key share  $g^{sk_i}$ . In our  $\text{rGG}[H_0]$  scheme, we extend the  $\text{GG}[H_0]$  scheme by providing algorithms  $\text{RandSK}$  and  $\text{RandPK}$  which deterministically rerandomize the secret key shares and the public key respectively w.r.t. a randomness  $\rho$ . At a high level, in order to rerandomize the secret key share  $sk_i$  of party  $P_i$  with randomness  $\rho$ , the  $\text{RandSK}$  algorithm deterministically generates a degree- $t$  polynomial  $F$  with coefficients in  $\mathbb{Z}_q$  and evaluates the polynomial at point  $i$ . This essentially yields a randomness share  $\rho_i$ , which is then added to the existing secret key share to compute the rerandomized secret key share  $sk'_i \leftarrow sk_i + \rho_i \pmod q$ . That is,  $sk'_i$  is essentially a share of the secret key  $sk + \rho \pmod q$ . The  $\text{RandPK}$  algorithm works correspondingly for the public key and public key shares.

The security of our  $\text{rGG}[H_0]$  scheme can be proven via a reduction to the (one-per message) unforgeability of the ECDSA scheme with rerandomizable keys by Das et al. [DEF+21] (see Appendix A.3). Note that the scheme of Das et al. is public key prefixed, i.e., whenever a message  $m$  is signed using secret key  $sk$ , the message is first prefixed with the corresponding public key  $pk$ , s.t. the signature is generated for  $(pk, m)$ . Since we reduce the security of our  $\text{rGG}[H_0]$  scheme to the (one-per message) unforgeability of the scheme of Das et al., we require public key prefixing in our scheme as well.

It is easy to see that the  $\text{rGG}[H_0]$  scheme satisfies the *correctness under rerandomized keys* property.

**Lemma 4.3** *Let  $H_0 : \{0, 1\}^* \rightarrow \mathbb{Z}_q$  be a hash function modeled as a random oracle and let the discrete logarithm problem be hard in  $\mathbb{G}$ . Then the interactive  $(t, n)$ -threshold ECDSA scheme with rerandomizable keys  $\text{rGG}[H_0]$  satisfies the rerandomizability of public keys property.*

*Proof Sketch.* We can prove the above lemma via reduction to the discrete logarithm problem. At a high level, assume there exists a PPT adversary  $\mathcal{A}$  that can distinguish the following distributions

$$\begin{aligned} & \{pk, pk' \mid (\cdot, pk) \leftarrow \text{Gen}(1^\kappa, t, n), \rho \xleftarrow{\$} \mathcal{R}, pk' \leftarrow \text{RandPK}(pk, \rho)\} \\ & \{pk, pk'' \mid (\cdot, pk) \leftarrow \text{Gen}(1^\kappa, t, n), (\cdot, pk'') \leftarrow \text{Gen}(1^\kappa, t, n)\} \end{aligned}$$

with more than negligible probability, then we can construct a PPT adversary  $\mathcal{B}$  that breaks the discrete logarithm problem with a related probability. In the following, we sketch how this reduction proceeds: adversary  $\mathcal{B}$  receives a discrete logarithm challenge  $g^\rho$  as input.  $\mathcal{B}$  then samples  $(\cdot, pk) \leftarrow \text{Gen}(1^\kappa, t, n)$  and computes  $pk'$  by executing algorithm  $\text{RandPK}(pk, \rho)$ , however with the following differences: (1) it

Algorithm $\text{Gen}(1^\kappa, t, n)$ 00 Return $\text{GG.Gen}(1^\kappa, t, n)$  Algorithm $\text{RandSK}(i, \text{sk}_i, \rho)$ 00 For $k \in [t] : a_k \leftarrow \text{H}_0(\rho, k)$ 01 Let $F(x) := a_t x^t + \dots + a_1 x + \rho$ 02 $\rho_i \leftarrow F(i) \bmod q$ 03 $\text{sk}'_i \leftarrow \text{sk}_i + \rho_i \bmod q$ 04 Return $\text{sk}'_i$  Protocol $\text{TSign}(\text{sk}_i, m)$ 00 $m' \leftarrow (\text{pk}, m)$ 01 Return $\text{GG.TSign}(\text{sk}_i, m')$	Algorithm $\text{RandPK}(\text{pk}, \rho)$ 00 Parse $\text{pk} := (X, (X_1, \dots, X_n))$ 01 For $k \in [t] : a_k \leftarrow \text{H}_0(\rho, k)$ 02 Let $F(x) := a_t x^t + \dots + a_1 x + \rho$ 03 $\rho_i \leftarrow F(i) \bmod q$ 04 For $i \in [n] : X'_i \leftarrow X_i \cdot g^{\rho_i}$ 05 Return $\text{pk}' := (X \cdot g^\rho, (X'_1, \dots, X'_n))$  Algorithm $\text{Verify}(\text{pk}, m, \sigma)$ 00 $m' \leftarrow (\text{pk}, m)$ 01 Return $\text{GG.Verify}(\text{pk}, m', \sigma)$
--	--

Figure 2: Public key prefixed interactive threshold ECDSA scheme  $\text{rGG}[\text{H}_0]$  with honestly rerandomizable keys based on the  $\text{GG}[\text{H}_0]$  scheme for a hash function  $\text{H}_0 : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ . For brevity, we denote scheme  $\text{GG}[\text{H}_0]$  by  $\text{GG}$ .

samples the coefficients  $a_k$  for  $k \in [t]$  of polynomial  $F$  uniformly at random from  $\mathbb{Z}_q$  instead of computing them as  $\text{H}_0(\rho, k)$ ; and (2) it evaluates polynomial  $F$  only in the exponent, i.e., it computes  $g^{\rho_i} \leftarrow g^{F(i)}$  for  $i \in [n]$ .<sup>7</sup> Note that  $\mathcal{A}$  can only detect these changes if it queries the random oracle  $\text{H}_0$  on input  $(\rho, k)$ . However, if  $\mathcal{A}$  makes a query of this form, adversary  $\mathcal{B}$  learns  $\rho$ , which is the discrete logarithm of its discrete logarithm challenge  $g^\rho$ . Otherwise, public key  $\text{pk}'$  is identically distributed to a freshly generated public key  $\text{pk}''$ . ■

**Theorem 4.4** *Let PKE be a semantically secure linearly homomorphic encryption scheme, ZK be a non-interactive zero-knowledge proof system and CT a non-malleable and equivocal commitment scheme. Further, let the DDH assumption hold in  $\mathbb{G}$  and let  $\text{rECDSA}[\text{H}_0]$  be the **uf-cma-hrk1**-secure ECDSA scheme with rerandomizable keys as described in Appendix A.3. Then the interactive  $(t, n)$ -threshold ECDSA scheme with rerandomizable keys  $\text{rGG}[\text{H}_0]$  as described above is **th-ufcma-hrk1**-secure.*

*Proof Sketch.* Gennaro and Goldfeder prove the  $\text{GG}[\text{H}_0]$  scheme unforgeable via reduction to the unforgeability of the single party ECDSA signature scheme. That is, they provide a reduction that simulates game **th-ufcma** $_{\text{GG}[\text{H}_0]}$  (cf. Definition 3.2) while having access to a signing oracle that outputs ECDSA signatures for adaptively chosen messages. Gennaro and Goldfeder prove that this simulation is computationally indistinguishable from the real game to a PPT adversary. We recall the simulation in Appendix B. We can prove the above theorem in the same way, with the difference that we reduce the **th-ufcma-hrk1** $_{\text{rGG}[\text{H}_0]}$  security to the **uf-cma-hrk1** $_{\text{rECDSA}[\text{H}_0]}$  security. That is, we have to provide a reduction that simulates game **th-ufcma-hrk1** $_{\text{rGG}[\text{H}_0]}$  to a PPT adversary while having access to the **RSign** and **Rand** oracles of game **uf-cma-hrk1** $_{\text{rECDSA}[\text{H}_0]}$ . In fact, we can use the same simulation as the one from GG18 with the following differences: (1) Upon the adversary querying the **Rand** oracle in game **th-ufcma-hrk1** $_{\text{rGG}[\text{H}_0]}$ , the reduction relays the query to its own **Rand** oracle in game **uf-cma-hrk1** $_{\text{rECDSA}[\text{H}_0]}$ ; (2) Upon the adversary querying oracle **RSign** in game **th-ufcma-hrk1** $_{\text{rGG}[\text{H}_0]}$  on input a message  $m$  and randomness  $\rho$ , the reduction first rerandomizes the secret key shares  $\text{sk}_i$  of the corrupted parties  $P_i \in \mathcal{C}$  as  $\text{sk}'_i \leftarrow \text{RandSK}(i, \text{sk}_i, \rho)$  as well as the public key  $\text{pk}' \leftarrow \text{RandPK}(\text{pk}, \rho)$ . The reduction then queries its own signing oracle on input  $m$  and  $\rho$  and uses the resulting signature and the rerandomized keys for the simulation of the **RSign** oracle of game **th-ufcma-hrk1** $_{\text{rGG}[\text{H}_0]}$ . These changes do not have any impact on the indistinguishability arguments and reduction from GG18. Note that, since we essentially repeat the proof of GG18, we must also repeat the assumptions their proof relies on in our theorem statement. ■

## 5 BIP32-Compatible Threshold Wallets

In order to construct threshold BIP32 wallets, we require two ingredients, namely (1) a threshold signature scheme with rerandomizable keys, and (2) mechanisms for the derivation of non-hardened and hardened wallets in the threshold setting. With requirement (1) in place, we will discuss in this section how the respective wallet derivations of a BIP32 wallet can be implemented in the threshold setting. In

<sup>7</sup>This step is necessary because  $\mathcal{B}$  only knows  $g^\rho$  but not  $\rho$ . Therefore,  $\mathcal{B}$  can only compute  $F$  in the exponent.

particular, we consider the following setting for our threshold BIP32 wallet: All non-hardened wallets are thresholdized, i.e., each non-hardened wallet consists of  $n$  devices which execute a  $(t, n)$ -threshold signature scheme with rerandomizable keys. We assume throughout the paper that  $t \leq \frac{n-1}{2}$  for all non-hardened nodes. That is, the security of a non-hardened wallet relies on the assumption that at most  $t$  devices of this wallet instance are corrupted. If an adversary can corrupt  $t + 1$  or more devices of a non-hardened instance, it can execute a privilege escalation attack, where the adversary can compute the secret keys of all parent wallet instances. Hardened wallets, on the other hand, are single devices (i.e. not thresholdized), since the corruption of a hardened wallet does not affect the security of the remaining wallets in the tree. Similar to the modeling of BIP32 wallets by Das et al. [DEF<sup>+</sup>21], we do not allow hardened wallets to derive child wallets, i.e., hardened wallets always represent leaves in the wallet tree. Therefore, we assume that in both cases, i.e., the non-hardened and hardened wallet derivation, the parent wallet is non-hardened and thresholdized. Recall that BIP32 specifies the (non-threshold) derivation mechanisms as follows: A non-hardened node with identifier  $ID'$  is derived from a parent node with identifier  $ID$ , key pair  $(sk_{ID}, pk_{ID})$  and chaincode  $ch_{ID}$  by computing  $(\rho, ch_{ID'}) \leftarrow H(pk_{ID}, ch_{ID}, ID')$ ,  $sk_{ID'} \leftarrow sk_{ID} + \rho$  and  $pk_{ID'} \leftarrow pk_{ID} \cdot g^\rho$ . The derivation of a hardened node works in the same way only that the tuple  $(\rho, ch_{ID'})$  is computed as  $H(sk_{ID}, ch_{ID}, ID')$ . We now analyze these derivation mechanisms for the threshold setting w.r.t. to our threshold signature scheme with rerandomizable keys  $rGG[H_0]$  in more detail.

## 5.1 Non-Hardened Node Derivation

The derivation of non-hardened nodes in the threshold setting is fairly straightforward and follows the ideas of the BIP32 standard. Essentially, a non-hardened parent node identified by  $ID$  and consisting of  $n$  devices s.t. each device stores a secret key share  $sk_{i,ID}$  and the chaincode  $ch_{ID}$  can derive a thresholdized non-hardened child wallet as follows: First, each device of the parent node computes locally  $(\rho, ch_{ID'}) \leftarrow H(pk_{ID}, ch_{ID}, ID')$  and  $sk_{i,ID'} \leftarrow rGG[H_0].RandSK(i, sk_{i,ID}, \rho)$ . Then the devices of the parent node must forward the rerandomized secret key shares  $sk_{i,ID'}$  and the chaincode  $ch_{ID'}$  to the  $n$  devices of the child node. The forwarding of the chaincode  $ch_{ID'}$  is straightforward, since we assume an honest majority among the parent devices and since each parent device knows  $ch_{ID'}$ . That is, all parent devices can simply send  $ch_{ID'}$  to all child devices. Each child device then receives at least  $t + 1$  times the value  $ch_{ID'}$  which it uses as the node's chaincode. The forwarding of the secret key shares  $sk_{i,ID'}$  is more involved and requires a protocol involving  $2n$  devices ( $n$  child and  $n$  parent wallet devices) of which a total of  $2t$  devices can be corrupted. Note that a simple forwarding of secret key share  $sk_{i,ID'}$  to the  $i$ -th device of the child wallet is insecure as it allows an adversary to learn a total of  $2t$  secret key shares. Instead, the  $2n$  devices must engage in the execution of a dynamic proactive secret sharing (DPSS) scheme (e.g., [BDLO15, MZW<sup>+</sup>19, SLL]), which allows to *securely* handover the rerandomized key shares to the devices of the child node even in the presence of  $2t$  corrupted devices. Note that DPSS schemes typically incur a significant communication overhead since all  $2n$  parties must interact with each other.

## 5.2 Hardened Node Derivation

The main challenge when considering BIP32 wallets in the threshold setting is designing a derivation mechanism for hardened nodes. Recall that the derivation of a hardened node according to BIP32 requires the computation of  $(\rho, ch_{ID'}) \leftarrow H(sk_{ID}, ch_{ID}, ID')$ , i.e., the evaluation of a hash function on input the parent secret key. In the threshold setting, however, the secret key  $sk_{ID}$  is shared among  $n$  devices such that no single device knows the full key. It is therefore not at all clear how  $H(sk_{ID}, ch_{ID}, ID')$  can be computed efficiently without naively reconstructing  $sk_{ID}$  (which would trivially break the security of the wallet). Furthermore, in the hardened derivation, each parent device can only learn a randomness *share*  $\rho_i$  instead of the entire randomness  $\rho$ . To see why that is, consider the setting where an adversary corrupts the hardened node, thereby learning its secret key  $sk_{ID} + \rho$ , as well as a parent node device, thereby learning  $\rho$ . The adversary could then trivially learn the parent node's secret key.

One obvious (and to the best of our knowledge the only) way to resolve the above issues is using generic multi-party computation (MPC) techniques [GMW87, Gol04, CCD88], which allow to securely compute any function in a distributed setting without revealing the function inputs. However, generic MPC is inherently inefficient, in particular since the BIP32 standard uses the well-known hash function SHA-512, which is known to be only inefficiently computable via MPC [BST21].

**An Improved Derivation Mechanism** Due to the above limitation, we consider a more efficient hardened node derivation mechanism, which achieves the same properties as the one originally specified in BIP32. We circumvent the inefficient distributed SHA-512 execution by letting the devices of the non-hardened parent wallet jointly and deterministically generate a random seed in such a way that only the hardened node but no parent device learns the seed. The hardened node can then use this seed as input to the key generation algorithm of a (non-threshold) signature scheme (ECDSA in our case) to deterministically generate its key pair. Said differently, instead of having the parent wallet devices rerandomize their secret key shares and forward them to the hardened wallet, we simply let the parent devices generate a random value from which the hardened node can deterministically derive its own keys. For the computation of the random seed, we employ the threshold verifiable random function (TVRF) from [GLOW21]. A  $(t, n)$ -TVRF is a cryptographic primitive that is executed by  $n$  parties, where each party  $P_i$  knows a secret key share  $sk_i$ , which it can use to deterministically compute an evaluation share  $\phi_i$  and proof  $\pi_i$  on a message  $m$ . Given at least  $t + 1$  evaluation shares for  $m$ , any party can deterministically compute a pseudorandom value  $\phi$  and a proof  $\pi$  and given the public key  $pk$ ,  $\phi$  and  $\pi$ , any party can verify that  $\phi$  was computed correctly. We recall the formal definition of a TVRF in Section 3.3.

We use the TVRF for the hardened wallet derivation in the following way: Each device of the non-hardened parent node maintains a secret key share for the TVRF and, upon the derivation of a hardened node with identifier  $ID$ , it uses this share to compute an evaluation share  $\phi_i$  and the corresponding proof  $\pi_i$  on  $ID$ . It then sends  $(\phi_i, \pi_i)$  to the hardened node, which combines  $t + 1$  shares to a pseudorandom seed  $\phi$ . The hardened node then verifies the correctness of  $\phi$  using the public key of the TVRF. Note that any set of  $t + 1$  correct evaluation shares will yield the same seed, but including only a single invalid evaluation share will lead to a different (incorrect) seed. Therefore, the verifiability of the seed is crucial to our solution. We use the TVRF from [GLOW21] which is not only deterministic and one-way but also non-interactively computable, therefore exhibiting the same properties as the original BIP32 derivation mechanism. We present our improved hardened node derivation mechanism pictorially in Figure 3.

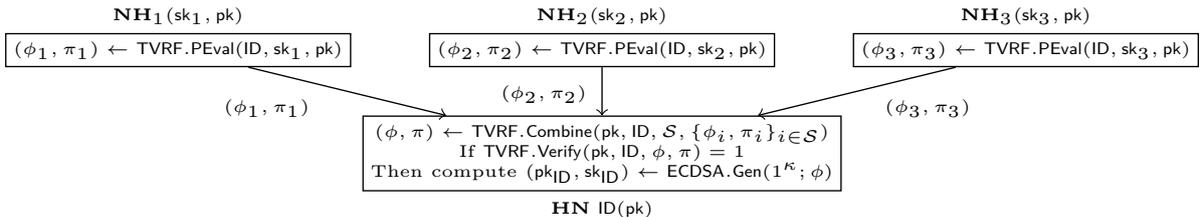


Figure 3: The improved hardened node derivation mechanism in the threshold setting. Each of the three devices  $\mathbf{NH}_1$ ,  $\mathbf{NH}_2$ ,  $\mathbf{NH}_3$  of the non-hardened parent node stores a TVRF public key  $pk$  and secret key share  $sk_i$  for  $i \in [3]$ . To derive a hardened node  $\mathbf{HN}$  with identity  $ID$ , each non-hardened device locally evaluates the TVRF on input  $ID$  and sends the resulting evaluation share to  $\mathbf{HN}$ . The hardened node then chooses a subset  $\mathcal{S}$  of  $[3]$ , combines the corresponding evaluation shares to a full random value  $\phi$ , verifies that the non-hardened devices in  $\mathcal{S}$  behaved honestly, and then uses  $\phi$  as input to the key generation algorithm of the ECDSA signature scheme. Note that this key generation is deterministic, as we explicitly give the randomness  $\phi$  as input.

**The Final Derivation Mechanism** While the above solution is compatible with BIP32, it has the significant drawback that each non-hardened device must maintain two secret key shares, one for the signature scheme and one for the TVRF. As a consequence, each device requires double the storage space which is an issue for space restricted devices. There is however another, more severe issue with the above solution. Similar to the signing keys, the TVRF keys must be deterministically derived throughout the wallet tree via executions of a communication heavy DPSS scheme. This incurs a significant communication overhead, especially since *all* non-hardened nodes must derive TVRF keys irrespectively of whether they want to derive a hardened node or not.

We observe that both, the DDH-based TVRF scheme of [GLOW21] (which we denote by TVRF and recall in Appendix A.2) and the ECDSA signature scheme, operate over a cyclic group  $\mathbb{G} = \langle g \rangle$  of prime order  $q$  and use secret/public key pairs  $sk \xleftarrow{\$} \mathbb{Z}_q$  and  $pk \leftarrow g^{sk}$ . The security of TVRF relies on the assumption that DDH is hard in  $\mathbb{G}$ . Bitcoin, Ethereum and several other cryptocurrencies use the group

$\mathbb{G}$  identified by the elliptic curve secp256k1, for which dlog and DDH are assumed to be hard. Therefore, our idea to mitigate the above issues is to use only a single key pair for both schemes. This allows non-hardened wallets to re-use their signing secret key shares for the TVRF during the hardened node derivation, thereby avoiding the overhead of maintaining a second key pair per wallet.

In the remainder of this section, we define a cryptographic scheme that consists of the joint procedures of the rGG scheme from Section 4 and of the DDH-based TVRF scheme, but that uses the same key pair for all procedures. We then define security properties and prove the scheme secure w.r.t. these properties.

### 5.3 Joint Threshold Signature/TVRF Scheme

We define a scheme  $(t, n)$ -TVRF-rGG $[H_0, H_1]$ , which consists of all procedures of the interactive  $(t, n)$ -threshold ECDSA scheme with rerandomizable keys rGG $[H_0]$  and the non-interactive  $(t, n)$ -threshold verifiable random function scheme TVRF $[H_1]$ , except that it uses only one of rGG $[H_0]$ .Gen and TVRF $[H_1]$ .Gen. Concretely, TVRF-rGG $[H_0, H_1]$  consists of the procedures

$$\begin{aligned} \text{TVRF-rGG}[H_0, H_1] = & (\text{rGG}[H_0].\text{Gen}, \text{rGG}[H_0].\text{RandSK}, \text{rGG}[H_0].\text{RandPK}, \text{rGG}[H_0].\text{TSign}, \\ & \text{rGG}[H_0].\text{Verify}, \text{TVRF}[H_1].\text{PEval}, \text{TVRF}[H_1].\text{Combine}, \text{TVRF}[H_1].\text{Verify}). \end{aligned}$$

For simplicity, we sometimes abbreviate the schemes TVRF-rGG $[H_0, H_1]$ , rGG $[H_0]$  and TVRF $[H_1]$  by TVRF-rGG, rGG and TVRF respectively. The TVRF-rGG scheme must satisfy the security properties *pseudorandomness*, *uniqueness*, and *robustness*. These security notions essentially combine the respective security properties of the TVRF scheme with the one-more unforgeability notion of our rGG scheme. That is, for each of the above security notions, we define a game, where an adversary (1) can corrupt  $t$  parties, (2) receives oracle access to all oracles of the one-more unforgeability game (i.e., **th-ufcma-hrk1**) and all oracles of the respective TVRF property (e.g., pseudorandomness), and (3) can win the game by either breaking the one-more unforgeability of rGG (**Case 1**) or the TVRF property (**Case 2**).

#### 5.3.1 Pseudorandomness of TVRF-rGG

In the following we define the pseudorandomness property of TVRF-rGG via a game **unf-prand** and prove that TVRF-rGG satisfies this property. Later in Section 5.3.2, we provide the uniqueness and robustness definitions and argue that the TVRF-rGG scheme satisfies them.

**Definition 5.1** (Pseudorandomness of TVRF-rGG). The  $(t, n)$ -TVRF-rGG scheme is **unf-prand**-secure if no PPT adversary  $\mathcal{A}$  wins game **unf-prand** as described below with more than negligible advantage. We define  $\mathcal{A}$ 's advantage in game **unf-prand** as

$$\begin{aligned} \text{Adv}^{\mathcal{A}} := & \Pr[\mathbf{unf-prand}_{\text{TVRF-rGG}}^{\mathcal{A}} = 1 | \mathbf{Case 1}] \cdot \Pr[\mathbf{Case 1}] \\ & + \left( \Pr[\mathbf{unf-prand}_{\text{TVRF-rGG}}^{\mathcal{A}} = 1 | \mathbf{Case 2}] - \frac{1}{2} \right) \cdot \Pr[\mathbf{Case 2}], \end{aligned}$$

where  $\Pr[\mathbf{Case 1}]$  and  $\Pr[\mathbf{Case 2}]$  denote the probabilities that  $\mathcal{A}$  tries to win game **unf-prand** via **Case 1** or **Case 2** respectively.

#### Game **unf-prand**<sub>TVRF-rGG</sub>:

- The adversary  $\mathcal{A}$  outputs a list of corrupted parties  $C$ , such that  $|C| \leq t$  and for all  $i \in C$  it holds that  $i \in [n]$ .
- The game initializes  $\text{SigList} \leftarrow \{\epsilon\}$ ,  $\text{RList} \leftarrow \{\epsilon\}$  and  $\text{EvalList} \leftarrow \{\epsilon\}$  and executes  $(\text{pk}, \{\text{sk}_1, \dots, \text{sk}_n\}) \leftarrow \text{TVRF-rGG.Gen}(1^\kappa, t, n)$ .  $\mathcal{A}$  receives as input  $\text{pk}$  and  $\{\text{sk}_i\}_{i \in C}$ .
- The adversary obtains access to the following oracles:
  - **Rand**: Same as in game **th-ufcma-hrk1**<sub>rGG</sub>.
  - **RSign**: Same as in game **th-ufcma-hrk1**<sub>rGG</sub>.

- **REval**: On input message  $m$ , index  $i \in [n] \setminus C$  and randomness  $\rho$ , check if  $\rho \in \text{RList}$  and abort otherwise. The oracle executes

$$\begin{aligned} (\text{pk}', \text{sk}'_i) &\leftarrow (\text{TVRF-rGG.RandPK}(\text{pk}, \rho), \text{TVRF-rGG.RandSK}(i, \text{sk}_i, \rho)), \\ (\phi_i, \pi_i) &\leftarrow \text{TVRF-rGG.PEval}(m, \text{sk}'_i, \text{pk}') \end{aligned}$$

and if  $(i, m, \rho) \notin \text{EvalList}$ , stores the tuple  $(i, m, \rho)$  in  $\text{EvalList}$ . The oracle returns  $(\phi_i, \pi_i)$ .

- The adversary wins the game if it wins either of the following cases:

- **Case 1**: Same as in game **th-ufcma-hrk1**<sub>rGG</sub>.
- **Case 2**: The adversary outputs a message  $m^*$ , a randomness  $\rho^*$ , a set of indices  $\mathcal{S}$  with  $|\mathcal{S}| > t$  and evaluation shares  $\{\phi_k, \pi_k\}_{k \in \mathcal{S} \cap C}$ . The game checks if there are less than  $t - |\mathcal{S} \cap C|$  tuples of the form  $(\cdot, m^*, \rho^*)$  in  $\text{EvalList}$  and if  $\rho^* \in \text{RList}$ . If so, for  $i \in \mathcal{S} \setminus C$  the game computes

$$\begin{aligned} (\text{pk}', \text{sk}'_i) &\leftarrow (\text{TVRF-rGG.RandPK}(\text{pk}, \rho^*), \text{TVRF-rGG.RandSK}(i, \text{sk}_i, \rho^*)), \\ (\phi_i, \pi_i) &\leftarrow \text{TVRF-rGG.PEval}(m^*, \text{sk}'_i, \text{pk}') \\ (\phi, \pi) &\leftarrow \text{TVRF-rGG.Combine}(\text{pk}', \mathcal{S}, \{(\phi_j, \pi_j)\}_{j \in \mathcal{S}}). \end{aligned}$$

If  $\phi = \perp$ , the game returns  $\phi$ . Otherwise the game chooses a bit  $b \stackrel{\$}{\leftarrow} \{0, 1\}$  and does the following:

- \* If  $b = 0$ : Return  $\phi$ .
- \* If  $b = 1$ : Sample  $\phi' \stackrel{\$}{\leftarrow} \mathbb{G}$  and output  $\phi'$ .

The adversary then outputs a bit  $b'$  and wins if  $b = b'$ .

**Theorem 5.2** *Let  $H_0 : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ ,  $H_1 : \{0, 1\}^* \rightarrow \mathbb{G}$  be hash functions modeled as a random oracle. Let  $\text{rGG}[H_0]$  be the **th-ufcma-hrk1**-secure interactive  $(t, n)$ -threshold ECDSA scheme with rerandomizable keys from Section 4.2 and let  $\text{TVRF}[H_1]$  be the **th-prand**-secure  $(t, n)$ -threshold verifiable random function as described in Appendix A.2. Further, let PKE be a semantically secure linearly homomorphic encryption scheme, ZK and DLEq as described in Appendix A.2 be non-interactive zero-knowledge proof systems, CT a non-malleable and equivocal commitment scheme and the DDH assumption hold in  $\mathbb{G}$ . Then the  $(t, n)$ -TVRF-rGG[ $H_0, H_1$ ] scheme as described above is **unf-prand**-secure.*

*Proof.* In order to prove this theorem we provide a reduction either to the one-more unforgeability of the rGG scheme, i.e., to the **th-ufcma-hrk1** security of rGG or to the pseudorandomness property of the TVRF scheme, i.e., to the **th-prand** security of TVRF. In other words, we show that if a PPT adversary  $\mathcal{A}$  is able to win the **unf-prand**<sub>TVRF-rGG</sub> game with more than negligible advantage, then we can construct a PPT adversary  $\mathcal{B}$  which can either win the **th-ufcma-hrk1**<sub>rGG</sub> or the **th-prand**<sub>TVRF</sub> game with more than negligible advantage. To this end,  $\mathcal{B}$  first guesses if  $\mathcal{A}$  is going to win game **unf-prand**<sub>TVRF-rGG</sub> through **Case 1** or **Case 2**. Depending on the guess,  $\mathcal{B}$  decides to either play in game **th-ufcma-hrk1**<sub>rGG</sub> or **th-prand**<sub>TVRF</sub>, while simulating  $\mathcal{A}$ 's oracle queries. Note that  $\mathcal{B}$  receives only access to the oracles of either game **th-ufcma-hrk1**<sub>rGG</sub> or **th-prand**<sub>TVRF</sub> which significantly complicates the simulation of  $\mathcal{A}$ 's oracle queries. In particular, when playing in game **th-prand**<sub>TVRF</sub>,  $\mathcal{B}$  does not get access to a signing oracle, yet has to simulate oracle **RSign** of game **unf-prand**<sub>TVRF-rGG</sub> to  $\mathcal{A}$ .

Let  $\mathcal{B} := (\mathcal{B}_0, \mathcal{B}_1)$  be composed of two subprocedures. At the beginning of game **unf-prand**<sub>TVRF-rGG</sub>,  $\mathcal{B}$  chooses a bit  $b \stackrel{\$}{\leftarrow} \{0, 1\}$ . If  $b = 0$ ,  $\mathcal{B}$  executes subprocedure  $\mathcal{B}_0$  that plays in game **th-ufcma-hrk1**<sub>rGG</sub> and otherwise  $\mathcal{B}$  executes  $\mathcal{B}_1$  that plays in game **th-prand**<sub>TVRF</sub>. In the following, we show for both cases (i.e.,  $b = 0$  and  $b = 1$ ) that the respective subprocedure  $\mathcal{B}_b$  can simulate game **unf-prand**<sub>TVRF-rGG</sub> to  $\mathcal{A}$  and use  $\mathcal{A}$ 's output to win their respective security games (i.e., either **th-ufcma-hrk1**<sub>rGG</sub> or **th-prand**<sub>TVRF</sub>). Finally, after analyzing both cases separately, we determine the advantage of  $\mathcal{B} := (\mathcal{B}_0, \mathcal{B}_1)$  to win either game **th-ufcma-hrk1**<sub>rGG</sub> or game **th-prand**<sub>TVRF</sub>. We additionally provide an intuitive proof sketch for both cases in Appendix B.4.

**Case  $b = 0$**  In this case we show via a series of computationally indistinguishable games that there exists an adversary  $\mathcal{B}_0$  which can use adversary  $\mathcal{A}$  in **Case 1** to win its own game **th-ufcma-hrk1**<sub>rGG</sub> <sup>$\mathcal{B}_0$</sup> . Game  $\mathcal{G}_0$ : This game is the original **unf-prand**<sub>TVRF-rGG</sub> game, in which adversary  $\mathcal{A}$  can corrupt  $t$

parties and receives access to oracles  $H_0$ ,  $H_1$ ,  $\text{RSign}$ ,  $\text{Rand}$  and  $\text{REval}$ . We have  $\Pr[\mathbf{unf-prand}_{\text{TVRF-rGG}}^A = 1 | \mathbf{Case 1}] = \Pr[\mathbf{G}_0^A = 1]$ .

Game  $\mathbf{G}_1$ : This game is similar to the previous game with two differences. First, in the beginning the game initializes a set  $\text{HList} := \epsilon$ . Second, upon  $\mathcal{A}$  sending a query to  $H_1$  on input  $m$ , if  $H_1(m) = \perp$  the game samples uniformly at random  $r \xleftarrow{\$} \mathbb{Z}_q$ , sets  $\text{HList}[m] := r$  and  $H_1(m) := g^r$ . The game outputs  $H_1(m)$ .

It is easy to see that the random oracle  $H_1$  returns uniformly random group elements since  $r$  is chosen uniformly at random from  $\mathbb{Z}_q$ . Therefore, we have that  $\Pr[\mathbf{G}_1^A = 1] = \Pr[\mathbf{G}_0^A = 1]$ .

Game  $\mathbf{G}_2$ : This game is similar to the previous game with a difference in the  $\text{REval}$  oracle. On input a message  $m$ , an index  $i$  and a randomness  $\rho$ , the game executes

$$\begin{aligned} \text{pk}' &\leftarrow \text{TVRF-rGG.RandPK}(\text{pk}, \rho), \\ \text{sk}'_i &\leftarrow \text{TVRF-rGG.RandSK}(i, \text{sk}_i, \rho), \\ (\phi_i, \pi_i) &\leftarrow \text{TVRF-rGG.PEval}(m, \text{sk}'_i, \text{pk}'). \end{aligned}$$

However, instead of outputting  $(\phi_i, \pi_i)$ , the game simulates a zero-knowledge proof  $\pi'_i$  that proves correctness of  $\phi_i$  and outputs  $(\phi_i, \pi'_i)$ .

Due to the zero-knowledge property of the proof system  $\text{DLEq}$  (cf. Appendix A.2), this game is indistinguishable from the previous one except with negligible probability. That is, we have that  $\Pr[\mathbf{G}_1^A = 1] \leq \Pr[\mathbf{G}_2^A = 1] + \text{negl}(\kappa)$  where  $\text{negl}$  is a negligible function in the security parameter  $\kappa$ .

Game  $\mathbf{G}_3$ : This game is similar to the previous game with a difference in the  $\text{REval}$  oracle. On input a message  $m$ , an index  $i$ , and a randomness  $\rho$ , the game checks if  $\text{HList}[m] = \perp$ . If so, it queries  $H_1(m)$ . It then executes  $\text{pk}' \leftarrow \text{TVRF-rGG.RandPK}(\text{pk}, \rho)$ , parses  $\text{pk}' := (X', \{X'_1, \dots, X'_n\})$  and computes  $\phi_i \leftarrow (X'_i)^r$  for  $r \leftarrow \text{HList}[m]$ .

This game is equivalent to the previous game since  $(X'_i)^r = g^{\text{sk}'_i \cdot r} = H_1(m)^{\text{sk}'_i}$ . Therefore, we have that  $\Pr[\mathbf{G}_2^A = 1] = \Pr[\mathbf{G}_3^A = 1]$ .

Reduction to  $\mathbf{th-ufcma-hrk1}_{\text{rGG}}$ : Having shown that the transition from game  $\mathbf{G}_0$  to game  $\mathbf{G}_3$  is indistinguishable, it remains to show that an adversary  $\mathcal{A}$  winning in game  $\mathbf{G}_3$  can be used to construct an adversary  $\mathcal{B}_0$  that wins game  $\mathbf{th-ufcma-hrk1}_{\text{rGG}}$ . To do so, we must show that  $\mathcal{B}_0$  playing in  $\mathbf{th-ufcma-hrk1}_{\text{rGG}}$  can simulate game  $\mathbf{G}_3$  to  $\mathcal{A}$ . The simulation differs from game  $\mathbf{G}_3$  in the following ways:

1.  $\mathcal{B}_0$  does not generate the secret key shares and public key, but instead corrupts the same set of parties  $\mathbf{C}$  in  $\mathbf{th-ufcma-hrk1}_{\text{rGG}}$  as  $\mathcal{A}$  does in  $\mathbf{G}_3$ .  $\mathcal{B}_0$  then forwards the public key  $\text{pk}$  and the secret key shares  $\{\text{sk}_i\}_{i \in \mathbf{C}}$  from game  $\mathbf{th-ufcma-hrk1}_{\text{rGG}}$  to  $\mathcal{A}$ .
2. Upon  $\mathcal{A}$  querying oracle  $\text{RSign}$  on input a message  $m$  and a randomness  $\rho$ ,  $\mathcal{B}_0$  queries its own oracle  $\text{RSign}$  on input  $m$  and  $\rho$  and relays the messages between  $\mathcal{A}$  and the  $\text{RSign}$  oracle in game  $\mathbf{th-ufcma-hrk1}_{\text{rGG}}$ .
3. Upon  $\mathcal{A}$  querying oracle  $H_0$  on input a message  $m$ ,  $\mathcal{B}_0$  forwards the query to its own random oracle and relays the output.

It is easy to see that  $\mathcal{B}_0$ 's simulation is indistinguishable from game  $\mathbf{G}_3$  to  $\mathcal{A}$ . It remains to show that  $\mathcal{B}_0$  can use  $\mathcal{A}$ 's forgery in game  $\mathbf{unf-prand}_{\text{TVRF-rGG}}$  to win its own game  $\mathbf{th-ufcma-hrk1}_{\text{rGG}}$ . Since  $\mathcal{B}_0$  forwards all queries to  $\text{RSign}$  in game  $\mathbf{unf-prand}_{\text{TVRF-rGG}}$  to the corresponding oracle in game  $\mathbf{th-ufcma-hrk1}_{\text{rGG}}$ ,  $\mathcal{B}_0$  and  $\mathcal{A}$  query their respective oracles on the same messages. Therefore, if  $\mathcal{A}$  outputs a valid forgery in  $\mathbf{unf-prand}_{\text{TVRF-rGG}}$ , then the forgery is also valid in  $\mathbf{th-ufcma-hrk1}_{\text{rGG}}$ . We finally have

$$\begin{aligned} \Pr[\mathbf{unf-prand}_{\text{TVRF-rGG}}^A = 1 | \mathbf{Case 1}] &= \Pr[\mathbf{G}_0^A = 1] \leq \Pr[\mathbf{G}_3^A = 1] + \text{negl}(\kappa) \\ &= \Pr[\mathbf{th-ufcma-hrk1}_{\text{rGG}[\mathbf{H}_0]}^{\mathcal{B}_0} = 1] + \text{negl}(\kappa) \\ &= \text{Adv}_{\mathbf{th-ufcma-hrk1}_{\text{rGG}[\mathbf{H}_0]}}^{\mathcal{B}_0} + \text{negl}(\kappa). \end{aligned}$$

**Case  $b = 1$**  We now show via a series of computationally indistinguishable games that there exists an adversary  $\mathcal{B}_1$  which can use adversary  $\mathcal{A}$  in **Case 2** to win its own game **th-prand** $_{\text{TVRF}}^{\mathcal{B}_1}$ .

**Game  $\mathbf{G}_0$ :** This game is the original **unf-prand** $_{\text{TVRF-rGG}}$  game, in which adversary  $\mathcal{A}$  can corrupt  $t$  parties and receives access to oracles  $\mathbf{H}_0$ ,  $\mathbf{H}_1$ , **RSign**, **Rand** and **REval**. We have  $\Pr[\mathbf{unf-prand}_{\text{TVRF-rGG}}^{\mathcal{A}} = 1 | \mathbf{Case 2}] = \Pr[\mathbf{G}_0^{\mathcal{A}} = 1]$ .

**Game  $\mathbf{G}_1$ :** This game is similar to the previous game with two differences. First, in the beginning the game initializes a set  $\mathbf{HSigs} := \epsilon$ . Second, upon a query to  $\mathbf{H}_0$  on input a public key prefixed message  $m := (\mathbf{pk}', m')$ , i.e., where  $(\mathbf{pk}', \cdot) \in \text{TVRF-rGG.Gen}(1^\kappa, t, n)$ , the game checks if  $\mathbf{H}_0(m) = \perp$ . If so, it executes  $\mathcal{S}_{\text{ECDSA}}$  as described in Figure 4 on input  $(X', m)$  where  $\mathbf{pk}' := (X', \{X'_1, \dots, X'_n\})$ . Finally, the game sets  $\mathbf{HSigs}[m] := (r, s)$ .

$$\boxed{\begin{array}{l} \mathcal{S}_{\text{ECDSA}}(X, m) : \\ a, b \xleftarrow{\$} \mathbb{Z}_q, R = X^a \cdot g^b, r = f(R), s = \frac{r}{a}, \mathbf{H}_0(m) := \frac{r \cdot b}{a} \end{array}}$$

Figure 4: Simulation of ECDSA signatures via programming of the random oracle  $\mathbf{H}_0$  as first presented by Fersch et al. [FKP17]. The function  $f : \mathbb{G} \rightarrow \mathbb{Z}_q$  is defined as the projection of a group element to its x-coordinate.

It is easy to see that  $\mathcal{S}_{\text{ECDSA}}$  programs the random oracle  $\mathbf{H}_0$  in such a way that  $\mathbf{H}_0$  returns uniformly random values. Therefore, this game is equivalent to the previous game, i.e.,  $\Pr[\mathbf{G}_0^{\mathcal{A}} = 1] = \Pr[\mathbf{G}_1^{\mathcal{A}} = 1]$ .

**Game  $\mathbf{G}_2$ :** This game is similar to the previous game with a difference in the **RSign** oracle. On input a message  $m$  and a randomness  $\rho$ , the game first computes the rerandomized (full) secret key  $\mathbf{sk}'$  and then generates a full ECDSA signature  $\sigma'$  under  $\mathbf{sk}'$  for message  $m$ . The game then executes the signing procedure in the same way as presented in the proof sketch of Theorem 4.4 using signature  $\sigma'$ .

The indistinguishability of this game to the previous one follows in the same way as in Theorem 4.4. Note that the simulation of the signing procedure as described in the proof sketch of Theorem 4.4 does not program  $\mathbf{H}_0$  and therefore does not conflict with the execution of  $\mathcal{S}_{\text{ECDSA}}$ . We have that  $\Pr[\mathbf{G}_1^{\mathcal{A}} = 1] \leq \Pr[\mathbf{G}_2^{\mathcal{A}} = 1] + \text{negl}_1(\kappa)$  where  $\text{negl}_1$  is a negligible function in the security parameter  $\kappa$ .

**Game  $\mathbf{G}_3$ :** This game is similar to the previous game again with a modification in the **RSign** oracle. On input a message  $m$  and a randomness  $\rho$ , the game does not generate a full ECDSA signature using  $\mathbf{sk}'$ , but it fetches  $(r, s) \leftarrow \mathbf{HSigs}[m']$  for  $m' \leftarrow (\mathbf{pk}', m)$  where  $\mathbf{pk}' \leftarrow \text{TVRF-rGG.RandPK}(\mathbf{pk}, \rho)$ .<sup>8</sup> The game then uses the tuple  $(r, s)$  as the full ECDSA signature under  $\mathbf{sk}'$ .

As shown in [FKP17], the tuple  $(r, s)$  as generated by the  $\mathcal{S}_{\text{ECDSA}}$  algorithm (see Figure 4) is computationally indistinguishable from an honestly generated ECDSA signature for message  $m'$  under public key  $X'$  (where  $\mathbf{pk}' := (X', \{X'_1, \dots, X'_n\})$ ) to a PPT adversary  $\mathcal{A}$  with access to random oracle  $\mathbf{H}_0$ . Since the simulation of the signing procedure as described in the proof sketch of Theorem 4.4 forces the execution of the **RSign** oracle to output  $(r, s)$ , adversary  $\mathcal{A}$  can distinguish this game from the previous one only with negligible probability. Therefore, we have that  $\Pr[\mathbf{G}_2^{\mathcal{A}} = 1] \leq \Pr[\mathbf{G}_3^{\mathcal{A}} = 1] + \text{negl}_2(\kappa)$  where  $\text{negl}_2$  is a negligible function in the security parameter  $\kappa$ .

**Game  $\mathbf{G}_4$ :** This game is similar to the previous game with a modification in the **REval** oracle. On input a message  $m$ , an index  $i$  and a randomness  $\rho$ , the game computes  $\mathbf{sk}'_i \leftarrow \text{TVRF-rGG.RandSK}(i, \mathbf{sk}_i, \rho)$  and  $\mathbf{pk}' \leftarrow \text{TVRF-rGG.RandPK}(\mathbf{pk}, \rho)$  and executes  $(\phi_i, \pi_i) \leftarrow \text{TVRF-rGG.PEval}(m, \mathbf{sk}'_i, \mathbf{pk}')$ . Instead of outputting  $(\phi_i, \pi_i)$ , however, the game simulates a zero-knowledge proof  $\pi'_i$  that proves correctness of  $\phi_i$ . The game then outputs  $(\phi_i, \pi'_i)$ .

Due to the zero-knowledge property of the proof system **DLEq** (cf. Appendix A.2), this game is indistinguishable from the previous one except with negligible probability. Therefore, we have that  $\Pr[\mathbf{G}_3^{\mathcal{A}} = 1] \leq \Pr[\mathbf{G}_4^{\mathcal{A}} = 1] + \text{negl}_3(\kappa)$  where  $\text{negl}_3$  is a negligible function in the security parameter  $\kappa$ .

**Game  $\mathbf{G}_5$ :** This game is similar to the previous game with a modification in the **REval** oracle. On input a message  $m$ , an index  $i$ , and a randomness  $\rho$ , instead of rerandomizing the secret key share  $\mathbf{sk}_i$  to

<sup>8</sup>We assume that  $\mathbf{H}_0$  has been queried on  $m'$  before the signing query.

$sk'_i$  and executing  $\text{TVRF-rGG.PEval}(m, sk'_i, pk')$ , the game computes

$$\text{TVRF-rGG.PEval}(m, sk_i, pk) \cdot H_1(m)^{\rho_i},$$

where  $\rho_i$  denotes the randomness share of  $\rho$  for party  $P_i$  according to the  $\text{TVRF-rGG.RandSK}$  (cf. Figure 2) algorithm.

This game is equivalent to the previous game since for  $sk'_i \leftarrow \text{TVRF-rGG.RandSK}(i, sk_i, \rho)$  and  $pk' \leftarrow \text{TVRF-rGG.RandPK}(pk, \rho)$  it holds that:

$$\text{TVRF-rGG.PEval}(m, sk'_i, pk') = \text{TVRF-rGG.PEval}(m, sk_i, pk) \cdot H_1(m)^{\rho_i} = H_1(m)^{sk_i + \rho_i}.$$

Therefore, we have that  $\Pr[\mathbf{G}_4^A = 1] = \Pr[\mathbf{G}_5^A = 1]$ .

**Game  $\mathbf{G}_6$ :** This game is similar to the previous game with a modification in the challenge phase. Upon  $\mathcal{A}$  outputting a message  $m^*$ , randomness  $\rho^*$ , a set of indices  $\mathcal{S}$ , and evaluation shares  $\{(\phi_k^*, \pi_k^*)\}_{k \in \mathcal{S} \cap \mathcal{C}}$ , the game verifies the proofs  $\pi_k^*$  and returns  $\perp$  if any proof does not verify. Otherwise the game checks if  $\phi_k^* = H_1(m^*)^{sk_k + \rho_k^*}$  and aborts if any of these checks does not hold.

Note that the only way that the game aborts is if the adversary manages to submit a verifying zero-knowledge proof  $\pi_k^*$  for a false statement. Due to the soundness property of the  $\text{DLEq}$  proof system, this event happens only with negligible probability. Therefore, we have that  $\Pr[\mathbf{G}_5^A = 1] \leq \Pr[\mathbf{G}_6^A = 1] + \text{negl}_4(\kappa)$  where  $\text{negl}_4$  is a negligible function in the security parameter  $\kappa$ .

**Game  $\mathbf{G}_7$ :** This game is similar to the previous game with a modification in the challenge phase. Namely, upon  $\mathcal{A}$  outputting a message  $m^*$ , randomness  $\rho^*$ , a set of indices  $\mathcal{S}$ , and evaluation shares  $\{(\phi_k^*, \pi_k^*)\}_{k \in \mathcal{S} \cap \mathcal{C}}$  the game computes  $\phi_k = \phi_k^* \cdot H_1(m^*)^{-\rho_k^*} = H_1(m^*)^{sk_k}$  and generates a new zero-knowledge proof  $\pi_k$  using  $sk_k$  and  $\phi_k$ . The game then computes  $(\phi_i, \pi_i) \leftarrow \text{TVRF-rGG.PEval}(m^*, sk_i, pk)$  for  $i \in \mathcal{S} \setminus \mathcal{C}$  and  $(\phi, \pi) \leftarrow \text{TVRF-rGG.Combine}(pk, \mathcal{S}, \{(\phi_j, \pi_j)\}_{j \in \mathcal{S}})$ . The game chooses uniformly at random  $b \xleftarrow{\$} \{0, 1\}$  and if  $b = 0$  sets  $\phi' := \phi$  and otherwise sets  $\phi' \xleftarrow{\$} \mathbb{G}$ . Finally, the game computes  $\phi^* = \phi' \cdot H_1(m^*)^{\rho^*}$  and returns it to  $\mathcal{A}$ .

Note that if  $\phi'$  was chosen randomly from  $\mathbb{G}$  by the game then  $\phi^*$  is also a uniformly random element, and if  $\phi'$  is a valid TVRF output, then so is  $\phi^*$  under the key randomized with  $\rho^*$ . This is since  $\phi^* = \phi' \cdot H_1(m^*)^{\rho^*} = H_1(m^*)^{sk} \cdot H_1(m^*)^{\rho^*} = H_1(m^*)^{sk + \rho^*}$ . We have that  $\Pr[\mathbf{G}_6^A = 1] = \Pr[\mathbf{G}_7^A = 1]$ .

**Reduction to  $\mathbf{th-prand}_{\text{TVRF}}$ :** Having shown that the transition from game  $\mathbf{G}_0$  to game  $\mathbf{G}_7$  is indistinguishable, it remains to show that an adversary  $\mathcal{A}$  winning in game  $\mathbf{G}_7$  can be used to construct an adversary  $\mathcal{B}_1$  that wins game  $\mathbf{th-prand}_{\text{TVRF}}$ . To do so, we must show that  $\mathcal{B}_1$  playing in  $\mathbf{th-prand}_{\text{TVRF}}$  can simulate game  $\mathbf{G}_7$  to  $\mathcal{A}$ . The simulation differs from game  $\mathbf{G}_7$  in the following points:

1.  $\mathcal{B}_1$  does not generate the secret key shares and public key, but instead corrupts the same set of parties  $\mathcal{C}$  in  $\mathbf{th-prand}_{\text{TVRF}}$  as  $\mathcal{A}$  does in  $\mathbf{G}_7$ .  $\mathcal{B}_1$  then forwards the public key  $pk$  and the secret key shares  $\{sk_i\}_{i \in \mathcal{C}}$  from game  $\mathbf{th-prand}_{\text{TVRF}}$  to  $\mathcal{A}$ .
2. Upon  $\mathcal{A}$  querying oracle  $\text{REval}$  on input a message  $m$ , an index  $i$  and a randomness  $\rho$ ,  $\mathcal{B}_1$  queries its own oracle  $\text{Eval}$  on input  $m$  and  $i$  and uses the oracle output to compute the output of  $\text{REval}$  as in  $\mathbf{G}_7$ .
3. Upon  $\mathcal{A}$  querying oracle  $H_1$  on input a message  $m$ ,  $\mathcal{B}_1$  forwards the query to its own random oracle and relays the output.
4. During the challenge phase,  $\mathcal{B}_1$  sends the shares  $\phi_k = \phi_k^* \cdot H_1(m^*)^{-\rho_k^*}$  together with the zero-knowledge proofs  $\pi_k$  to its own game and receives an element  $\phi^*$ .  $\mathcal{B}_1$  forwards to  $\mathcal{A}$  the element  $\phi^* \cdot H_1(m^*)^{\rho^*}$ .

It is easy to see that  $\mathcal{B}_1$ 's simulation is indistinguishable from game  $\mathbf{G}_7$  to  $\mathcal{A}$  and that if  $\mathcal{A}$  wins game  $\mathbf{G}_7$  with more than negligible probability, then  $\mathcal{B}_1$  wins game  $\mathbf{th-prand}_{\text{TVRF}}$  with the same probability. The latter is because  $\mathcal{B}_1$  makes the same queries to oracle  $\text{Eval}$  as  $\mathcal{A}$  does to oracle  $\text{REval}$ . We finally have that

$$\begin{aligned} \Pr[\mathbf{unf-prand}_{\text{TVRF-rGG}}^A = 1 | \mathbf{Case 2}] &= \Pr[\mathbf{G}_0^A = 1] \leq \Pr[\mathbf{G}_7^A = 1] + \text{negl}'(\kappa) \\ &= \Pr[\mathbf{th-prand}_{\text{TVRF}[H_1]}^{\mathcal{B}_1} = 1] + \text{negl}'(\kappa), \end{aligned}$$

where  $\text{negl}'(\kappa) := \sum_{i=1}^4 \text{negl}_i(\kappa)$ .

Finally, we determine the advantage of adversary  $\mathcal{B} := (\mathcal{B}_0, \mathcal{B}_1)$  to win either in game **th-ufcma-hrk1**<sub>rGG</sub> or **th-prand**<sub>TVRF</sub>. Note that  $\mathcal{B}$ 's advantage is:

$$\text{Adv}^{\mathcal{B}} := \frac{1}{2} \text{Adv}_{\text{uf-cma-hrk1}_{\text{ECDSA}[H_0]}}^{\mathcal{B}_0} \cdot \Pr[\text{Case 1}] + \frac{1}{2} \text{Adv}_{\text{th-prand}_{\text{TVRF}[H_1]}}^{\mathcal{B}_1} \cdot \Pr[\text{Case 2}].$$

Therefore we can conclude that:

$$\begin{aligned} \text{Adv}^{\mathcal{A}} &:= \Pr[\text{unf-prand}_{\text{TVRF-rGG}}^{\mathcal{A}} = 1 | \text{Case 1}] \cdot \Pr[\text{Case 1}] \\ &\quad + (\Pr[\text{unf-prand}_{\text{TVRF-rGG}}^{\mathcal{A}} = 1 | \text{Case 2}] - \frac{1}{2}) \cdot \Pr[\text{Case 2}] \\ &\leq (\Pr[\text{th-ufcma-hrk1}_{\text{rGG}[H_0]}^{\mathcal{B}_0} = 1] + \text{negl}(\kappa)) \cdot \Pr[\text{Case 1}] \\ &\quad + (\Pr[\text{th-prand}_{\text{TVRF}[H_1]}^{\mathcal{B}_1} = 1] + \text{negl}'(\kappa) - \frac{1}{2}) \cdot \Pr[\text{Case 2}] \\ &\leq (\text{Adv}_{\text{uf-cma-hrk1}_{\text{ECDSA}[H_0]}}^{\mathcal{B}_0} + \text{negl}(\kappa)) \cdot \Pr[\text{Case 1}] \\ &\quad + (\text{Adv}_{\text{th-prand}_{\text{TVRF}[H_1]}}^{\mathcal{B}_1} + \text{negl}'(\kappa)) \cdot \Pr[\text{Case 2}] \\ &= 2 \cdot \text{Adv}^{\mathcal{B}} + \text{negl}''(\kappa) \end{aligned}$$

where  $\text{negl}''(\kappa) := \text{negl}(\kappa) \cdot \Pr[\text{Case 1}] + \text{negl}'(\kappa) \cdot \Pr[\text{Case 2}]$  is a negligible function in  $\kappa$ . ■

### 5.3.2 Uniqueness and Robustness of TVRF-rGG

Besides pseudorandomness, the TVRF-rGG scheme must additionally satisfy the properties of uniqueness and robustness, which are defined in a similar manner as the pseudorandomness property in the sense that they combine the respective property of the TVRF scheme with the one-more unforgeability of our rGG scheme. In the following we provide the formal definitions of these two properties.

**Definition 5.3** (Uniqueness of TVRF-rGG). The  $(t, n)$ -TVRF-rGG scheme is unique if no PPT adversary  $\mathcal{A}$  wins game **unf-unique** as described below with more than negligible advantage. We define  $\mathcal{A}$ 's advantage in game **unf-unique**<sub>TVRF-rGG</sub> as  $\text{Adv}_{\text{unf-unique}}^{\mathcal{A}} := \Pr[\text{unf-unique}_{\text{TVRF-rGG}}^{\mathcal{A}} = 1]$ .

Game **unf-unique**<sub>TVRF-rGG</sub>:

- The adversary  $\mathcal{A}$  outputs a list of corrupted parties  $\mathcal{C}$ , such that  $|\mathcal{C}| \leq t$  and for all  $i \in \mathcal{C}$  it holds that  $i \in [n]$ .
- The game initializes  $\text{SigList} \leftarrow \{\epsilon\}$  and  $\text{RList} \leftarrow \{\epsilon\}$  and executes  $(\text{pk}, \{\text{sk}_1, \dots, \text{sk}_n\}) \leftarrow \text{TVRF-rGG.Gen}(1^\kappa, t, n)$ . Then  $\mathcal{A}$  is run on input  $\text{pk}$  and  $\{\text{sk}_i\}_{i \in \mathcal{C}}$ .
- The adversary obtains access to the following oracles:
  - **Rand**: Same as in game **unf-prand**<sub>TVRF-rGG</sub>.
  - **RSign**: Same as in game **unf-prand**<sub>TVRF-rGG</sub>.
  - **REval**: Same as in game **unf-prand**<sub>TVRF-rGG</sub>.
  - **KeyLeak**: On input  $i \in [n]$ , the oracle outputs  $\text{sk}_i$ .
- The adversary wins the game if it wins either of the following cases:
  - **Case 1**: Output 0 if there has been any query to oracle **KeyLeak**. Otherwise this case is the same as **Case 1** in game **unf-prand**<sub>TVRF-rGG</sub>.
  - **Case 2**: The adversary outputs a message  $m^*$ , a randomness  $\rho^*$  and evaluations  $\{(\phi^{i^*}, \pi^{i^*})\}_{i \in \{0,1\}}$ . If  $\rho^* \in \text{RList}$ , the game computes  $\text{pk}' \leftarrow \text{TVRF-rGG.RandPK}(\text{pk}, \rho^*)$ . The game outputs 1 if  $\phi^{0^*} \neq \phi^{1^*}$  and

$$\text{TVRF-rGG.Verify}(\text{pk}', m^*, \phi^{0^*}, \pi^{0^*}) = \text{TVRF-rGG.Verify}(\text{pk}', m^*, \phi^{1^*}, \pi^{1^*}) = 1.$$

Otherwise it outputs 0.

**Definition 5.4** (Robustness of TVRF-rGG). The  $(t, n)$ -TVRF-rGG scheme is robust if no PPT adversary  $\mathcal{A}$  wins game **unf-robust** as described below with more than negligible advantage. We define  $\mathcal{A}$ 's advantage in game **unf-robust**<sub>TVRF-rGG</sub> as  $\text{Adv}_{\text{unf-robust}}^{\mathcal{A}} := \Pr[\text{unf-robust}_{\text{TVRF-rGG}}^{\mathcal{A}} = 1]$ .

Game **unf-robust**<sub>TVRF-rGG</sub>:

- The game is exactly the same as game **unf-prand**<sub>TVRF-rGG</sub>, except for the winning conditions, which we will describe below.
- The adversary wins the game if it wins either of the following cases:
  - **Case 1:** Same as **Case 1** in game **unf-prand**<sub>TVRF-rGG</sub>.
  - **Case 2:** The adversary outputs a message  $m^*$ , a set  $\mathcal{S}$  with  $|\mathcal{S}| > t$ , a list of evaluation shares  $\{\phi_i, \pi_i\}_{i \in \mathcal{S} \cap \mathcal{C}}$  and a randomness  $\rho^*$ . The game checks if  $\rho^* \in \text{RList}$  and if so computes for all  $i \in \mathcal{S} \setminus \mathcal{C}$ :

$$\begin{aligned} (\text{pk}', \text{sk}'_i) &\leftarrow (\text{TVRF-rGG.RandPK}(\text{pk}, \rho^*), \text{TVRF-rGG.RandSK}(i, \text{sk}_i, \rho^*)), \\ (\phi_i, \pi_i) &\leftarrow \text{TVRF-rGG.PEval}(m^*, \text{sk}'_i, \text{pk}') \end{aligned}$$

The game finally sets

$$(\phi^*, \pi^*) \leftarrow \text{TVRF-rGG.Combine}(\text{pk}', m^*, \mathcal{S}, \{\phi_i, \pi_i\}_{i \in \mathcal{S}}).$$

If  $\phi^* \neq \perp$  and  $\text{TVRF-rGG.Verify}_{\text{TVRF}}(\text{pk}', m^*, \phi^*, \pi^*) = 0$ , the game outputs 1 and 0 otherwise.

The proof of the uniqueness and robustness property of the TVRF-rGG scheme is similar to the proof of the pseudorandomness property in the sense that the reduction guesses whether the adversary is going to win in **Case 1** or **Case 2**. In **Case 1**, we reduce to the one-more unforgeability of rGG. The simulation of the respective security game works in the same way as in the proof of Theorem 5.2. In **Case 2**, we can show a contradiction to the soundness property of the NIZK proof system DLEq (cf. Appendix A.2) in the same way as was previously shown by Galindo et al. [GLOW21]). The simulation of the RSign oracle is then straightforward since the reduction can choose the initial key set itself.

## 5.4 Constructing BIP32-Compatible Threshold Wallets from TVRF-rGG

Now, we describe at a high level how our joint primitive TVRF-rGG can be used to construct BIP32-compatible threshold wallets. We illustrate the design of such wallets in Figure 5. Initially, to set up the wallet, a user generates the master secret and public key shares by executing TVRF-rGG.Gen, and also samples a uniformly random string serving as the wallet's initial chaincode. The root node can then deterministically derive either non-hardened or hardened nodes. That is, in order to derive a non-hardened node each root device first locally executes TVRF-rGG.RandSK and TVRF-rGG.RandPK to rerandomize its secret key share and the root node's public key using randomness derived from the chaincode. All root and child node devices then engage in a DPSS protocol to securely send a re-sharing of the rerandomized secret key to the child node. To derive a hardened node, the root node devices execute TVRF-rGG.PEval to locally compute randomness shares from which the hardened node computes a random value (via TVRF-rGG.Combine and TVRF-rGG.Verify<sub>TVRF</sub>). The hardened node device then uses this random value as input to the ECDSA key generation to generate its signing key pair. Naturally, the root node and all non-hardened child nodes can execute the procedure TVRF-rGG.TSign to compute signatures.

**Choosing the Parameters  $t$  and  $n$ .** The choice of the parameters  $t$  and  $n$  depends on the specific use case and the corresponding security requirements. For instance, for a multi-device personal wallet, it might be sufficient to use  $t = 2$  and  $n = 5$ , whereas for enterprise-grade wallets it might be necessary to use a larger number of devices, e.g.,  $t = 31$  and  $n = 64$ . We show in the following section that our solution remains concretely efficient even for such larger parameters.

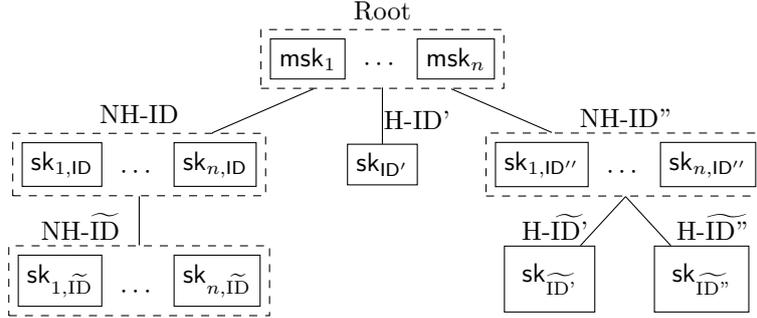


Figure 5: Illustration of our BIP32-compatible threshold wallet. Non-hardened nodes (NH) and the root node are thresholdized, whereas hardened nodes (H) are single devices. For simplicity, this figure abstracts away the chain code and public key that each node additionally stores.

## 6 Evaluation

In this section, we evaluate the hardened node derivation in the threshold setting as outlined previously. Concretely, we compare our TVRF-based derivation mechanism to the standard BIP32 hardened node derivation, which requires the distributed computation of the SHA-512 hash function. We evaluate both approaches w.r.t. running time and communication cost for different parameters  $n$  and  $t$ , and we always assume the worst-case corruption of  $t = \frac{n}{2} - 1$ . In the following, we refer to the original BIP32 hardened derivation mechanism in the threshold setting as MPC-based, since it requires computing SHA-512 via a generic MPC protocol.

We note that we do not evaluate the non-hardened derivation mechanism in the threshold setting since (1) our non-hardened derivation mechanism follows the exact specification of the BIP32 standard, and (2) the reuse of the TVRF and ECDSA key pair ensures that our TVRF-based solution does not introduce any communication or storage overhead for the non-hardened derivation as compared to the MPC-based solution.

The evaluation was done on a single machine with an Apple M3 Max CPU (14 cores), 36 GB of memory, and macOS 14.4.1. In order to account for network latency, we included a delay of 10 ms per communication round, which corresponds approximately to the speed of a fast LAN connection. The source code of our implementation is publicly available.<sup>9</sup> We implemented the TVRF-based hardened derivation in Go 1.21.0 and we parallelized the local computation of each device. For the MPC-based solution, we employed MP-SPDZ<sup>10</sup> [Kel20]. We choose MP-SPDZ since it is one of the most advanced, well documented and actively maintained open source frameworks for MPC, and is the de-facto standard for benchmarking MPC protocols [BDST20]. More concretely, we used `mal - shamir`, an MPC protocol based on Shamir’s secret sharing with malicious security in the  $t \leq \frac{n}{2} - 1$  setting.

(t,n)	TVRF	MPC	MPC No Delay
(2, 5)	22.12	68694.51	1404.51
(4, 10)	33.04	73998.3	6268.3
(7, 16)	47.76	81961	14571
(15, 32)	91.71	133070.1	65780.1
(31, 64)	202.23	413021	346731

Table 1: Running times in milliseconds of the TVRF-based and MPC-based hardened derivation. The columns TVRF and MPC contain a 10 ms network delay per communication round. For reference, the table also contains a column for the MPC-based solution without any network delay.

The results of the evaluation are shown in Table 1 and Table 2, respectively. Table 1 demonstrates the running time for the hardened derivation using either the TVRF-based or the MPC-based approach. In both approaches, we assumed a network delay of 10 ms, but since the MPC-based approach requires a

<sup>9</sup>[https://github.com/AppliedCryptoGroup/BIP32\\_Threshold\\_Wallets](https://github.com/AppliedCryptoGroup/BIP32_Threshold_Wallets)

<sup>10</sup><https://github.com/data61/MP-SPDZ>

significant number of communication rounds, we also include its runtime without any network delay for reference. While our TVRF-based solution takes 22.12 ms for a 2-out-of-5 setting with network delay, the MPC-based solution requires already 68.69 seconds. As the number of parties increases, the TVRF-based derivation scales mostly linearly, while the MPC-based solution scales roughly quadratically. Table 1 clearly demonstrates that the running time of our TVRF-based solution is faster by several orders of magnitude. Since the generic MPC-based derivation requires more than 6000 rounds of communication, the network delay has a significant impact on the runtime. We note that the technique presented by Beaver-Micali-Rogaway (BMR) [BMR90] can be used to reduce the number of communication rounds for the MPC-based approach, however, we found that using this technique within MP-SPDZ neither improves the runtime nor the communication costs of the MPC-based solution.

The communication costs are shown in Table 2, i.e., the total amount of data sent by all parties of the TVRF-based and MPC-based approaches, respectively. Clearly, our TVRF-based solution requires significantly less data to be exchanged.

(t,n)	TVRF (KB)	MPC (MB)
(2, 5)	0.64	104.59
(4, 10)	1.28	413.56
(7, 16)	2.05	1120.22
(15, 32)	4.09	4676.96
(31, 64)	8.19	19086.20

Table 2: Comparison of communication costs for the TVRF-based solution (in KB) and the MPC-based solution (in MB).

## Acknowledgments

This result is part of a project that received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 and Horizon Europe research and innovation programs (grant CRYPTOLAYER-101044770). Additionally, this work is supported by the German Research Foundation DFG - SFB 1119 - 236615297 (CROSSING Project S7), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the *National Research Center for Applied Cybersecurity ATHENE*.

## References

- [ADE<sup>+</sup>20] Nabil Alkeilani Alkadri, Poulami Das, Andreas Erwig, Sebastian Faust, Juliane Krämer, Siavash Riahi, and Patrick Struck. Deterministic wallets in a quantum world. pages 1017–1031, 2020. (Cited on page 4, 9.)
- [AGKK19] Myrto Arapinis, Andriana Gkaniatsou, Dimitris Karakostas, and Aggelos Kiayias. A formal treatment of hardware wallets. pages 426–445, 2019. (Cited on page 4.)
- [AHKP24] Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. Malicious security for SCALES: Outsourced computation with ephemeral servers. Cryptology ePrint Archive, Paper 2024/383, 2024. (Cited on page 5.)
- [AHS20] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. A survey of ecdsa threshold signing. Cryptology ePrint Archive, Paper 2020/1390, 2020. (Cited on page 4.)
- [BDLO15] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security*, Cham, 2015. (Cited on page 12.)

- [BDST20] Lennart Braun, Daniel Demmler, Thomas Schneider, and Oleksandr Tkachenko. MOTION - A framework for mixed-protocol multi-party computation. Cryptology ePrint Archive, Report 2020/1137, 2020. <https://eprint.iacr.org/2020/1137>. (Cited on page 2, 5, 21.)
- [Bit18] BitcoinExchangeGuide. CipherTrace Releases Report Exposing Close to \$1 Billion Stolen in Crypto Hacks During 2018. <https://coinexchangeguide.com/ciphertrace-releases-report-exposing-close-to-1-billion-stolen-in-crypto-hacks-during-2018/>, 2018. (Cited on page 1.)
- [Blo18] Bloomberg. How to Steal \$500 Million in Cryptocurrency. <http://fortune.com/2018/01/31/coincheck-hack-how/>, 2018. (Cited on page 1.)
- [BMP22] Constantin Blokh, Nikolaos Makriyannis, and Udi Peled. Efficient asymmetric threshold ecdsa for mpc-based cold storage. Cryptology ePrint Archive, Paper 2022/1296, 2022. <https://eprint.iacr.org/2022/1296>. (Cited on page 4.)
- [BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). pages 503–513, 1990. (Cited on page 22.)
- [BST21] Charlotte Bonte, Nigel Smart, and Titouan Tanguy. Thresholdizing hasheddsa: Mpc to the rescue. *International Journal of Information Security*, 20, 12 2021. (Cited on page 5, 12.)
- [CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). pages 11–19, 1988. (Cited on page 2, 12.)
- [CCL<sup>+</sup>20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. pages 266–296, 2020. (Cited on page 2, 4, 5.)
- [CCL<sup>+</sup>21] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold ec-dsa revisited: Online/offline extensions, identifiable aborts, proactivity and adaptive security. Cryptology ePrint Archive, Paper 2021/291, 2021. <https://eprint.iacr.org/2021/291>. (Cited on page 2, 4, 5.)
- [CGG<sup>+</sup>20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS '20*, 2020. (Cited on page 4, 5, 30.)
- [CHL23] ChihYun Chuang, IHung Hsu, and TingFang Lee. A two-party hierarchical deterministic wallets in practice. In *Proceedings of the 20th International Conference on Security and Cryptography, SECRYPT 2023*. SCITEPRESS, 2023. (Cited on page 2, 4.)
- [CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. pages 89–105, 1993. (Cited on page 26.)
- [DEF<sup>+</sup>21] Poulami Das, Andreas Erwig, Sebastian Faust, Julian Loss, and Siavash Riahi. The exact security of bip32 wallets. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS '21*, 2021. (Cited on page 2, 3, 4, 5, 7, 9, 10, 12, 27.)
- [DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. pages 307–315, 1990. (Cited on page 2.)
- [DFL19] Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. pages 651–668, 2019. (Cited on page 4.)
- [DMZ<sup>+</sup>21] Yi Deng, Shunli Ma, Xinxuan Zhang, Hailong Wang, Xuyang Song, and Xiang Xie. Promise  $\sigma$ -protocol: How to construct efficient threshold ecdsa from encryptions based on class groups. Springer-Verlag, 2021. (Cited on page 2, 4, 5.)
- [Dod03] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. pages 1–17, 2003. (Cited on page 3, 5.)

- [EGPS22] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Turbopack: honest majority mpc with constant online communication. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 951–964, 2022. (Cited on page 2, 5.)
- [Ele13] Version bytes for BIP32 extended public and private keys. [https://electrum.readthedocs.io/en/latest/xpub\\_version\\_bytes.html](https://electrum.readthedocs.io/en/latest/xpub_version_bytes.html), 2013. (Cited on page 2.)
- [ER] Andreas Erwig and Siavash Riahi. Deterministic wallets for adaptor signatures. In *Computer Security – ESORICS 2022*. (Cited on page 4.)
- [FKM<sup>+</sup>16] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. pages 301–330, 2016. (Cited on page 7.)
- [FKP17] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the one-per-message unforgeability of (EC)DSA and its variants. pages 519–534, 2017. (Cited on page 17, 33.)
- [GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18*, 2018. (Cited on page 4, 5, 10, 28, 30, 31.)
- [GLOW21] David Galindo, Jia Liu, Mihair Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2021. (Cited on page 7, 8, 13, 20, 26, 27.)
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. pages 218–229, 1987. (Cited on page 2, 12.)
- [Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004. (Cited on page 12.)
- [GS15] Gus Gutoski and Douglas Stebila. Hierarchical deterministic bitcoin wallets that tolerate key leakage. pages 497–504, 2015. (Cited on page 4.)
- [GS20] Vipul Goyal and Yifan Song. Malicious security comes free in honest-majority MPC. Cryptology ePrint Archive, Paper 2020/134, 2020. (Cited on page 5.)
- [GS22] Jens Groth and Victor Shoup. Design and analysis of a distributed ecdsa signing service. Cryptology ePrint Archive, Paper 2022/506, 2022. <https://eprint.iacr.org/2022/506>. (Cited on page 4.)
- [HHPV21] Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. *Journal of Cryptology*, 34(3):19, 2021. (Cited on page 2, 5.)
- [Kel20] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 1575–1590, 2020. (Cited on page 5, 21.)
- [KMOS21] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. Refresh when you wake up: Proactive threshold wallets with offline devices. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 608–625, 2021. (Cited on page 4.)
- [Led14] Ledger Support, Ledger Nano OS. <https://support.ledger.com/hc/en-us/articles/115005297709-Export-your-accounts>, 2014. (Cited on page 2.)
- [LFA20] Adriano Di Luzio, Danilo Francati, and Giuseppe Ateniese. Arcula: A secure hierarchical deterministic wallet for multi-asset blockchains. pages 323–343, 2020. (Cited on page 4.)

- [LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. CCS '18, 2018. (Cited on page 4.)
- [MPC20] 'bristol fashion' mpc circuits. 2020. (Cited on page 5.)
- [MPs19] Antonio Marcedone, Rafael Pass, and abhi shelat. Minimizing trust in hardware wallets with two factor signatures. pages 407–425, 2019. (Cited on page 4.)
- [MZW<sup>+</sup>19] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. Churp: Dynamic-committee proactive secret sharing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2369–2386, 2019. (Cited on page 12.)
- [Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. pages 223–238, 1999. (Cited on page 30.)
- [Ske18] Rhys Skellern. Cryptocurrency Hacks: More Than \$2b USD lost between 2011-2018. [https://medium.com/ecomio/cryptocurrency-hacks-more-than-2b-usd-lost-between-2011-2018\\_-67054b342219](https://medium.com/ecomio/cryptocurrency-hacks-more-than-2b-usd-lost-between-2011-2018_-67054b342219), 2018. (Cited on page 1.)
- [SLL] David Schultz, Barbara Liskov, and Moses Liskov. Mpss: Mobile proactive secret sharing. *ACM Trans. Inf. Syst. Secur.*, 13(4). (Cited on page 12.)
- [Tre14] Trezor Wiki, Cryptocurrency standards, Hierarchical deterministic wallets. [https://wiki.trezor.io/Cryptocurrency\\_standards](https://wiki.trezor.io/Cryptocurrency_standards), 2014. (Cited on page 2.)
- [Wik18] Bitcoin Wiki. BIP32 proposal. [https://en.bitcoin.it/wiki/BIP\\_0032](https://en.bitcoin.it/wiki/BIP_0032), 2018. (Cited on page 2.)
- [Yeh23] Yehuda Lindell. Cryptography and MPC in Coinbase Wallet as a Service (WaaS). <https://coinbase.bynder.com/m/687ea39fd77aa80e/original/CB-MPC-Whitepaper.pdf>, 2023. (Cited on page 2.)
- [YLY<sup>+</sup>22] Xin Yin, Zhen Liu, Guomin Yang, Guoxing Chen, and Haojin Zhu. Secure hierarchical deterministic wallet supporting stealth address. Cryptology ePrint Archive, Paper 2022/627, 2022. (Cited on page 4.)

## A Additional Preliminaries

### A.1 Correctness and one-per message unforgeability under honestly rerandomizable keys of signature schemes with rerandomizable keys

For the empty string  $\epsilon$ , we have  $\text{RandPK}(\text{pk}, \epsilon) = \text{pk}$  and  $\text{RandSK}(\text{sk}, \epsilon) = \text{sk}$ .

We further require:

1. *(Perfect) rerandomizability of keys:* For all  $\kappa \in \mathbb{N}$ , all  $(\text{sk}, \text{pk}) \in \text{Gen}(1^\kappa)$  and  $\rho \xleftarrow{\$} \mathcal{R}$ , the distributions of  $(\text{sk}', \text{pk}')$  and  $(\text{sk}'', \text{pk}'')$  are identical, where:

$$\begin{aligned} (\text{sk}', \text{pk}') &\leftarrow (\text{RandSK}(\text{sk}, \rho), \text{RandPK}(\text{pk}, \rho)) \\ &\text{and} \\ (\text{sk}'', \text{pk}'') &\xleftarrow{\$} \text{Gen}(1^\kappa). \end{aligned}$$

2. *Correctness under rerandomized keys:* For all  $\kappa \in \mathbb{N}$ , all  $(\text{sk}, \text{pk}) \in \text{Gen}(1^\kappa)$ , all  $\rho \in \mathcal{R}$ , and all  $m \in \{0, 1\}^*$ , the rerandomized keys  $\text{sk}' \leftarrow \text{RandSK}(\text{sk}, \rho)$  and  $\text{pk}' \leftarrow \text{RandPK}(\text{pk}, \rho)$  satisfy:

$$\Pr[\text{Verify}(\text{pk}', \sigma, m) = 1 \mid \sigma \leftarrow \text{Sign}(\text{sk}', m)] = 1.$$

The security notion of *one-per message existential unforgeability under honestly rerandomizable keys* (**uf-cma-hrk1**) differs from the unforgeability notion of standard signature scheme in the following ways: (1) the signing oracle cannot only return signatures under  $\text{sk}$ , but it can also return signatures that were produced with keys that represent *honest* rerandomizations of  $\text{sk}$ ; (2) the randomness for the rerandomization is chosen uniformly at random from  $\mathcal{R}$  by the game; (3) the signing oracle returns at most one signature for each randomness/message pair  $(\rho, m)$ . The notion of **uf-cma-hrk1** for a rerandomizable signature scheme  $\text{RSig}$  is formally modeled in the form of a game  $\text{uf-cma-hrk1}_{\text{RSig}}$  which we recall in the following definition.

**Definition A.1** (One-per message unforgeability under honestly rerandomizable keys of signature schemes with rerandomizable keys). A signature scheme with honestly rerandomizable keys  $\text{RSig}$  is **uf-cma-hrk1**-secure if no PPT adversary  $\mathcal{A}$  wins game **uf-cma-hrk1** as described below with more than advantage. We define  $\mathcal{A}$ 's advantage in game  $\text{uf-cma-hrk1}_{\text{RSig}}$  as  $\text{Adv}_{\text{uf-cma-hrk1}_{\text{RSig}}}^{\mathcal{A}} := \Pr[\text{uf-cma-hrk1}_{\text{RSig}}^{\mathcal{A}} = 1]$ .

Game **uf-cma-hrk1**<sub>RSig</sub>:

- The challenger initializes two lists as  $\text{SigList} \leftarrow \{\epsilon\}$  and  $\text{RList} \leftarrow \{\epsilon\}$  and samples a pair of keys  $(\text{pk}, \text{sk}) \xleftarrow{\$} \text{RSig.Gen}(1^\kappa)$ . Then  $\mathcal{A}$  is run on input  $\text{pk}$ .
- $\mathcal{A}$  is given access to the following oracles:
  - **Rand**: Upon a query, this oracle samples a fresh random value from  $\mathcal{R}$  as  $\rho \xleftarrow{\$} \mathcal{R}$ , stores  $\rho$  in  $\text{RList}$ , and returns  $\rho$ .
  - **RSig**: On input a message  $m$  and a randomness  $\rho$ , if  $\rho$  was not obtained via a prior **Rand** query (i.e.,  $\rho \notin \text{RList}$ ), then this oracle returns  $\perp$ . Otherwise, it derives a pair of keys rerandomized with the randomness  $\rho$ , as  $\text{sk}' \leftarrow \text{RSig.RandSK}(\text{sk}, \rho)$  and  $\text{pk}' \leftarrow \text{RSig.RandPK}(\text{pk}, \rho)$ . If  $(\text{pk}', m) \in \text{SigList}$  then the oracle returns  $\perp$ . Otherwise, it derives a signature on message  $m$  under the secret key  $\text{sk}'$  as  $\sigma \leftarrow \text{RSig.Sign}(\text{sk}', m)$ . The oracle stores the tuple  $(\text{pk}', m)$  in  $\text{SigList}$  and returns  $\sigma$ .
- $\mathcal{A}$  wins if it returns a forgery  $\sigma^*$  together with a message  $m^*$  and a public key  $\text{pk}^* \leftarrow \text{RSig.RandPK}(\text{pk}, \rho^*)$ , s.t. the following holds: (1) the randomness  $\rho^*$  has been derived via a **Rand** query, i.e.,  $\rho^* \in \text{RList}$ , (2)  $(\text{pk}^*, m^*) \notin \text{SigList}$ , and (3)  $\sigma^*$  is a valid forgery, i.e.,  $\text{RSig.Verify}(\text{pk}^*, \sigma^*, m^*) = 1$ .

## A.2 TVRF Construction from Galindo et al. [GLOW21]

We briefly recall the TVRF construction from Galindo et al. that is based on the DDH assumption. The construction relies on a non-interactive zero-knowledge proof system (NIZK) for the relation  $\mathbf{R} := \{(g, h, X, Y), x \mid X = g^x, Y = h^x\}$  where  $g$  and  $h$  are two generators of a cyclic group  $\mathbb{G}$  of prime order  $q$  and  $x \in \mathbb{Z}_q$ . At a high level, the NIZK proves that two group elements  $X$  and  $Y$  have the same discrete logarithm w.r.t. generators  $g$  and  $h$ . This proof system was first introduced by Chaum and Pedersen [CP93] and we denote it by **DLEq**. We recall the proof system in Figure 6 and the TVRF construction, which we denote by **TVRF**, in Figure 7. The corruption threshold for the  $(t, n)$ -TVRF scheme is set to  $t \leq \frac{n-1}{2}$ .

<b>DLEq.Prove</b> $(g^x, h^x, x)$ 00 Sample $r \xleftarrow{\$} \mathbb{Z}_q$ . 01 Compute $c \leftarrow \mathbf{H}(g^x, h^x, g^r, h^r)$ 02 Compute $s = r + c \cdot x$ . 03 Return $\pi := (c, s)$ .	<b>DLEq.Verify</b> $(g^x, h^x, \pi)$ 00 Parse $\pi := (c, s)$ . 01 $R \leftarrow g^s / (g^x)^c$ . 02 $R' \leftarrow h^s / (h^x)^c$ . 03 If $c \neq \mathbf{H}_1(g^x, h^x, R, R')$ : Return 0. 04 Return 1
--	--

Figure 6: NIZK proof of equality of discrete logarithms with  $\mathbf{H} : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ .

<b>TVRF.Gen</b> ( $1^\kappa, t, n$ ) 00 Sample $a_i \xleftarrow{\$} \mathbb{Z}_q$ for $i \in [t] \cup \{0\}$ 01 Let $F(x) := a_t x^t + \dots + a_1 x + a_0$ 02 $\text{sk} := x \leftarrow a_0 \pmod q, X \leftarrow g^x$ 03 $\text{sk}_i := x_i \leftarrow F(i) \pmod q, X_i \leftarrow g^{x_i}$ 04 $\text{pk} := (X, \{X_1, \dots, X_n\})$ 05 Return $(\text{pk}, \{\text{sk}_1, \dots, \text{sk}_n\})$	<b>TVRF.PEval</b> ( $m, \text{sk}_i, \text{pk}$ ) 00 Parse $\text{pk} := (X, \{X_1, \dots, X_n\})$ 01 $\phi_i \leftarrow \text{H}_1(m)^{\text{sk}_i}$ . 02 $\pi_i \leftarrow \text{DLEq.Prove}(\phi_i, X_i, \text{sk}_i)$ . 03 Return $(\phi_i, \pi_i)$ .
<b>TVRF.Combine</b> ( $\text{pk}, \mathcal{S}, \{(\phi_i, \pi_i)\}_{i \in \mathcal{S}}$ ) 00 If $ \mathcal{S}  \leq t$ : Return $\perp$ . 01 Let $\mathcal{S}' := \emptyset$ . 02 Parse $\text{pk} := (X, \{X_1, \dots, X_n\})$ . 03 For $i \in \mathcal{S}$ , if $\text{DLEq.Verify}(\phi_i, X_i, \pi_i) = 1$ : 04     Then $\mathcal{S}' \leftarrow \mathcal{S}' \cup i$ . 05 If $ \mathcal{S}'  \leq t$ : Return $\perp$ . 06 $\phi \leftarrow \prod_{i \in \mathcal{S}'} \phi_i^{\lambda_i}$ and $\pi := \{(\phi_i, \pi_i)\}_{i \in \mathcal{S}'}$ . 07 Return $(\phi, \pi)$ .	<b>TVRF.Verify</b> ( $\text{pk}, m, \phi, \pi$ ) 00 Parse $\text{pk} := (X, \{X_1, \dots, X_n\})$ . 01 Parse $\pi := \{(\phi_i, \pi_i)\}_{i \in \mathcal{S}'}$ . 02 Let $\mathcal{S}' := \emptyset$ . 03 For $i \in \mathcal{S}'$ : 04     if $\text{DLEq.Verify}(\phi_i, X_i, \pi_i) \neq 1$ 05         return $\perp$ . 06 If $\phi = \prod_{i \in \mathcal{S}'} \phi_i^{\lambda_i}$ : Return 1. 07 Else return 0.

Figure 7: Threshold verifiable random function from [GLOW21] for a cyclic group  $\mathbb{G} = \langle g \rangle$  of prime order  $q$  and for a cryptographic hash function  $\text{H}_1 : \{0, 1\}^* \rightarrow \mathbb{G}$ .

### A.3 ECDSA with Rerandomizable Keys

We briefly recall the standard ECDSA signature scheme in Figure 8 and then describe how it can be extended to achieve the ECDSA-based signature scheme with additively rerandomizable keys as shown in [DEF+21].

The ECDSA signature scheme is defined for a cyclic group  $\mathbb{G} = \langle g \rangle$  of prime order  $q$  where the discrete logarithm problem in  $\mathbb{G}$  is hard. We briefly recall the scheme here, which we denote by  $\text{ECDSA}[\text{H}]$ , where  $\text{H} : \{0, 1\}^* \rightarrow \mathbb{Z}_q$  is a cryptographic hash function.

<b>Gen</b> ( $1^\kappa$ ) 00 $x \xleftarrow{\$} \mathbb{Z}_q$ 01 $X \leftarrow g^x$ 02 $(\text{sk}, \text{pk}) := (x, X)$ 03 Return $(\text{sk}, \text{pk})$	<b>Sign</b> ( $\text{sk}, m$ ) 00 Parse $\text{sk} := x$ 01 $k \xleftarrow{\$} \mathbb{Z}_q, R \leftarrow g^k$ 02 If $R = 1$ : Return $\perp$ 03 $r \leftarrow f(R)$ 04 If $r = 0$ : Return $\perp$ 05 $h \leftarrow \text{H}(m)$ 06 $s = k^{-1}(h + r \cdot x)$ 07 If $s = 0$ : Return $\perp$ 08 Return $\sigma := (r, s)$	<b>Verify</b> ( $\text{pk}, m, \sigma$ ) 00 Parse $\text{pk} := X$ and $\sigma := (r, s)$ 01 If $s = 0 \vee t = 0$ : Return $\perp$ 02 $h \leftarrow \text{H}_0(m)$ 03 $u_1 \leftarrow h \cdot s^{-1}$ 04 $u_2 \leftarrow r \cdot s^{-1}$ 05 $R \leftarrow g^{u_1} + X^{u_2}$ 06 If $f(R) = r$ : Return 1 07 Return 0
--	---	---

Figure 8: ECDSA signature scheme  $\text{ECDSA}[\text{H}]$  instantiated with a cryptographic hash function  $\text{H} : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ .

In Figure 9, we recall the ECDSA-based signature scheme with rerandomizable keys  $\text{rECDSA}[\text{H}]$  as introduced in [DEF+21].

<b>Sign</b> ( $sk, m$ ) 00 $m' \leftarrow (pk, m)$ 01 $\sigma \leftarrow \text{ECDSA}[H].\text{Sign}(sk, m')$ 02 Return $\sigma$	<b>RandSK</b> ( $sk, \rho$ ) 00 $sk' \leftarrow (sk + \rho) \bmod q$ 01 Return $sk'$
<b>Verify</b> ( $pk, \sigma, m$ ) 03 $m' \leftarrow (pk, m)$ 04 Return <b>ECDSA</b> [H].Verify ( $pk, m', \sigma$ )	<b>RandPK</b> ( $pk, \rho$ ) 02 $pk' \leftarrow (pk + g^\rho)$ 03 Return $pk'$

Figure 9: Public key prefixed version of the ECDSA signature scheme with perfectly rerandomizable keys  $r\text{ECDSA}[H]$  based on the ECDSA signature scheme  $\text{ECDSA}[H]$ . Above  $H: \{0, 1\}^* \rightarrow \mathbb{Z}_q$  denotes a cryptographic hash function.

## B The GG scheme by Gennaro and Goldfeder [GG18]

### B.1 Underlying Assumptions and Building Blocks

**Decisional Diffie-Hellman Problem (DDH)** Let  $\mathbb{G}$  be a cyclic group of prime order  $q$  and let  $g$  be a generator of  $\mathbb{G}$ . Let  $a, b, c$  be elements chosen uniformly at random from  $\mathbb{Z}_q$ . Then the distributions  $(g, g^a, g^b, g^{ab})$  and  $(g, g^a, g^b, g^c)$  are computationally indistinguishable.

**Non-interactive zero knowledge proof (NIZK)** A NIZK proof of knowledge with respect to a polynomial-time recognizable binary relation  $R$  is given by the following tuple of PPT algorithms  $\text{ZK} := (\text{Setup}, \text{Prove}, \text{Verify})$ , where (i)  $\text{Setup}(1^\kappa)$  outputs a common reference string  $\text{crs}$ ; (ii)  $\text{Prove}(\text{crs}, (Y, y))$  outputs a proof  $\pi$  for  $(Y, y) \in R$ ; (iii)  $\text{Verify}(\text{crs}, Y, \pi)$  outputs a bit  $b \in \{0, 1\}$ . Further, the NIZK proof of knowledge w.r.t.  $R$  should satisfy the following properties:

1. *Completeness*: For all  $(Y, y) \in R$ , all  $\kappa \in \mathbb{N}$  and  $\text{crs} \leftarrow \text{Setup}(1^\kappa)$ , it holds that  $\text{Verify}(\text{crs}, Y, \text{Prove}(\text{crs}, (Y, y))) = 1$  except with negligible probability;
2. *Soundness*: For any  $(Y, y) \notin R$ , all  $\kappa \in \mathbb{N}$  and  $\text{crs} \leftarrow \text{Setup}(1^\kappa)$ , it holds that  $\text{Verify}(\text{crs}, Y, \text{Prove}(\text{crs}, (Y, y))) = 0$  except with negligible probability;
3. *Zero knowledge*: For any PPT adversary  $\mathcal{A}$ , there exist a PPT algorithm  $\pi_{\mathcal{S}} \leftarrow \mathcal{S}(\text{crs}, Y)$  such that for all  $\kappa \in \mathbb{N}$ , all  $\text{crs} \leftarrow \text{Setup}(1^\kappa)$  and all  $(Y, y) \in R$ , the distributions  $\{(\pi, Y) : \pi \leftarrow \text{Prove}(\text{crs}, Y, y)\}$  and  $\{(\pi_{\mathcal{S}}, Y) : \pi_{\mathcal{S}} \leftarrow \mathcal{S}(\text{crs}, Y)\}$  are indistinguishable to  $\mathcal{A}$  except with negligible probability.

**Non-Malleable and Equivocable Commitments** A non-malleable and equivocable commitment scheme with message space  $\{0, 1\}^*$ , commitment space  $\mathcal{C}$  and opening space  $\mathcal{O}$  consists of a tuple of three PPT algorithms  $\text{CT} := (\text{Gen}, \text{Com}, \text{Open}, \text{Equivocate})$  where  $\text{Gen}$  gets as input the security parameter  $\kappa \in \mathbb{N}$  and outputs public parameters  $\text{par}$  and a trapdoor  $\tau$ ;  $\text{Com}$  takes as input  $\text{par}$  and a message  $m \in \{0, 1\}^*$  and outputs a tuple  $(c, d)$ ;  $\text{Open}$  takes as input  $\text{par}$  and a tuple  $(c, d) \in (\mathcal{C} \times \mathcal{O})$  and either outputs a message  $m$  or  $\perp$ ;  $\text{Equivocate}$  takes as input a trapdoor  $\tau$ , a commitment  $c \in \mathcal{C}$  and a message  $m \in \{0, 1\}^*$  and outputs an opening  $d$ . A non-malleable and equivocable commitment scheme must satisfy the following properties:

1. *Computationally Hiding*: For all  $\kappa \in \mathbb{N}$ , all  $(\text{par}, \tau) \leftarrow \text{Gen}(1^\kappa)$ , any two messages  $m, m' \in \{0, 1\}^*$  and  $(c, d) \leftarrow \text{Com}(\text{par}, m)$  and  $(c', d') \leftarrow \text{Com}(\text{par}, m')$ , there exists no PPT adversary  $\mathcal{A}$  which can distinguish the tuples  $(m, m', c)$  and  $(m, m', c')$  except with negligible probability.
2. *Computationally Binding*: For all  $\kappa \in \mathbb{N}$  and all  $(\text{par}, \tau) \leftarrow \text{Gen}(1^\kappa)$ , there exists no PPT adversary  $\mathcal{A}$  which can output  $(c, d, d')$  such that  $\text{Open}(\text{par}, c, d) \neq \text{Open}(\text{par}, c, d')$  and  $\text{Open}(\text{par}, c, d) \neq \perp$  and  $\text{Open}(\text{par}, c, d') \neq \perp$  except with negligible probability.

3. *Equivocable*: For all  $\kappa \in \mathbb{N}$ , all  $(\text{par}, \tau) \leftarrow \text{Gen}(1^\kappa)$  and any message  $m \in \{0, 1\}^*$  the distributions  $\{(c, d) : (c, d) \leftarrow \text{Com}(\text{par}, m)\}$  and  $\{(c', d') : c' \xleftarrow{\$} \mathcal{C}, d' \leftarrow \text{Equivocate}(\tau, c', m)\}$  are computationally indistinguishable.

Finally, a commitment scheme is *non-malleable* if for all  $\kappa \in \mathbb{N}$ , all  $(\text{par}, \tau) \leftarrow \text{Gen}(1^\kappa)$ , any message  $m \in \{0, 1\}^*$  and  $(c, d) \leftarrow \text{Com}(\text{par}, m)$ , there exists no PPT adversary  $\mathcal{A}$  which on input  $c$  can output a commitment  $c'$  such that after receiving the opening  $d$  the adversary  $\mathcal{A}$  can output an opening  $d'$  such that for  $m' \leftarrow \text{Open}(\text{par}, c', d')$  the messages  $m$  and  $m'$  are related.

**Public Key Encryption** A public key encryption scheme consists of three algorithms  $\text{PKE} := (\text{Gen}, \text{Enc}, \text{Dec})$ , where (i)  $\text{Gen}(1^\kappa)$  outputs a public key  $\text{pk}$  and a secret key  $\text{sk}$ ; (ii)  $\text{Enc}(\text{pk}, m)$  outputs a ciphertext  $ct$ ; and (iii)  $\text{Dec}(\text{sk}, ct)$  outputs either  $\perp$  or a message  $m$ .

A public key encryption scheme  $\text{pk} := (\text{Gen}, \text{Enc}, \text{Dec})$  is linearly homomorphic if (1) there exists an efficiently computable operation  $\oplus$  s.t. for two ciphertexts  $ct_1 \leftarrow \text{Enc}(\text{pk}, m_1)$  and  $ct_2 \leftarrow \text{Enc}(\text{pk}, m_2)$  it holds that  $ct_1 \oplus ct_2 = \text{Enc}(\text{pk}, m_1 + m_2)$ ; and (2) there exists an efficiently computable operation  $\odot$  s.t. for a ciphertext  $ct_1 \leftarrow \text{Enc}(\text{pk}, m_1)$  and a constant  $k$  it holds that  $ct_1 \odot k = \text{Enc}(\text{pk}, m_1 \cdot k)$ .

A public key encryption scheme is semantically secure if for every PPT adversaries  $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$  there exists a negligible function  $\nu$  in the security parameter  $\kappa \in \mathbb{N}$  s.t.:

$$\Pr \left[ b' = b \mid \begin{array}{l} (\text{pk}, \text{sk}) \leftarrow \text{Gen}(1^\kappa), \\ (m_1, m_2, s) \leftarrow \mathcal{A}_1(\text{pk}), \\ b \xleftarrow{\$} \{0, 1\}, ct \leftarrow \text{Enc}(\text{pk}, m_b), \\ b' \leftarrow \mathcal{A}_2(s, ct) \end{array} \right] \leq 1/2 + \nu(\kappa).$$

## B.2 Construction

The  $\text{GG}[\text{H}_0]$  scheme relies on a multiplicative to additive share conversion protocol, which allows two parties  $P_i$  and  $P_j$  with shares  $a_i \in \mathbb{Z}_q$  and  $b_j \in \mathbb{Z}_q$  respectively s.t.  $x = a_i \cdot b_j \pmod q$  to transform  $a_i$  and  $b_j$  into additive shares of  $x$ , i.e., into shares  $\alpha_i$  and  $\beta_j$  s.t.  $x = \alpha_i + \beta_j$ . We briefly recall this protocol here. We denote by  $\text{PKE}$  a linearly homomorphic encryption scheme (with operations  $\odot$  for multiplication with a constant and  $\oplus$  for homomorphic addition) over an integer  $N$  and we denote by  $(\text{pk}_{i, \text{PKE}}, \text{sk}_{i, \text{PKE}})$  the public/secret key pair of scheme  $\text{PKE}$  of  $P_i$ .

$P_i(\text{pk}_{j, \text{PKE}}, \text{sk}_{i, \text{PKE}}, a_i)$	$P_j(\text{pk}_{i, \text{PKE}}, b_j)$
$c_i \leftarrow \text{PKE.Enc}(\text{pk}_{i, \text{PKE}}, a_i)$ Compute a ZK proof $\pi_i$ that $a < q^3$	$\xrightarrow{c_i, \pi_i}$ If $\pi_i$ is not valid, abort. $y \xleftarrow{\$} \mathbb{Z}_{q^5}, c_j \leftarrow (c_i \odot b_j) \oplus \text{PKE.Enc}(\text{pk}_{i, \text{PKE}}, y)$ Compute a ZK proof $\pi_j$ that $P_j$ knows $b_j < q^3, y < q^7$ s.t. $c_j \leftarrow (c_i \odot b_i) \oplus \text{PKE.Enc}(\text{pk}_{i, \text{PKE}}, y)$ $\xleftarrow{c_j, \pi_j}$
If $\pi_j$ is not valid, abort. $\alpha \leftarrow \text{PKE.Dec}(\text{sk}_{i, \text{PKE}}, c_j) \pmod q$	$\beta \leftarrow -y \pmod q$

Figure 10: Multiplicative to additive share conversion protocol MtA.

Gennaro and Goldfeder also consider a slight adjustment of the above protocol which they call  $\text{MtAwc}$ , which differs only from the above protocol in the following way: If  $B_j = g^{b_j}$  is a public value (where  $g$  is the generator of a cyclic group of prime order  $q$ ), then party  $P_j$  additionally proves in zero-knowledge that  $b_j$  is the discrete log of  $B_j$ . We now recall the key generation and signing procedures of the  $\text{GG}[\text{H}_0]$  scheme. For simplicity, we slightly deviate from the original  $\text{GG}[\text{H}_0]$  scheme in two ways, which however has no impact on the scheme's security: Gennaro and Goldfeder consider a distributed key generation, whereas we assume that the key generation is initially executed by a trusted party. In addition, we do

```

Algorithm Gen( $1^\kappa, t, n$ )
00 For  $k \in [t] \cup \{0\}$ , sample  $a_k \xleftarrow{\$} \mathbb{Z}_q$ .
01 Let  $F(x) := a_t x^t + \dots + a_1 x + a_0$ .
02  $\text{sk} := x \leftarrow a_0 \pmod q$ .
03 Set  $X \leftarrow g^x$  and  $\text{sk}_i := x_i \leftarrow F(i) \pmod q$ .
04 Set  $X_i \leftarrow g^{x_i}$ .
05 Set  $\text{pk} := (X, \{X_i\}_{i \in [n]})$ .
06 Return  $(\text{pk}, \{\text{sk}_i\}_{i \in [n]})$ .

```

Figure 11: Key generation algorithm. Note that Gennaro and Goldfeder consider a distributed key generation, whereas we assume that the key generation is initially executed by a trusted party.

not generate the keys for the linearly homomorphic encryption scheme during the initial key generation but we let parties generate fresh keys in the beginning of an execution of the signing procedure.

In Figure 12 we recall the signing procedure of the  $\text{GG}[\text{H}_0]$  scheme. The procedure makes use of a non-malleable and equivocable commitment scheme  $\text{CT} := (\text{Com}, \text{Open})$  as well as a hash function  $\text{H}_0 : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ , a linearly homomorphic encryption scheme  $\text{PKE} := (\text{Gen}, \text{Enc}, \text{Dec})$  and a non-interactive zero-knowledge proof system  $\text{ZK}$ . We slightly adjust the signing procedure as follows: Instead of letting parties generate their key pair for PKE during the initial key generation, we let parties generate a fresh key pair  $(\text{pk}_{i, \text{PKE}}, \text{sk}_{i, \text{PKE}})$  for the PKE scheme before Phase 1 of the signing procedure. Each party then broadcasts  $\text{pk}_{i, \text{PKE}}$  together with a zero-knowledge proof that the key was generated honestly.<sup>11</sup> The parties then engage in the signing procedure as specified in Figure 12.

### B.3 Security Proof

Gennaro and Goldfeder [GG18] prove that their scheme is unforgeable as per Definition 3.2 under the assumptions from Appendix B.1. In Figure 13 we recall the simulation of the signing procedure that was used as part of the unforgeability proof in [GG18] (with some minor modifications). The forger  $\mathcal{F}$  provides a computationally indistinguishable view of the signing procedure of the GG scheme to a PPT adversary on input the secret key shares of corrupted parties and with access to a signing oracle.

### B.4 Proof Sketch of Theorem 5.2

#### B.4.1 Case $b = 0$

In this case,  $\mathcal{B}$  executes  $\mathcal{B}_0$  which plays in game  $\text{th-ufcma-hrk1}_{r\text{GG}}$ . That is, upon  $\mathcal{A}$  sending the list  $\mathcal{C}$  of parties to corrupt in game  $\text{unf-prand}_{\text{TVRF-rGG}}$ ,  $\mathcal{B}_0$  corrupts the same parties in  $\text{th-ufcma-hrk1}_{r\text{GG}}$  and forwards the resulting secret key shares and the public key to  $\mathcal{A}$ .

The simulation of the oracles  $\text{Rand}$ ,  $\text{RSign}$  and the random oracle  $\text{H}_0$  happens in a straightforward way, i.e.,  $\mathcal{B}_0$  simply forwards queries from  $\mathcal{A}$  in game  $\text{unf-prand}_{\text{TVRF-rGG}}$  to the corresponding oracle in game  $\text{th-ufcma-hrk1}_{r\text{GG}}$ .

The simulation of the random oracle  $\text{H}_1$  and the  $\text{REval}$  oracle is a bit more challenging as  $\mathcal{B}_0$  does not have access to any such oracle in game  $\text{th-ufcma-hrk1}_{r\text{GG}}$ . Upon a query from  $\mathcal{A}$  to the random oracle  $\text{H}_1$  on input a message  $m$ ,  $\mathcal{B}_0$  first checks if  $\text{H}_1(m)$  has been set already. If so, it simply returns  $\text{H}_1(m)$ . Otherwise it samples a uniformly random value  $r \xleftarrow{\$} \mathbb{Z}_q$  and sets  $\text{H}_1(m) := g^r$  and returns  $g^r$ . The simulation of the  $\text{REval}$  oracle then works as follows: On input a message  $m$ , an index  $i \in [n]$  and a randomness  $\rho \in \text{RList}$ ,  $\mathcal{B}_0$  first executes  $\text{pk}' \leftarrow \text{TVRF-rGG.RandPK}(\text{pk}, \rho)$  and parses  $\text{pk}' := (X', \{X'_1, \dots, X'_n\})$ .  $\mathcal{B}_0$  then retrieves  $r \leftarrow \text{H}_1(m)$ , sets  $\phi_i := (X'_i)^r = \text{H}_1(m)^{\text{sk}'_i}$ , simulates the corresponding zero-knowledge proof  $\pi_i$  and returns  $(\phi_i, \pi_i)$ .

<sup>11</sup>We note that if PKE is instantiated with the Paillier encryption scheme [Pai99], then this zero-knowledge proof can be instantiated with the Paillier-Blum Modulus zero-knowledge proof system [CGG+20].

$P_i(w_i, m)$	<b>Phase 1</b>	$P_j(w_j, m)\{j \neq i\}$
$k_i \xleftarrow{\$} \mathbb{Z}_q, \gamma_i \xleftarrow{\$} \mathbb{Z}_q$ $(C_i, D_i) \leftarrow \text{CT.Com}(g^{\gamma_i})$	$\xrightarrow{C_i}$	
Define the following: $k = \sum_{i \in S} k_i, \gamma = \sum_{i \in S} \gamma_i$ $k\gamma = \sum_{i,j \in S} k_i \gamma_j \pmod q$ $kx = \sum_{i,j \in S} k_i w_j \pmod q$		
<b>Phase 2</b>		
$\delta_i = k_i \gamma_i + \sum_{j \neq i} (\alpha_{i,j} + \beta_{j,i})$ $\sigma_i = k_i w_i + \sum_{j \neq i} (\mu_{i,j} + \nu_{j,i})$		
<b>Phase 3</b>		
$\xrightarrow{\delta_i}$		$\delta = \sum_{i \in S} \delta_i = k\gamma$
<b>Phase 4</b>		
$\pi_{\gamma_i} = \text{ZK}_{\Gamma_i}\{(\gamma_i): \Gamma_i = g^{\gamma_i}\}$	$\xrightarrow{D_i, \pi_{\gamma_i}}$	$\Gamma_i = \text{CT.Open}(C_i, D_i)$ Abort if $\pi_{\gamma_i}$ does not verify $R = \left(\prod_{i \in S} \Gamma_i\right)^{\delta^{-1}} = g^{k^{-1}}$ , where $R = (r_x, r_y)$ . Set $r = r_x \pmod q$
<b>Phase 5</b>		
$m' = H_0(m), s_i = m' k_i + r \sigma_i$ $l_i \xleftarrow{\$} \mathbb{Z}_q, \rho_i \xleftarrow{\$} \mathbb{Z}_q$ $V_i = R^{s_i} \cdot g^{l_i}, A_i = g^{\rho_i}$		
$(\hat{C}_i, \hat{D}_i) = \text{CT.Com}(V_i, A_i)$		$\xrightarrow{\hat{C}_i}$
$\hat{\pi}_i = \text{ZK}_{(V_i, A_i)}\{(s_i, l_i, \rho_i): (V_i = R^{s_i} \cdot g^{l_i}) \wedge (A_i = g^{\rho_i})\}$		$\xrightarrow{\hat{D}_i, \hat{\pi}_i}$
Abort if a proof fails $V = g^{-m'} \cdot Q^{-r} \cdot \prod_{i \in S} V_i = g^l$ $A = \prod_{i \in S} A_i$		
$U_i = V^{\rho_i}, T_i = A^{l_i}$		
$(\tilde{C}_i, \tilde{D}_i) = \text{CT.Com}(U_i, T_i)$		$\xrightarrow{\tilde{C}_i}$
$\xrightarrow{\tilde{D}_i}$		Abort if $\sum_{i \in S} T_i \neq \sum_{i \in S} U_i$
$\xrightarrow{s_i}$		$s = \sum_{i \in S} s_i$

Figure 12: Interactive  $(t, n)$ -threshold ECDSA scheme by Gennaro and Goldfeder [GG18], where  $|S| \subseteq [n], |S| = t + 1$ . For all parties  $\{P_i\}_{i \in [n]}$ ,  $x_i$  denotes secret share of the secret  $x$ . For all parties  $P_{i \in S}$ ,  $w_i$  represents the secret share of  $x$  due to  $(t, t + 1)$ -secret sharing of  $x$ , such that  $x = \sum_{i \in S} w_i$ .

**Simulation of the Signing Procedure:** Before Phase 1 of the signing procedure,  $\mathcal{F}$  samples uniformly at random a public key  $\text{pk}_{1,\text{PKE}}$  s.t.  $(\text{pk}_{1,\text{PKE}}, \cdot) \in \text{PKE.Gen}(1^\kappa)$  and simulates the zero-knowledge proof  $\pi_{1,\text{PKE}}$ .

Phase 1:  $\mathcal{F}$  executes Phase 1 honestly for party  $P_1$ , i.e., it samples  $k_1, \gamma_1 \xleftarrow{\$} \mathbb{Z}_q$  and commits to  $g^{\gamma_1}$ . It then broadcasts the commitment  $C_1$ .

Phase 2:  $\mathcal{F}$  executes the first MtA protocol correctly for  $P_1$  using the values  $k_1$  and  $\gamma_1$  and extracts the following values from the zero-knowledge proofs that are exchanged during the MtA protocol:  $k_i, \gamma_i, y_i$  for  $i > 1$ . It then computes  $\alpha_{1,j} = k_1 \gamma_i + y_i \pmod q$  and  $\tilde{k} = \sum_{i \in S} k_i \pmod q$ .

For the execution of the MtA<sub>wc</sub> protocol,  $\mathcal{F}$  does not know  $w_1$  when  $P_1$  is the reacting party. Therefore, it simply chooses a random  $\gamma_{j,1}$  and simulates the corresponding zero-knowledge proofs. When  $P_1$  is the initiating party,  $\mathcal{F}$  can execute the protocol honestly with input  $k_1$  and extract the share  $\nu_{1,j}$  from the zero-knowledge proofs.

Phase 3:  $\mathcal{F}$  executes this phase correctly for  $P_1$ .

Phase 4: All players decommit to  $\Gamma_i$ .  $\mathcal{F}$  extracts  $\gamma_j$  for all  $j_1$  from the zero-knowledge proofs  $\pi_{\gamma_j}$  and computes  $k = \delta \cdot \left( \sum_{i>0} \gamma_i \right)^{-1} \pmod q$ .

**If  $\tilde{k} = k$ , then  $\mathcal{F}$  proceeds as follows:**

- (a)  $\mathcal{F}$  queries its own signing oracle on message  $m$  to receive a signature  $(r, s)$  and computes  $R = g^{\text{H}(m)s^{-1}} \cdot X^{rs^{-1}}$ .
- (b)  $\mathcal{F}$  rewinds the adversary to the beginning of Phase 4 and equivocates the decommitment of  $P_1$  to  $\hat{\Gamma}_1 = R^\delta \prod_{i>1} \Gamma_i^{-1}$ .
- (c)  $\mathcal{F}$  computes  $s_1 = s - \sum_{i>1} s_i$ .

Phase 5:  $\mathcal{F}$  executes this phase correctly for  $P_1$  using  $s_1$ .

**Else if  $\tilde{k} \neq k$ , then  $\mathcal{F}$  proceeds as follows:**

Phase 4:  $\mathcal{F}$  runs this phase correctly for  $P_1$ .

Phase 5:  $\mathcal{F}$  chooses  $\tilde{s}_1 \xleftarrow{\$} \mathbb{Z}_q$  and runs this phase using this value.

Figure 13: Simulation of the signing procedure of the GG scheme. The forger  $\mathcal{F}$  receives as input the secret key shares of all corrupted parties and obtains access to a signing oracle.

Eventually, the adversary outputs a forgery which  $\mathcal{B}_0$  also forwards to the **th-ufcma-hrk1**<sub>rGG</sub> game. It is easy to see that  $\mathcal{B}_0$  wins the **th-ufcma-hrk1**<sub>rGG</sub> game if  $\mathcal{A}$  is able to win the **unf-prand**<sub>TVRF-rGG</sub> game by satisfying the winning condition in **Case 1**.

#### B.4.2 Case $b = 1$

In this case,  $\mathcal{B}$  executes  $\mathcal{B}_1$  which plays in game **th-prand**<sub>TVRF</sub>. That is, upon  $\mathcal{A}$  sending the list  $\mathcal{C}$  of parties to corrupt in game **unf-prand**<sub>TVRF-rGG</sub>,  $\mathcal{B}_1$  corrupts the same parties in **th-prand**<sub>TVRF</sub> and forwards the resulting secret key shares and the public key to  $\mathcal{A}$ . The simulation of oracles **Rand**, **RSign**,  $\mathsf{H}_0$ ,  $\mathsf{H}_1$  and **REval** then works as follows:

- **Oracle Rand:** On a query to **Rand** from  $\mathcal{A}$ ,  $\mathcal{B}_1$  samples uniformly at random  $\rho \leftarrow^{\$} \mathbb{Z}_q$ , stores  $\rho$  in **RList** and returns  $\rho$ .
- **Oracle  $\mathsf{H}_0$ :** Upon  $\mathcal{A}$  querying  $\mathsf{H}_0$  on input a message  $m$ ,  $\mathcal{B}_1$  first checks whether  $m$  is public key prefixed, i.e., whether  $m$  can be parsed as  $m := (\text{pk}', m')$  where  $(\text{pk}', \cdot) \in \text{TVRF-rGG.Gen}(1^\kappa, t, n)$ . If so,  $\mathcal{B}_1$  executes  $\mathcal{S}_{\text{ECDSA}}$  as described in Figure 4 on input  $(X', m)$  with  $\text{pk}' := (X', \{X'_1, \dots, X'_n\})$ . If  $m$  is not public key prefixed,  $\mathcal{B}_1$  simply samples a uniformly random value  $r \leftarrow^{\$} \mathbb{Z}_q$ , sets  $\mathsf{H}_0(m) := r$  and returns  $\mathsf{H}_0(m)$ . Note that in order for  $\mathcal{B}_1$  to abort in this simulation,  $\mathcal{A}$  would have to guess a randomness  $\rho \in \mathbb{Z}_q$  before it has been output by the **Rand** oracle. This happens only with negligible probability. Further, note that  $\mathcal{S}_{\text{ECDSA}}$  programs the random oracle  $\mathsf{H}_0$  in such a way that (1)  $\mathsf{H}_0(m)$  is set to uniform random value in  $\mathbb{Z}_q$ , and (2) the values  $(r, s)$  look like a valid ECDSA signature for  $m$  and  $X'$  to  $\mathcal{A}$  except with negligible probability (this has been shown in [FKP17]).
- **Oracle RSign:** Upon  $\mathcal{A}$  querying this oracle on input a message  $m$  and randomness  $\rho \in \text{RList}$ ,  $\mathcal{B}_1$  simulates the signing procedure in the same way as described in Theorem 4.4. Note that this simulation relies on the availability of a signing oracle, which returns full valid ECDSA signatures on arbitrary messages and rerandomized public keys. Since  $\mathcal{B}_1$  does not have access to such an oracle, it uses the simulated signatures  $(r, s)$  that are generated during the programming of  $\mathsf{H}_0$ . Note that the simulator code from Theorem 4.4 does not program  $\mathsf{H}_0$  such that there is no conflict between the execution of the simulator code from Theorem 4.4 and  $\mathcal{S}_{\text{ECDSA}}$ .
- **Oracle  $\mathsf{H}_1$ :** Upon  $\mathcal{A}$  querying  $\mathsf{H}_1$  on some message  $m$ ,  $\mathcal{B}_1$  simply queries its own random oracle on  $m$  and relays the output.
- **Oracle REval:** Upon a query from  $\mathcal{A}$  on input  $(m, i, \rho)$ ,  $\mathcal{B}_1$  queries its own oracle on input  $m$  and receives an evaluation share  $(\phi_i, \pi_i)$  where  $\phi_i = \mathsf{H}_1(m)^{\text{sk}_i}$ .  $\mathcal{B}_1$  then computes  $\phi'_i = \phi_i \cdot \mathsf{H}_1(m)^{\rho_i} = \mathsf{H}_1(m)^{\text{sk}_i + \rho_i}$  (where  $\rho_i$  is the randomness share of  $\rho$  for party  $P_i$  according to the **TVRF-rGG.RandSK** algorithm), simulates a NIZK proof  $\pi'_i$  of the DLEq proof system (cf. Appendix A.2) and sends  $(\phi'_i, \pi'_i)$  to  $\mathcal{A}$ .

**Reduction to th-prand**<sub>TVRF</sub>: During the challenge phase of **th-prand**<sub>TVRF-rGG</sub> (in **Case 2**),  $\mathcal{A}$  outputs a message  $m^*$ , randomness  $\rho^*$ , a set of indices  $\mathcal{S}$  and evaluation shares  $\{(\phi_i^*, \pi_i^*)\}_{i \in \mathcal{S} \cap \mathcal{C}}$ . Upon receiving these values, the adversary  $\mathcal{B}_1$  computes  $\phi_i = \phi_i^* \cdot \mathsf{H}_1(m^*)^{-\rho_i^*} = \mathsf{H}_1(m^*)^{\text{sk}_i}$  and generates a new zero-knowledge proof  $\pi_i$  using  $\text{sk}_i$  and  $\phi_i$ .  $\mathcal{B}_1$  then returns the set of indices  $\mathcal{S}$ , the message  $m^*$  and evaluation shares  $\{(\phi_i, \pi_i)\}_{i \in \mathcal{S} \cap \mathcal{C}}$  to game **th-prand**<sub>TVRF</sub>. Upon receiving the challenge value  $\phi$  from the underlying game,  $\mathcal{B}_1$  computes  $\phi^* = \phi \cdot \mathsf{H}_1(m^*)^{\rho^*}$  and returns it to  $\mathcal{A}$ . Note that if  $\phi$  was chosen randomly by the **th-prand**<sub>TVRF</sub> game then  $\phi^*$  is also random, and if  $\phi$  is a valid TVRF output, then so is  $\phi^*$  under the key randomized with  $\rho^*$ .  $\mathcal{B}_1$  then simply relays the output of  $\mathcal{A}$  to its own game.

It is easy to see that if  $\mathcal{A}$  can distinguish between a random value and the output of the rerandomized TVRF,  $\mathcal{B}_1$  can distinguish between a random value and the output of the TVRF.