BIP32-Compatible Threshold Wallets

Poulami Das [2] Andreas Erwig [1] Sebastian Faust [1] Julian Loss [2]
Siavash Riahi [1]

[1] TU Darmstadt, Germany
firstname.lastname@tu-darmstadt.de
[2] CISPA Helmholtz Center of Information Security, Germany
lossjulian@gmail.com
poulami.das@cispa.de

Abstract

Cryptographic wallets have become an essential tool to secure users' secret keys and consequently their funds in Blockchain networks. The most prominent wallet standard that is widely adopted in practice is the BIP32 specification. This standard specifies so-called hierarchical deterministic wallets, which are organized in a tree-like structure such that each node in the tree represents a wallet instance and such that a parent node can derive a new child node in a deterministic fashion. BIP32 considers two types of child nodes, namely non-hardened and hardened nodes, which differ in the security guarantees they provide. While the corruption of a hardened wallet does not affect the security of any other wallet instance in the tree, the corruption of a non-hardened node leads to a breach of the entire scheme.

In this work, we address this significant drawback of non-hardened nodes by laying out the design for the first hierarchical deterministic wallet scheme with thresholdized non-hardened nodes. 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 further observe that the derivation of hardened child wallets according to the BIP32 specification does not translate easily to the threshold setting. Therefore, we devise a new and efficient derivation mechanism for hardened wallets in the threshold setting that satisfies the same properties as the original BIP32 derivation mechanism and therefore allows for efficient constructions of BIP32-compatible threshold wallets.

1 Introduction

Blockchain technologies gained huge popularity in the past few years as they provide 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, which allows parties in Blockchain networks 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 pk_A, Bob's address which is derived from his public key pk_B and the amount being spent. Alice can then sign the transaction using her secret key sk_A, so that the final transaction has the form "(pk_A pays c coins to pk_B), σ" where σ is Alice's signature on the transaction under her secret key sk_A. Essentially, 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. In order to do so, a cryptocurrency wallet guarantees the secure storage and maintenance of a user's signing keys. While there were several different proposals for such wallet schemes [MPs19, AGKK19, KMOS21], the most widely used in practice is the BIP32 specification [Wik18], which outlines the design of *hierarchical deterministic wallets*.

Hierarchical Deterministic Wallets A hierarchical deterministic wallet scheme is organized in a tree-like structure, where one root wallet deterministically derives child wallets which in turn deterministically derive further child wallets. Each wallet in the tree is identified by an ID and consists of a signing secret and public key pair $(\mathsf{sk}_{\mathsf{ID}}, \mathsf{pk}_{\mathsf{ID}})$ as well as a so-called chaincode $\mathsf{ch}_{\mathsf{ID}}$. BIP32 considers two types of child

wallets, non-hardened and hardened child wallets, which differ only in how they are derived from the parent. More concretely, a parent wallet identified by ID derives a non-hardened wallet with identifier ID' by first computing $(\rho, \mathsf{ch}_{\mathsf{ID}'}) \leftarrow \mathsf{H}(\mathsf{pk}_{\mathsf{ID}}, \mathsf{ch}_{\mathsf{ID}}, \mathsf{ID}')$ and then $\mathsf{sk}_{\mathsf{ID}'} \leftarrow \mathsf{sk}_{\mathsf{ID}} + \rho$ and $\mathsf{pk}_{\mathsf{ID}'} \leftarrow \mathsf{pk}_{\mathsf{ID}} \cdot g^{\rho}$ where H is a cryptographic hash function. The values $(\mathsf{sk}_{\mathsf{ID}'}, \mathsf{pk}_{\mathsf{ID}'})$ and $\mathsf{ch}_{\mathsf{ID}'}$ then form the key pair and chaincode of the non-hardened child wallet. In contrast, the derivation of a hardened wallet differs from the above in the sense that the computation of ρ and $\mathsf{ch}_{\mathsf{ID}'}$ is carried out as $(\rho, \mathsf{ch}_{\mathsf{ID}'}) \leftarrow \mathsf{H}(\mathsf{sk}_{\mathsf{ID}}, \mathsf{ch}_{\mathsf{ID}}, \mathsf{ID}')$, i.e., it uses the parent's secret key $\mathsf{sk}_{\mathsf{ID}}$ as input to the hash function evaluation instead of its public key $\mathsf{pk}_{\mathsf{ID}}$. BIP32 considers these two different types of child wallets as a trade-off between usability and security. On the one hand, the derivation of a hardened wallet's public key requires knowledge of the parent secret key $\mathsf{sk}_{\mathsf{ID}}$, but provides strong security guarantees since its corruption does not affect the security of the parent node and the remaining wallets in the tree. On the other hand, the derivation of a non-hardened wallet's public key can be done by any party knowing $\mathsf{pk}_{\mathsf{ID}}$ and $\mathsf{ch}_{\mathsf{ID}}$, i.e., without knowledge of the parent secret key. This is particularly useful in the hot/cold setting (see below), where the secret key is stored in an offline device and therefore not accessible at any time. However, the corruption of a single non-hardened wallet breaks the security of the entire wallet scheme, which poses a significant security risk and severely restricts the usage of non-hardened wallets in practice.

Hot/Cold Setting Hierarchical deterministic wallets were first formally modeled and analyzed by Das et al. [DEF+21]. In their work, the authors propose to implement non-hardened wallets in the hot/cold wallet setting in order to mitigate the risk of non-hardened wallets being corrupted. In this setting, a non-hardened wallet consists of two devices, a hot wallet and a cold wallet. The hot wallet is permanently connected to the Internet and stores the public key and the chaincode whereas the cold wallet stores the secret key and the chaincode and remains offline most of the time. The assumption is then that an adversary cannot corrupt the cold wallet (and therefore does not learn the secret key) since it is mostly offline. However, this assumption might not hold where, e.g., an adversary obtains physical access to the cold wallet. Therefore, the following natural question arises:

Can we construct a secure BIP32 wallet scheme that does not rely on the idealized assumption of incorruptible cold wallets?

1.1 Our Contribution

In this work, we answer the above question affirmatively. To this end, we consider thresholdizing non-hardened nodes, 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.

From an application perspective such a thresholdized wallet scheme can be used as a core building block for strengthening the security of so-called custodial (e.g., Coinbase [1] or BitGo [2]), self-custodial (e.g., Electrum or Trezor [Ele13, Tre14]), or shared-custodial wallets (e.g., ZenGo [3] or Sepior [4]). Custodial wallets are maintained solely by a service provider on behalf of a user, while a self-custodial wallet is maintained completely by the user itself. Naturally, while the former requires the user to trust the service provider completely, the latter is cumbersome to use as the user must possess the technical expertise to securely store its signing keys. As a trade-off between custodial and self-custodial wallets, the concept of shared-custodial wallets offers a solution where the secret key is shared among both the user and a service provider such that none can generate a valid signature without the other. For instance, a simple shared-custodial wallet can be instantiated from a $(2, 2)$-threshold wallet scheme. Of course, this can be extended to higher threshold parameters with our generic (t, n) construction. Due to these various useful

[1] https://custody.coinbase.com/
[2] https://www.bitgo.com/
[3] https://zengo.com/
[4] https://sepior.com/

applications, it has been an open question how to construct such BIP32-compatible threshold wallets. [5] Let us now summarize our contributions in more details.

Threshold Signature Schemes with Rerandomizable Keys Das et al. [DEF+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 then show an instantiation based on the threshold ECDSA scheme of Gennaro and Goldfeder [GG18]. Importantly, for our instantiation 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 in order 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 Gennaro and Goldfeder for our instantiation for the following two reasons: (1) it is a relatively simple scheme, i.e., it does not include advanced features such as offline signing or proactive/adaptive security which significantly increase the complexity of other threshold ECDSA schemes; (2) several threshold ECDSA schemes directly build upon the protocol of Gennaro and Goldfeder [CGG+20],[DMZ+21],[CCL+20],[CCL+21], improving either its efficiency, functionality, or security. Since the general idea of these schemes is similar to the original scheme of Gennaro and Goldfeder, we believe that our results can be extended to these schemes as well. 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 interactive multi-party computation (MPC) protocol which securely computes this hash function evaluation. This is however highly inefficient and hence not a practical solution. Our main technical contribution is to provide an alternative and highly efficient 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, during the derivation of 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 would like to emphasize that the TVRF keys are only required for the derivation of hardened nodes. Yet, in practice most 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

[5] https://cyber.biu.ac.il/wp-content/uploads/2022/11/Building-MPC-Wallets.pdf

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.

1.2 Related Work

Cryptographic Wallets There has been a plethora of works on cryptographic wallets such as [MPs19, AGKK19, CEV14, 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. Finally, Erwig and Riahi [ER22] proposed deterministic wallets with support for adaptor signatures.

Threshold ECDSA In recent years, there has been huge interest on threshold ECDSA (e.g., [Lin17, LN18, CGG+20, DMZ+21, CCL+20, CCL+21, BMP22]). 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]. A recent work by 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-specifc 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 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 $\{1, \cdots, l\}$ and for an algorithm A, we denote by $y \leftarrow A(x)$ the execution of A on input x that outputs y. We use the notation $y \in A(x)$ 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 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 $\{P_1, \cdots, P_n\}$ and an interactive algorithm Π, we denote by $\langle \Pi(x_1), \cdots, \Pi(x_n) \rangle$ the joint execution of Π by all parties P_i for $i \in [n]$ with respective inputs x_i.

2.1 Interactive Threshold Signature Scheme

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

Definition 2.1 (Interactive Threshold Signature Scheme). An interactive (t, n)-threshold signature scheme TSig is executed among a set of n parties $\{P_1, \cdots, P_n\}$ and consists of a tuple of procedures $\mathsf{TSig} = (\mathsf{Gen}, \mathsf{TSign}, \mathsf{Verify})$ which are defined as follows:

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

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

Definition 2.2 (Unforgeability of interactive threshold signature schemes). An interactive (t, n)-threshold signature scheme 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 $\textbf{th-ufcma}_{\mathsf{TSig}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\textbf{th-ufcma}_{\mathsf{TSig}}} := \Pr[\textbf{th-ufcma}^{\mathcal{A}}_{\mathsf{TSig}} = 1]$.

Game $\textbf{th-ufcma}_{\mathsf{TSig}}$:

- The adversary $\mathcal{A}$ outputs a list of corrupted parties C, such that $|\mathsf{C}| \leq t$ and for all $i \in \mathsf{C}$ it holds that $i \in [n]$.
- The game initializes a list $\mathsf{SigList} \leftarrow \{\epsilon\}$ and executes $(\{\mathsf{sk}_1, \cdots, \mathsf{sk}_n\}, \mathsf{pk}) \leftarrow \mathsf{TSig.Gen}(1^\kappa, t, n)$. Then $\mathcal{A}$ is run on input pk and $\{\mathsf{sk}_i\}_{i \in \mathsf{C}}$.
- The adversary obtains access to the following `Sign` oracle: On input message m, the oracle and the adversary jointly execute the procedure $\mathsf{TSig.TSign}$, where the oracle runs all honest parties P_i on input (sk_i, m). The message m is then stored in $\mathsf{SigList}$.
- Eventually, the adversary outputs a forgery σ^* and a message m^*. The adversary wins the game, if the following conditions hold: (1) $\mathsf{TSig.Verify}(\mathsf{pk}^*, m^*, \sigma^*) = 1$ and (2) $m^* \notin \mathsf{SigList}$.

2.2 Signature Scheme with Honestly Rerandomizable Keys

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

Definition 2.3 (Signature Scheme with Perfectly Rerandomizable Keys). Let the public parameters par define a randomness space $\mathcal{R} := \mathcal{R}(\mathsf{par})$. A signature scheme with perfectly rerandomizable keys is then a tuple of algorithms $\mathsf{RSig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{RandSK}, \mathsf{RandPK})$ where $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ are the standard algorithms of a signature scheme. The algorithms RandSK and RandPK are defined as follows:

- $\mathsf{RandSK}(\mathsf{sk}, \rho)$: The deterministic *secret key rerandomization algorithm* RandSK takes as input a secret key sk and randomness $\rho \in \mathcal{R}$ and outputs a rerandomized secret key sk'.
- $\mathsf{RandPK}(\mathsf{pk}, \rho)$: The deterministic *public key rerandomization algorithm* RandPK takes as input a public key pk and randomness $\rho \in \mathcal{R}$ and outputs a rerandomized public key 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+21] in Appendix A.1. Further, we recall the construction of an additively rerandomizable and **uf-cma-hrk1**-secure ECDSA signature scheme as presented by Das et al. [DEF+21] in Appendix A.3.

2.3 Non-Interactive Threshold Verifiable Random Function

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

Definition 2.4 A non-interactive (t, n)-threshold verifiable random function (TVRF) is defined w.r.t. to a randomness space Rand and is executed among n parties $\{P_1, \cdots, P_n\}$. It consists of a tuple of algorithms $\mathsf{TVRF} = (\mathsf{Gen}, \mathsf{PEval}, \mathsf{Combine}, \mathsf{Verify})$ which are defined as follows:

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

A TVRF must satisfy the properties *uniqueness*, *pseudorandomness*, and *robustness*.

Definition 2.5 (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 $\mathbf{th\text{-}unique}_{\mathsf{TVRF}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{th\text{-}unique}_{\mathsf{TVRF}}} \coloneqq \Pr[\mathbf{th\text{-}unique}^{\mathcal{A}}_{\mathsf{TVRF}} = 1]$.

Game $\mathbf{th\text{-}unique}_{\mathsf{TVRF}}$:

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

Definition 2.6 (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 $\mathbf{th\text{-}prand}_{\mathsf{TVRF}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{th\text{-}prand}_{\mathsf{TVRF}}} \coloneqq \Pr[\mathbf{th\text{-}prand}^{\mathcal{A}}_{\mathsf{TVRF}} = 1] - \frac{1}{2}$.

Game $\mathbf{th\text{-}prand}_{\mathsf{TVRF}}$:

- The adversary $\mathcal{A}$ outputs a list of corrupted parties C, such that $|\mathsf{C}| \leq t$ and for all $i \in \mathsf{C}$ it holds that $i \in [n]$.

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

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

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

Definition 2.7 (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 game $\textbf{th-robust}_{\mathsf{TVRF}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\textbf{th-robust}_{\mathsf{TVRF}}} \coloneqq \Pr[\textbf{th-robust}^{\mathcal{A}}_{\mathsf{TVRF}} = 1]$.

Game $\textbf{th-robust}_{\mathsf{TVRF}}$:

- The game differs from game $\textbf{th-prand}_{\mathsf{TVRF}}$ 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 $\{\phi_i, \pi_i\}_{i \in \mathcal{S} \cap \mathsf{C}}$. The game computes $(\phi_i, \pi_i) \leftarrow \mathsf{TVRF.PEval}(m^*, \mathsf{sk}_i, \mathsf{pk})$ for all $i \in \mathcal{S} \setminus \mathsf{C}$. The game finally sets $(\phi^*, \pi^*) \leftarrow \mathsf{TVRF.Combine}(\mathsf{pk}, \mathcal{S}, \{\phi_i, \pi_i\}_{i \in \mathcal{S}})$. If $\phi^* \neq \perp$ and $\mathsf{TVRF.Verify}(\mathsf{pk}, m^*, \phi^*, \pi^*) = 0$, the game outputs 1 and 0 otherwise.

3 Rerandomizable Interactive Threshold Signing

3.1 Model

In the following, we introduce the notion of interactive threshold signature schemes with rerandomizable keys. More concretely, we extend the standard notion of a threshold signature scheme by two algorithms RandSK and RandPK, which allow to individually 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 that corresponds to the derived public key.

Definition 3.1 (Interactive Threshold Signature Scheme With Rerandomizable Keys). An interactive (t, n)-threshold signature scheme with rerandomizable keys is a tuple of procedures $\mathsf{RTSig} = (\mathsf{Gen}, \mathsf{RandSK}, \mathsf{RandPK}, \mathsf{TSign}, \mathsf{Verify})$ where $(\mathsf{Gen}, \mathsf{TSign}, \mathsf{Verify})$ are defined as for interactive (t, n)-threshold signatures. We assume that the public parameters par define a randomness space $\mathcal{R} \coloneqq \mathcal{R}(\mathsf{par})$. The algorithms RandSK and RandPK are defined as:

- $\mathsf{RandSK}(i, \mathsf{sk}_i, \rho)$: The deterministic secret key share rerandomization algorithm takes as input an index $i \in [n]$, a secret key share sk_i and a randomness $\rho \in \mathcal{R}$ and it outputs a rerandomized secret key share sk'_i.
- $\mathsf{RandPK}(\mathsf{pk}, \rho)$: The deterministic public key rerandomization algorithm takes as input a public key pk and a randomness $\rho \in \mathcal{R}$ and it outputs a rerandomized public key 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 $(\cdot, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa, t, n)$ and all $\rho \xleftarrow{\$} \mathcal{R}$, the distributions of pk' and pk'' are computationally indistinguishable, where $\mathsf{pk}' \leftarrow \mathsf{RandPK}(\mathsf{pk}, \rho)$ and $(\cdot, \mathsf{pk}'') \leftarrow \mathsf{Gen}(1^\kappa, t, n)$.

- *Correctness under rerandomized keys*: For all $\kappa \in \mathbb{N}$, all $t, n \in \mathbb{N}$ with $t < n$, all $(\{\mathsf{sk}_1, \cdots, \mathsf{sk}_n\}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa, t, n)$, all $\rho \xleftarrow{\$} \mathcal{R}$ and all $m \in \{0,1\}^*$, the rerandomized keys $\{\mathsf{sk}'_i\}_{i\in[n]} \leftarrow \{\mathsf{RandSK}(i, \mathsf{sk}_i, \rho)\}_{i\in[n]}$ and $\mathsf{pk}' \leftarrow \mathsf{RandPK}(\mathsf{pk}, \rho)$ satisfy:

$$\Pr[\mathsf{Verify}(\mathsf{pk}', m, \sigma) | \sigma \leftarrow \langle \mathsf{TSign}(\mathsf{sk}'_1, m), \cdots, \mathsf{TSign}(\mathsf{sk}'_n, m) \rangle] = 1$$

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. Appendix A) 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+20], this weaker rerandomizability property is sufficient for the wallet setting. At a high level, that is because this notion is required to ensure 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. 2.2) 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` cannot only generate signatures under the initial set of keys $(\{\mathsf{sk}_1, \cdots, \mathsf{sk}_n\}, \mathsf{pk})$, but also under key sets that have been 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+21] this weaker notion is sufficient for the wallet setting.

Definition 3.2 (One-per message unforgeability of interactive threshold signature schemes with honestly rerandomizable keys). An interactive (t, n)-threshold signature scheme with rerandomizable keys $\mathsf{RTSig} = (\mathsf{Gen}, \mathsf{RandSK}, \mathsf{RandPK}, \mathsf{TSign}, \mathsf{Verify})$ is **th-ufcma-hrk1**-secure if no PPT adversary $\mathcal{A}$ wins game **th-ufcma-hrk1** as described below with more than negligible probability in the security parameter κ.

Game **th-ufcma-hrk1**$_{\mathsf{RTSig}}$:

- The adversary $\mathcal{A}$ outputs a list of corrupted parties C, such that $|\mathsf{C}| \leq t$ and for all $i \in \mathsf{C}$ it holds that $i \in [n]$.

- The game initializes two lists $\mathsf{RList} \leftarrow \{\epsilon\}$ and $\mathsf{SigList} \leftarrow \{\epsilon\}$ and executes $(\{\mathsf{sk}_1, \cdots, \mathsf{sk}_n\}, \mathsf{pk}) \leftarrow \mathsf{RTSig.Gen}(1^\kappa, t, n)$. Then, $\mathcal{A}$ is run on input pk and $\{\mathsf{sk}_i\}_{i\in\mathsf{C}}$.

- The adversary obtains access to the following two oracles:

  - `Rand`: This oracle, upon a query, samples $\rho \xleftarrow{\$} \mathcal{R}$, stores ρ in RList and outputs ρ to $\mathcal{A}$.
  - `RSign`: On input message m and a randomness ρ, the oracle checks whether $\rho \notin \mathsf{RList}$ and if so outputs $\perp$. Otherwise, it derives a public key and secret key shares for honest parties with the randomness ρ, i.e., it computes $\mathsf{pk}' \leftarrow \mathsf{RTSig.RandPK}(\mathsf{pk}, \rho)$ and $\mathsf{sk}'_i \leftarrow \mathsf{RTSig.RandSK}(i, \mathsf{sk}_i, \rho)$ for all $i \in \{1, \cdots, n\} \setminus \mathsf{C}$. If $(\mathsf{pk}', m) \in \mathsf{SigList}$ then the oracle returns $\perp$. Otherwise, the oracle and the adversary jointly execute the procedure $\mathsf{RTSig.TSign}$, where the oracle runs all honest parties P_i on input (sk'_i, m). The oracle then stores the tuple (pk', m) in $\mathsf{SigList}$.

- Eventually, the adversary outputs a forgery σ^*, a message m^* and a public key $\mathsf{pk}^* \leftarrow \mathsf{RTSig.RandPK}(\mathsf{pk}, \rho^*)$. The adversary wins the game, if the following conditions hold: (1) $\rho^* \in \mathsf{RList}$, (2) $(\mathsf{pk}^*, m^*) \notin \mathsf{SigList}$, and (3) $\mathsf{RTSig.Verify}(\mathsf{pk}^*, m^*, \sigma^*) = 1$.

3.2 Construction

We show how to extend the interactive threshold ECDSA scheme as proposed by Gennaro and Goldfeder [GG18] (which we denote by $\mathsf{GG}[\mathsf{H}_0]$) to an interactive threshold ECDSA scheme with rerandomizable keys (which we denote by $\mathsf{rGG}[\mathsf{H}_0]$). We recall the scheme of Gennaro and Goldfeder (with a slight adjustment) in detail in Appendix B. In Figure 1, we describe our $\mathsf{rGG}[\mathsf{H}_0]$ scheme w.r.t. the $\mathsf{GG}[\mathsf{H}_0]$ 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 $(\mathsf{pk}, \mathsf{sk})$ is simply computed as $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q$ and $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$. In the $\mathsf{GG}[\mathsf{H}_0]$ 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 $\mathsf{rGG}[\mathsf{H}_0]$ scheme, we extend the $\mathsf{GG}[\mathsf{H}_0]$ scheme by providing algorithms RandSK and RandPK which deterministically rerandomize the secret key shares and the public key respectively w.r.t. a randomness ρ. At a high level, in order to rerandomize the secret key share sk_i of party P_i with randomness ρ, the 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 ρ_i, which is then added to the existing secret key share to compute the rerandomized secret key share $\mathsf{sk}'_i \leftarrow \mathsf{sk}_i + \rho_i \mod q$. That is, sk'_i is essentially a share of the secret key $\mathsf{sk} + \rho \mod q$. The RandPK algorithm works correspondingly for the public key and public key shares.

The security of our $\mathsf{rGG}[\mathsf{H}_0]$ 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], which we recall in 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 $\mathsf{rGG}[\mathsf{H}_0]$ scheme to the (one-per message) unforgeability of the scheme of Das et al., we require public key prefixing in our scheme as well.

Algorithm $\mathsf{Gen}(1^\kappa, t, n)$
00 Return $\mathsf{GG.Gen}(1^\kappa, t, n)$

Algorithm $\mathsf{RandSK}(i, \mathsf{sk}_i, \rho)$
00 For $k \in [t] : a_k \leftarrow \mathsf{H}_0(\rho, k)$
01 Let $F(x) := a_t x^t + \cdots + a_1 x + \rho$
02 $\rho_i \leftarrow F(i) \mod q$
03 $\mathsf{sk}'_i \leftarrow \mathsf{sk}_i + \rho_i \mod q$
04 Return sk'_i

Protocol $\mathsf{TSign}(\mathsf{sk}_i, m)$
00 $m' \leftarrow (\mathsf{pk}, m)$
01 Return $\mathsf{GG.TSign}(\mathsf{sk}_i, m')$

Algorithm $\mathsf{RandPK}(\mathsf{pk}, \rho)$
00 Parse $\mathsf{pk} := (X, (X_1, \cdots, X_n))$
01 For $k \in [t] : a_k \leftarrow \mathsf{H}_0(\rho, k)$
02 Let $F(x) := a_t x^t + \cdots + a_1 x + \rho$
03 $\rho_i \leftarrow F(i) \mod q$
04 For $i \in [n] : X'_i \leftarrow X_i \cdot g^{\rho_i}$
05 Return $\mathsf{pk}' := (X \cdot g^\rho, (X'_1, \cdots, X'_n))$

Algorithm $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$
00 $m' \leftarrow (\mathsf{pk}, m)$
01 Return $\mathsf{GG.Verify}(\mathsf{pk}, m', \sigma)$

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

It is easy to see that the $\mathsf{rGG}[\mathsf{H}_0]$ scheme satisfies the *correctness under rerandomized keys* property.

Lemma 3.3 *Let $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{Z}_q$ 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 $\mathsf{rGG}[\mathsf{H}_0]$ 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 distributions

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

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^ρ as input. $\mathcal{B}$ then samples $(\cdot, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa, t, n)$ and computes pk' by executing algorithm $\mathsf{RandPK}(\mathsf{pk}, \rho)$, however with the following differences: (1) it samples the coefficients a_k for $k \in [t]$ of polynomial F uniformly at random from $\mathbb{Z}_q$ instead of computing them as $\mathsf{H}_0(\rho, k)$; and (2) it evaluates polynomial F only in the exponent, i.e., it computes $g^{\rho_i} \leftarrow g^{F(i)}$ for $i \in [n]$.[7] Note that $\mathcal{A}$ can only detect these changes if it queries the random oracle H_0 on input (ρ, k). However, if $\mathcal{A}$ makes a query of this form, adversary $\mathcal{B}$ learns ρ, which is the discrete logarithm of its discrete logarithm challenge g^ρ. Otherwise, public key pk' is identically distributed to a freshly generated public key pk''. ■

Theorem 3.4 *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 equivocable commitment scheme. Further, let the DDH assumption hold in* $\mathbb{G}$ *and let* $\mathsf{rECDSA}[\mathsf{H}_0]$ *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* $\mathsf{rGG}[\mathsf{H}_0]$ *as described above is* **th-ufcma-hrk1***-secure.*

Sketch. Gennaro and Goldfeder prove the $\mathsf{GG}[\mathsf{H}_0]$ scheme unforgeable via reduction to the unforgeability of the single party ECDSA signature scheme. That is, they provide a reduction that simulates game $\textbf{th-ufcma}_{\mathsf{GG}[\mathsf{H}_0]}$ (cf. Definition 2.2) 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[8]. We can prove the above theorem in the same way, with the difference that we reduce the $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}[\mathsf{H}_0]}$ security to the $\textbf{uf-cma-hrk1}_{\mathsf{rECDSA}[\mathsf{H}_0]}$ security. That is, we have to provide a reduction that simulates game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}[\mathsf{H}_0]}$ to a PPT adversary while having access to the $\mathtt{RSign}$ and $\mathtt{Rand}$ oracles of game $\textbf{uf-cma-hrk1}_{\mathsf{rECDSA}[\mathsf{H}_0]}$. In fact, we can use the same simulation as the one from Gennaro and Goldfeder with the following differences: (1) Upon the adversary querying the $\mathtt{Rand}$ oracle in game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}[\mathsf{H}_0]}$, the reduction relays the query to its own $\mathtt{Rand}$ oracle in game $\textbf{uf-cma-hrk1}_{\mathsf{rECDSA}[\mathsf{H}_0]}$; (2) Upon the adversary querying oracle $\mathtt{RSign}$ in game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}[\mathsf{H}_0]}$ on input a message m and randomness ρ, the reduction first rerandomizes the secret key shares sk_i of corrupted parties $P_i \in \mathsf{C}$ by computing $\mathsf{sk}_i' \leftarrow \mathsf{RandSK}(i, \mathsf{sk}_i, \rho)$ as well as the public key $\mathsf{pk}' \leftarrow \mathsf{RandPK}(\mathsf{pk}, \rho)$. The reduction then queries its own signing oracle on input m and ρ and uses the resulting signature and the rerandomized keys for the simulation of the $\mathtt{RSign}$ oracle of game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}[\mathsf{H}_0]}$. These changes do not have any impact on the indistinguishability arguments and reduction from Gennaro and Goldfeder. Note that, since we essentially repeat the proof of Gennaro and Goldfeder, we must also repeat the assumptions their proof relies on in our theorem statement. ■

4 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 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. 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+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 $(\mathsf{sk}_{\mathsf{ID}}, \mathsf{pk}_{\mathsf{ID}})$ and chaincode $\mathsf{ch}_{\mathsf{ID}}$ by computing $(\rho, \mathsf{ch}_{\mathsf{ID}'}) \leftarrow \mathsf{H}(\mathsf{pk}_{\mathsf{ID}}, \mathsf{ch}_{\mathsf{ID}}, \mathsf{ID}')$, $\mathsf{sk}_{\mathsf{ID}'} \leftarrow \mathsf{sk}_{\mathsf{ID}} + \rho$ and $\mathsf{pk}_{\mathsf{ID}'} \leftarrow \mathsf{pk}_{\mathsf{ID}} \cdot g^\rho$. The derivation of a hardened node works in the same way only that the tuple $(\rho, \mathsf{ch}_{\mathsf{ID}'})$ is computed as $\mathsf{H}(\mathsf{sk}_{\mathsf{ID}}, \mathsf{ch}_{\mathsf{ID}}, \mathsf{ID}')$. We now analyze these derivation mechanisms

[7] This step is necessary because $\mathcal{B}$ only knows g^ρ but not ρ. Therefore, $\mathcal{B}$ can only compute F in the exponent.

[8] To be exact, since our $\mathsf{GG}[\mathsf{H}_0]$ scheme differs slightly from the original scheme of Gennaro and Goldfeder, we recall a slightly adjusted simulation. See Appendix B for details.

for the threshold setting w.r.t. to our threshold signature scheme with rerandomizable keys $\mathsf{rGG}[\mathsf{H}_0]$ in more detail.

4.1 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 $\mathsf{sk}_{i,\mathsf{ID}}$ and the chaincode $\mathsf{ch}_{\mathsf{ID}}$ can derive a thresholdized non-hardened child wallet as follows: First, each device of the parent node computes locally $(\rho, \mathsf{ch}_{\mathsf{ID}'}) \leftarrow \mathsf{H}(\mathsf{pk}_{\mathsf{ID}}, \mathsf{ch}_{\mathsf{ID}}, \mathsf{ID}')$ and $\mathsf{sk}_{i,\mathsf{ID}'} \leftarrow \mathsf{rGG}[\mathsf{H}_0].\mathsf{RandSK}(i, \mathsf{sk}_{i,\mathsf{ID}}, \rho)$. Then the devices of the parent node must forward the rerandomized secret key shares $\mathsf{sk}_{i,\mathsf{ID}'}$ to the n devices of the child node. This forwarding 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 $\mathsf{sk}_{i,\mathsf{ID}'}$ 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+19, SLL10]), 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.

4.2 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 $(\rho, \mathsf{ch}_{\mathsf{ID}'}) \leftarrow \mathsf{H}(\mathsf{sk}_{\mathsf{ID}}, \mathsf{ch}_{\mathsf{ID}}, \mathsf{ID}')$, i.e., the evaluation of a hash function on input the parent secret key. In the threshold setting, however, the secret key $\mathsf{sk}_{\mathsf{ID}}$ is shared among n devices such that no single device knows the full key. It is therefore not at all clear how $\mathsf{H}(\mathsf{sk}_{\mathsf{ID}}, \mathsf{ch}_{\mathsf{ID}}, \mathsf{ID}')$ can be computed efficiently without naively reconstructing $\mathsf{sk}_{\mathsf{ID}}$ (which would trivially break the security of the wallet). Furthermore, in the hardened derivation, each parent device can only learn a randomness *share* ρ_i instead of the entire randomness ρ. To see why that is, consider the setting where an adversary corrupts the hardened node, thereby learning its secret key $\mathsf{sk}_{\mathsf{ID}} + \rho$, as well as a parent node device, thereby learning ρ. 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 ϕ_i and proof π_i on a message m. Given at least $t+1$ evaluation shares for m, any party can deterministically compute a pseudorandom value ϕ and a proof π and given the public key pk, ϕ and π, any party can verify that ϕ was computed correctly. We recall the formal definition of a TVRF in Section 2.3.

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 ϕ_i and the corresponding proof π_i on ID. It then sends (ϕ_i, π_i) to the hardened node, which combines $t+1$ shares to a pseudorandom

seed ϕ. The hardened node then verifies the correctness of ϕ 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 2.

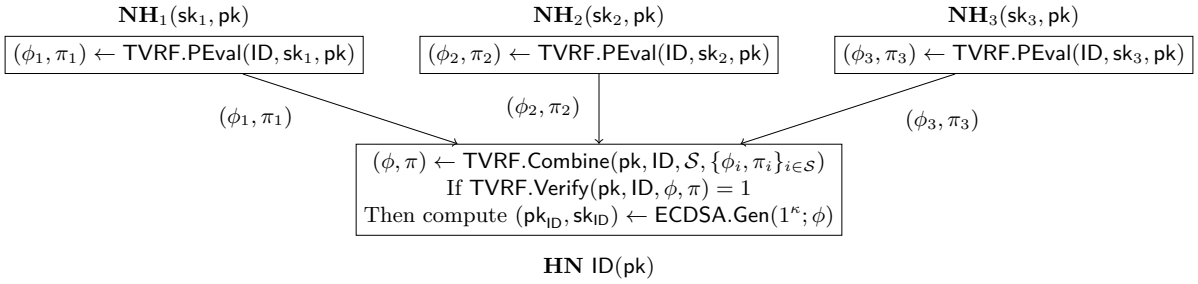

Figure 2: Pictorial representation of our improved hardened node derivation mechanism in the threshold setting. Each of the three devices $\mathbf{NH}_1$, $\mathbf{NH}_2$, $\mathbf{NH}_3$ of the non-hardened parent node stores a TVRF public key pk and secret key share sk_i for $i \in [3]$. In order to 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 can then choose a subset $\mathcal{S}$ of $[3]$, combine the corresponding evaluation shares to a full random value ϕ, verify that the non-hardened devices in $\mathcal{S}$ behaved honestly, and then use ϕ as input to the key generation algorithm of the ECDSA signature scheme. Note that this key generation is deterministic, since we explicitly give the randomness ϕ 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 $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q$ and $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$. 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 3 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.

4.3 Joint Threshold Signature/TVRF Scheme

We define the scheme $\mathsf{TVRF\text{-}rGG}[\mathsf{H}_0, \mathsf{H}_1]$, which consists of all procedures of $\mathsf{rGG}[\mathsf{H}_0]$ and $\mathsf{TVRF}[\mathsf{H}_1]$, except that it uses only one of $\mathsf{rGG}[\mathsf{H}_0].\mathsf{Gen}$ and $\mathsf{TVRF}[\mathsf{H}_1].\mathsf{Gen}$. Concretely, $\mathsf{TVRF\text{-}rGG}[\mathsf{H}_0, \mathsf{H}_1]$ consists of the procedures

$$\begin{aligned}\mathsf{TVRF\text{-}rGG}[\mathsf{H}_0, \mathsf{H}_1] = (&\mathsf{rGG}[\mathsf{H}_0].\mathsf{Gen}, \mathsf{rGG}[\mathsf{H}_0].\mathsf{RandSK}, \mathsf{rGG}[\mathsf{H}_0].\mathsf{RandPK}, \mathsf{rGG}[\mathsf{H}_0].\mathsf{TSign},\\ &\mathsf{rGG}[\mathsf{H}_0].\mathsf{Verify}, \mathsf{TVRF}[\mathsf{H}_1].\mathsf{PEval}, \mathsf{TVRF}[\mathsf{H}_1].\mathsf{Combine}, \mathsf{TVRF}[\mathsf{H}_1].\mathsf{Verify}).\end{aligned}$$

For simplicity, we sometimes abbreviate the schemes $\mathsf{TVRF\text{-}rGG}[\mathsf{H}_0, \mathsf{H}_1]$, $\mathsf{rGG}[\mathsf{H}_0]$ and $\mathsf{TVRF}[\mathsf{H}_1]$ by $\mathsf{TVRF\text{-}rGG}$, rGG and TVRF respectively. The scheme $\mathsf{TVRF\text{-}rGG}$ is defined w.r.t parameters $t, n \in \mathbb{N}$, s.t. the corruption threshold t is the minimum of the corruption thresholds of schemes rGG and TVRF. That is, for schemes (t', n)-rGG and (t'', n)-TVRF it must hold that $t = \min\{t', t''\}$. The $\mathsf{TVRF\text{-}rGG}$ 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**).

4.3.1 Pseudorandomness of $\mathsf{TVRF\text{-}rGG}$

In the following we define the pseudorandomness property of $\mathsf{TVRF\text{-}rGG}$ via a game **unf-prand** and prove that $\mathsf{TVRF\text{-}rGG}$ satisfies this property. Later in Section 4.3.2, we provide the uniqueness and robustness definitions and argue that the $\mathsf{TVRF\text{-}rGG}$ scheme satisfies them.

Definition 4.1 (Pseudorandomness of $\mathsf{TVRF\text{-}rGG}$). The $\mathsf{TVRF\text{-}rGG}$ 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

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

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

Game $\textbf{unf-prand}_{\mathsf{TVRF\text{-}rGG}}$:

- The adversary $\mathcal{A}$ outputs a list of corrupted parties C, such that $|\mathsf{C}| \leq t$ and for all $i \in \mathsf{C}$ it holds that $i \in [n]$.
- The game initializes $\mathsf{SigList} \leftarrow \{\epsilon\}$, $\mathsf{RList} \leftarrow \{\epsilon\}$ and $\mathsf{EvalList} \leftarrow \{\epsilon\}$ and executes $(\mathsf{pk}, \{\mathsf{sk}_1, \cdots, \mathsf{sk}_n\}) \leftarrow \mathsf{TVRF\text{-}rGG.Gen}(1^\kappa, t, n)$. $\mathcal{A}$ receives as input pk and $\{\mathsf{sk}_i\}_{i \in \mathsf{C}}$.
- The adversary obtains access to the following oracles:
  - `Rand`: Same as in game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}}$.
  - `RSign`: Same as in game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}}$.
  - `REval`: On input message m, index $i \in [n] \setminus \mathsf{C}$ and randomness ρ, check if $\rho \in \mathsf{RList}$ and abort otherwise. The oracle executes

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

    and if $(i, m, \rho) \notin \mathsf{EvalList}$, stores the tuple (i, m, ρ) in $\mathsf{EvalList}$. The oracle returns (ϕ_i, π_i).
- The adversary wins the game if it wins either of the following cases:
  - **Case 1**: Same as in game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}}$.
  - **Case 2**: The adversary outputs a message m^*, a randomness ρ^*, a set of indices $\mathcal{S}$ with $|\mathcal{S}| > t$ and evaluation shares $\{\phi_k, \pi_k\}_{k \in \mathcal{S} \cap \mathsf{C}}$. The game checks if there are less than $t - |\mathcal{S} \cap \mathsf{C}|$ tuples of the form $(\cdot, m^*, \rho^*)$ in $\mathsf{EvalList}$ and if $\rho^* \in \mathsf{RList}$. If so, for $i \in \mathcal{S} \setminus \mathsf{C}$ the game computes

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

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

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

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

Theorem 4.2 *Let* $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{Z}_q$, $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}$ *be hash functions modeled as a random oracle. Let* $\mathsf{rGG}[\mathsf{H}_0]$ *be the* **th-ufcma-hrk1***-secure interactive* (t', n)*-threshold ECDSA scheme with rerandomizable keys from Section 3.2 and let* $\mathsf{TVRF}[\mathsf{H}_1]$ *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 equivocable commitment scheme and the DDH assumption hold in* $\mathbb{G}$*. Then the* (t, n)-$\mathsf{TVRF\text{-}rGG}[\mathsf{H}_0, \mathsf{H}_1]$ *scheme as described above is* **unf-prand***-secure, where* $t = \min\{t', t''\}$.

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 $\textbf{unf-prand}_{\mathsf{TVRF\text{-}rGG}}$ game with more than negligible advantage, then we can construct a PPT adversary $\mathcal{B}$ which can either win the $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}}$ or the $\textbf{th-prand}_{\mathsf{TVRF}}$ game with more than negligible advantage. To this end, $\mathcal{B}$ first guesses if $\mathcal{A}$ is going to win game $\textbf{unf-prand}_{\mathsf{TVRF\text{-}rGG}}$ through **Case 1** or **Case 2**. Depending on the guess, $\mathcal{B}$ decides to either play in game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}}$ or $\textbf{th-prand}_{\mathsf{TVRF}}$, while simulating $\mathcal{A}$'s oracle queries. Note that $\mathcal{B}$ receives only access to the oracles of either game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}}$ *or* $\textbf{th-prand}_{\mathsf{TVRF}}$ which significantly complicates the simulation of $\mathcal{A}$'s oracle queries. In particular, when playing in game $\textbf{th-prand}_{\mathsf{TVRF}}$, $\mathcal{B}$ does not get access to a signing oracle, yet has to simulate oracle $\mathtt{RSign}$ of game $\textbf{unf-prand}_{\mathsf{TVRF\text{-}rGG}}$ to $\mathcal{A}$.

Let $\mathcal{B} := (\mathcal{B}_0, \mathcal{B}_1)$ be composed of two subprocedures. At the beginning of game $\textbf{unf-prand}_{\mathsf{TVRF\text{-}rGG}}$, $\mathcal{B}$ chooses a bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{B}$ executes subprocedure $\mathcal{B}_0$ that plays in game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}}$ and otherwise $\mathcal{B}$ executes $\mathcal{B}_1$ that plays in game $\textbf{th-prand}_{\mathsf{TVRF}}$. 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 $\textbf{unf-prand}_{\mathsf{TVRF\text{-}rGG}}$ to $\mathcal{A}$ and use $\mathcal{A}$'s output to win their respective security games (i.e., either $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}}$ or $\textbf{th-prand}_{\mathsf{TVRF}}$). Finally, after analyzing both cases separately, we determine the advantage of $\mathcal{B} := (\mathcal{B}_0, \mathcal{B}_1)$ to win either game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}}$ or game $\textbf{th-prand}_{\mathsf{TVRF}}$. We additionally provide an intuitive proof sketch for both cases in Appendix B.3.

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 $\textbf{th-ufcma-hrk1}^{\mathcal{B}_0}_{\mathsf{rGG}}$. Game $\boldsymbol{G}_0$: This game is the original $\textbf{unf-prand}_{\mathsf{TVRF\text{-}rGG}}$ game, in which adversary $\mathcal{A}$ can corrupt t parties and receives access to oracles H_0, H_1, $\mathtt{RSign}$, $\mathtt{Rand}$ and $\mathtt{REval}$. We have $\Pr[\textbf{unf-prand}^{\mathcal{A}}_{\mathsf{TVRF\text{-}rGG}} = 1 | \textbf{Case 1}] = \Pr[\boldsymbol{G}^{\mathcal{A}}_0 = 1]$.

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

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 $\Pr[\boldsymbol{G}^{\mathcal{A}}_1 = 1] = \Pr[\boldsymbol{G}^{\mathcal{A}}_0 = 1]$.

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

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

However, instead of outputting (ϕ_i, π_i), the game simulates a zero-knowledge proof π'_i that proves correctness of ϕ_i and outputs (ϕ_i, π'_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. That is, we have that $\Pr[\boldsymbol{G}_1^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G}_2^{\mathcal{A}} = 1] + \mathsf{negl}(\kappa)$ where negl is a negligible function in the security parameter κ

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

This game is equivalent to the previous game since $(X'_i)^r = g^{\mathsf{sk}'_i \cdot r} = \mathsf{H}_1(m)^{\mathsf{sk}'_i}$. Therefore, we have that $\Pr[\boldsymbol{G}_2^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G}_3^{\mathcal{A}} = 1]$.

Reduction to $\mathbf{th\text{-}ufcma\text{-}hrk1}_{\mathsf{rGG}}$: Having shown that the transition from game $\boldsymbol{G}_0$ to game $\boldsymbol{G}_3$ is indistinguishable, it remains to show that an adversary $\mathcal{A}$ winning in game $\boldsymbol{G}_3$ can be used to construct an adversary $\mathcal{B}_0$ that wins game $\mathbf{th\text{-}ufcma\text{-}hrk1}_{\mathsf{rGG}}$. To do so, we must show that $\mathcal{B}_0$ playing in $\mathbf{th\text{-}ufcma\text{-}hrk1}_{\mathsf{rGG}}$ can simulate game $\boldsymbol{G}_3$ to $\mathcal{A}$. The simulation differs from game $\boldsymbol{G}_3$ 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 C in $\mathbf{th\text{-}ufcma\text{-}hrk1}_{\mathsf{rGG}}$ as $\mathcal{A}$ does in $\boldsymbol{G}_3$. $\mathcal{B}_0$ then forwards the public key pk and the secret key shares $\{\mathsf{sk}_i\}_{i \in \mathsf{C}}$ from game $\mathbf{th\text{-}ufcma\text{-}hrk1}_{\mathsf{rGG}}$ to $\mathcal{A}$.
2. Upon $\mathcal{A}$ querying oracle `RSign` on input a message m and a randomness ρ, $\mathcal{B}_0$ queries its own oracle `RSign` on input m and ρ and relays the messages between $\mathcal{A}$ and the `RSign` oracle in game $\mathbf{th\text{-}ufcma\text{-}hrk1}_{\mathsf{rGG}}$.
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 $\boldsymbol{G}_3$ to $\mathcal{A}$. It remains to show that $\mathcal{B}_0$ can use $\mathcal{A}$'s forgery in game $\mathbf{unf\text{-}prand}_{\mathsf{TVRF\text{-}rGG}}$ to win its own game $\mathbf{th\text{-}ufcma\text{-}hrk1}_{\mathsf{rGG}}$. Since $\mathcal{B}_0$ forwards all queries to `RSign` in game $\mathbf{unf\text{-}prand}_{\mathsf{TVRF\text{-}rGG}}$ to the corresponding oracle in game $\mathbf{th\text{-}ufcma\text{-}hrk1}_{\mathsf{rGG}}$, $\mathcal{B}_0$ and $\mathcal{A}$ query their respective oracles on the same messages. Therefore, if $\mathcal{A}$ outputs a valid forgery in $\mathbf{unf\text{-}prand}_{\mathsf{TVRF\text{-}rGG}}$, then the forgery is also valid in $\mathbf{th\text{-}ufcma\text{-}hrk1}_{\mathsf{rGG}}$. We finally have

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

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 $\mathbf{th\text{-}prand}^{\mathcal{B}_1}_{\mathsf{TVRF}}$.

Game $\boldsymbol{G}_0$: This game is the original $\mathbf{unf\text{-}prand}_{\mathsf{TVRF\text{-}rGG}}$ game, in which adversary $\mathcal{A}$ can corrupt t parties and receives access to oracles H_0, H_1, `RSign`, `Rand` and `REval`. We have $\Pr[\mathbf{unf\text{-}prand}^{\mathcal{A}}_{\mathsf{TVRF\text{-}rGG}} = 1 | \mathbf{Case\ 2}] = \Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1]$.

Game $\boldsymbol{G}_1$: This game is similar to the previous game with a difference in the random oracle H_0. Upon $\mathcal{A}$ querying H_0 on a message m, the game checks whether m is public key prefixed, i.e., whether m can be parsed as $m \coloneqq (\mathsf{pk}', m')$ where $\mathsf{pk}' \leftarrow \mathsf{TVRF\text{-}rGG.RandPK}(\mathsf{pk}, \rho)$ for some $\rho \in \mathsf{RList}$. If so and if $\mathsf{H}_0(m) \neq \bot$, then the game returns $\bot$.

This game can only be distinguished from the previous game in case game $\boldsymbol{G}_1$ returns $\bot$ during a query to H_0, i.e., if $\mathcal{A}$ queries H_0 on a message $m \coloneqq (\mathsf{pk}', m')$ before having received the corresponding randomness ρ from the `Rand` oracle. Note that this event only happens with negligible probability since the `Rand` oracle outputs uniformly random values in $\mathbb{Z}_q$. Therefore, we have that $\Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G}_1^{\mathcal{A}} = 1] + \mathsf{negl}_1(\kappa)$ where negl_1 is a negligible function in the security parameter κ.

Game $\boldsymbol{G}_2$: This game is similar to the previous game with two differences. First, in the beginning the game initializes a set $\mathsf{HSigs} := \epsilon$. Second, upon a query to H_0 on input a public key prefixed message $m := (\mathsf{pk}', m')$, the game checks if $\mathsf{H}_0(m) = \bot$ and if so, it executes $\mathcal{S}_{\mathsf{ECDSA}}$ as described in Figure 3 on input (X', m) where $\mathsf{pk}' := (X', \{X'_1, \cdots, X'_n\})$. Finally, the game sets $\mathsf{HSigs}[m] := (r, s)$.

$\mathcal{S}_{\mathsf{ECDSA}}(X, m)$:
$a, b \xleftarrow{\$} \mathbb{Z}_q,\ R = X^a \cdot g^b,\ r = f(R),\ s = \frac{r}{a},\ \mathsf{H}_0(m) := \frac{r \cdot b}{a}$

Figure 3: Simulation of ECDSA signatures via programming of the random oracle H_0 as first presented by Fersch et al. [FKP17]. The function $f : \mathbb{G} \to \mathbb{Z}_q$ is defined as the projection of a group element to its x-coordinate.

It is easy to see that $\mathcal{S}_{\mathsf{ECDSA}}$ programs the random oracle H_0 in such a way that H_0 returns uniformly random values. Therefore, this game is equivalent to the previous game, i.e., $\Pr[\boldsymbol{G}_1^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G}_2^{\mathcal{A}} = 1]$.

Game $\boldsymbol{G}_3$: This game is similar to the previous game with a difference in the `RSign` oracle. On input a message m and a randomness ρ, the game first computes the rerandomized (full) secret key sk' and then generates a full ECDSA signature σ' under sk' for message m. The game then executes the signing procedure in the same way as presented in the proof sketch of Theorem 3.4 using signature σ'.

The indistinguishability of this game to the previous one follows in the same way as in Theorem 3.4. Note that this simulator code does not program H_0 and therefore does not conflict with the execution of $\mathcal{S}_{\mathsf{ECDSA}}$. We have that $\Pr[\boldsymbol{G}_2^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G}_3^{\mathcal{A}} = 1] + \mathsf{negl}_2(\kappa)$ where negl_2 is a negligible function in the security parameter κ.

Game $\boldsymbol{G}_4$: This game is similar to the previous game again with a modification in the `RSign` oracle. On input a message m and a randomness ρ, the game does not generate a full ECDSA signature using sk', but it fetches $(r, s) \leftarrow \mathsf{HSigs}[m']$ for $m' \leftarrow (\mathsf{pk}', m)$ where $\mathsf{pk}' \leftarrow \mathsf{TVRF\text{-}rGG.RandPK}(\mathsf{pk}, \rho)$.[9] The game then uses the tuple (r, s) as the full ECDSA signature under sk'.

As shown in [FKP17], the tuple (r, s) as generated by the $\mathcal{S}_{\mathsf{ECDSA}}$ algorithm (see Figure 3) is computationally indistinguishable from an honestly generated ECDSA signature for message m' under public key X' (where $\mathsf{pk}' := (X', \{X'_1, \cdots, X'_n\})$) to a PPT adversary $\mathcal{A}$ with access to random oracle H_0. Since the simulator code from Theorem 3.4 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 $\Pr[\boldsymbol{G}_3^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G}_4^{\mathcal{A}} = 1] + \mathsf{negl}_3(\kappa)$ where negl_3 is a negligible function in the security parameter κ.

Game $\boldsymbol{G}_5$: 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 ρ, the game computes $\mathsf{sk}'_i \leftarrow \mathsf{TVRF\text{-}rGG.RandSK}(i, \mathsf{sk}_i, \rho)$ and $\mathsf{pk}' \leftarrow \mathsf{TVRF\text{-}rGG.RandPK}(\mathsf{pk}, \rho)$ and executes $(\phi_i, \pi_i) \leftarrow \mathsf{TVRF\text{-}rGG.PEval}(m, \mathsf{sk}'_i, \mathsf{pk}')$. Instead of outputting (ϕ_i, π_i), however, the game simulates a zero-knowledge proof π'_i that proves correctness of ϕ_i. The game then outputs (ϕ_i, π'_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 $\Pr[\boldsymbol{G}_4^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G}_5^{\mathcal{A}} = 1] + \mathsf{negl}_4(\kappa)$ where negl_4 is a negligible function in the security parameter κ.

Game $\boldsymbol{G}_6$: 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 ρ, instead of rerandomizing the secret key share sk_i to sk'_i and executing $\mathsf{TVRF\text{-}rGG.PEval}(m, \mathsf{sk}'_i, \mathsf{pk}')$, the game computes

$$\mathsf{TVRF\text{-}rGG.PEval}(m, \mathsf{sk}_i, \mathsf{pk}) \cdot \mathsf{H}_1(m)^{\rho_i},$$

where ρ_i denotes the randomness share of ρ for party P_i according to the $\mathsf{TVRF\text{-}rGG.RandSK}$ (cf. Figure 1) algorithm.

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

$$\mathsf{TVRF\text{-}rGG.PEval}(m, \mathsf{sk}'_i, \mathsf{pk}') = \mathsf{TVRF\text{-}rGG.PEval}(m, \mathsf{sk}_i, \mathsf{pk}) \cdot \mathsf{H}_1(m)^{\rho_i} = \mathsf{H}_1(m)^{\mathsf{sk}_i + \rho_i}.$$

[9] We assume that H_0 has been queried on m' before the signing query.

Therefore, we have that $\Pr[\boldsymbol{G}_5^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1]$.

Game $\boldsymbol{G}_7$: This game is similar to the previous game with a modification in the challenge phase. Upon $\mathcal{A}$ outputting a message m^*, randomness ρ^*, a set of indices $\mathcal{S}$, and evaluation shares $\{(\phi_k^*, \pi_k^*)\}_{k \in \mathcal{S} \cap \mathsf{C}}$, the game verifies the proofs π_k^* and returns $\perp$ if any proof does not verify. Otherwise the game checks if $\phi_k^* = \mathsf{H}_1(m^*)^{\mathsf{sk}_k + \rho_k^*}$ 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 π_k^* for a false statement. Due to the soundness property of the DLEq proof system, this event happens only with negligible probability. Therefore, we have that $\Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G}_7^{\mathcal{A}} = 1] + \mathsf{negl}_5(\kappa)$ where negl_5 is a negligible function in the security parameter κ.

Game $\boldsymbol{G}_8$: This game is similar to the previous game with a modification in the challenge phase. Namely, upon $\mathcal{A}$ outputting a message m^*, randomness ρ^*, a set of indices $\mathcal{S}$, and evaluation shares $\{(\phi_k^*, \pi_k^*)\}_{k \in \mathcal{S} \cap \mathsf{C}}$ the game computes $\phi_k = \phi_k^* \cdot \mathsf{H}_1(m^*)^{-\rho_k^*} = \mathsf{H}_1(m^*)^{\mathsf{sk}_k}$ and generates a new zero-knowledge proof π_k using sk_k and ϕ_k. The game then computes $(\phi_i, \pi_i) \leftarrow \mathsf{TVRF\text{-}rGG.PEval}(m^*, \mathsf{sk}_i, \mathsf{pk})$ for $i \in \mathcal{S} \setminus \mathsf{C}$ and $(\phi, \pi) \leftarrow \mathsf{TVRF\text{-}rGG.Combine}(\mathsf{pk}, \mathcal{S}, \{(\phi_j, \pi_j)\}_{j \in \mathcal{S}})$. The game chooses uniformly at random $b \xleftarrow{\$} \{0,1\}$ and if $b = 0$ sets $\phi' \coloneqq \phi$ and otherwise sets $\phi' \xleftarrow{\$} \mathbb{G}$. Finally, the game computes $\phi^* = \phi' \cdot \mathsf{H}_1(m^*)^{\rho^*}$ and returns it to $\mathcal{A}$.

Note that if ϕ' was chosen randomly from $\mathbb{G}$ by the game then ϕ^* is also a uniformly random element, and if ϕ' is a valid TVRF output, then so is ϕ^* under the key randomized with ρ^*. This is since $\phi^* = \phi' \cdot \mathsf{H}_1(m^*)^{\rho^*} = \mathsf{H}_1(m^*)^{\mathsf{sk}} \cdot \mathsf{H}_1(m^*)^{\rho^*} = \mathsf{H}_1(m^*)^{\mathsf{sk} + \rho^*}$. We have that $\Pr[\boldsymbol{G}_7^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G}_8^{\mathcal{A}} = 1]$.

Reduction to $\mathbf{th\text{-}prand}_{\mathsf{TVRF}}$: Having shown that the transition from game $\boldsymbol{G}_0$ to game $\boldsymbol{G}_8$ is indistinguishable, it remains to show that an adversary $\mathcal{A}$ winning in game $\boldsymbol{G}_8$ can be used to construct an adversary $\mathcal{B}_1$ that wins game $\mathbf{th\text{-}prand}_{\mathsf{TVRF}}$. To do so, we must show that $\mathcal{B}_1$ playing in $\mathbf{th\text{-}prand}_{\mathsf{TVRF}}$ can simulate game $\boldsymbol{G}_8$ to $\mathcal{A}$. The simulation differs from game $\boldsymbol{G}_8$ 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 C in $\mathbf{th\text{-}prand}_{\mathsf{TVRF}}$ as $\mathcal{A}$ does in $\boldsymbol{G}_8$. $\mathcal{B}_1$ then forwards the public key pk and the secret key shares $\{\mathsf{sk}_i\}_{i \in \mathsf{C}}$ from game $\mathbf{th\text{-}prand}_{\mathsf{TVRF}}$ to $\mathcal{A}$.

2. Upon $\mathcal{A}$ querying oracle `REval` on input a message m, an index i and a randomness ρ, $\mathcal{B}_1$ queries its own oracle `Eval` on input m and i and uses the oracle output to compute the output of `REval` as in $\boldsymbol{G}_8$.

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 $\phi_k = \phi_k^* \cdot \mathsf{H}_1(m^*)^{-\rho_k^*}$ together with the zero-knowledge proofs π_k to its own game and receives an element ϕ^*. $\mathcal{B}_1$ forwards to $\mathcal{A}$ the element $\phi^* \cdot \mathsf{H}_1(m^*)^{\rho^*}$.

It is easy to see that $\mathcal{B}_1$'s simulation is indistinguishable from game $\boldsymbol{G}_8$ to $\mathcal{A}$ and that if $\mathcal{A}$ wins game $\boldsymbol{G}_8$ with more than negligible probability, then $\mathcal{B}_1$ wins game $\mathbf{th\text{-}prand}_{\mathsf{TVRF}}$ with the same probability. The latter is because $\mathcal{B}_1$ makes the same queries to oracle `Eval` as $\mathcal{A}$ does to oracle `REval`. We finally have that

$$\begin{aligned}\Pr[\mathbf{unf\text{-}prand}^{\mathcal{A}}_{\mathsf{TVRF\text{-}rGG}} = 1 | \mathbf{Case\ 2}] = \Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1] &\leq \Pr[\boldsymbol{G}_8^{\mathcal{A}} = 1] + \mathsf{negl}'(\kappa) \\ &= \Pr[\mathbf{th\text{-}prand}^{\mathcal{B}_1}_{\mathsf{TVRF}[\mathsf{H}_1]} = 1] + \mathsf{negl}'(\kappa),\end{aligned}$$

where $\mathsf{negl}'(\kappa) \coloneqq \sum_{i=1}^{5} \mathsf{negl}_i(\kappa)$.

Finally, we determine the advantage of adversary $\mathcal{B} \coloneqq (\mathcal{B}_0, \mathcal{B}_1)$ to win either in game $\mathbf{th\text{-}ufcma\text{-}hrk1}_{\mathsf{rGG}}$ or $\mathbf{th\text{-}prand}_{\mathsf{TVRF}}$. Note that $\mathcal{B}$'s advantage is:

$$\mathsf{Adv}^{\mathcal{B}} \coloneqq \frac{1}{2}\mathsf{Adv}^{\mathcal{B}_0}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{rECDSA}[\mathsf{H}_0]}} \cdot \Pr[\mathbf{Case\ 1}] + \frac{1}{2}\mathsf{Adv}^{\mathcal{B}_1}_{\mathbf{th\text{-}prand}_{\mathsf{TVRF}[\mathsf{H}_1]}} \cdot \Pr[\mathbf{Case\ 2}].$$

Therefore we can conclude that:

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

where $\mathsf{negl}''(\kappa) \coloneqq \mathsf{negl}(\kappa) \cdot \Pr[\mathbf{Case\ 1}] + \mathsf{negl}'(\kappa) \cdot \Pr[\mathbf{Case\ 2}]$ is a negligible function in κ. ■

4.3.2 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 4.3 (Uniqueness of TVRF-rGG). The scheme TVRF-rGG 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 $\mathbf{unf\text{-}unique}_{\mathsf{TVRF\text{-}rGG}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{unf\text{-}unique}} \coloneqq \Pr[\mathbf{unf\text{-}unique}^{\mathcal{A}}_{\mathsf{TVRF\text{-}rGG}} = 1]$.

Game $\mathbf{unf\text{-}unique}_{\mathsf{TVRF\text{-}rGG}}$:

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

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

    Otherwise it outputs 0.

Definition 4.4 (Robustness of TVRF-rGG). The scheme TVRF-rGG 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 $\mathbf{unf\text{-}robust}_{\mathsf{TVRF\text{-}rGG}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{unf\text{-}robust}} \coloneqq \Pr[\mathbf{unf\text{-}robust}^{\mathcal{A}}_{\mathsf{TVRF\text{-}rGG}} = 1]$.

Game **unf-robust**$_{\mathsf{TVRF\text{-}rGG}}$:

- The game is exactly the same as game **unf-prand**$_{\mathsf{TVRF\text{-}rGG}}$, 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**$_{\mathsf{TVRF\text{-}rGG}}$.
  - **Case 2**: The adversary outputs a message m^*, a set $\mathcal{S}$ with $|\mathcal{S}| > t$, a list of evaluation shares $\{\phi_i, \pi_i\}_{i \in \mathcal{S} \cap \mathsf{C}}$ and a randomness ρ^*. The game checks if $\rho^* \in \mathsf{RList}$ and if so computes for all $i \in \mathcal{S} \setminus \mathsf{C}$:

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

    The game finally sets

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

    If $\phi^* \neq \perp$ and $\mathsf{TVRF\text{-}rGG.Verify}(\mathsf{pk}', m^*, \phi^*, \pi^*) = 0$, 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 4.2. 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.

Acknowledgments

This work is supported by the German Research Foundation (DFG) Emmy Noether Program FA 1320/1-1, by the German Research Foundation DFG - SFB 1119 - 236615297 (CROSSING Projects P1 and S7), by the German Federal Ministry of Education and Research (BMBF) *iBlockchain Project* (grant nr. 16KIS0902), 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+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, 8.)

[AGKK19] Myrto Arapinis, Andriana Gkaniatsou, Dimitris Karakostas, and Aggelos Kiayias. A formal treatment of hardware wallets. pages 426–445, 2019. (Cited on page 1, 4.)

[AHS20] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. A survey of ecdsa threshold signing. Cryptology ePrint Archive, Paper 2020/1390, 2020. https://eprint.iacr.org/2020/1390. (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*, page 23–41, Cham, 2015. Springer International Publishing. (Cited on page 11.)

[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.)

[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 11.)

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). pages 11–19, 1988. (Cited on page 11.)

[CCL+20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. pages 266–296, 2020. (Cited on page 3, 4.)

[CCL+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 3, 4.)

[CEV14] Nicolas T. Courtois, Pinar Emirdag, and Filippo Valsorda. Private key recovery combination attacks: On extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor RNG events. Cryptology ePrint Archive, Report 2014/848, 2014. https://eprint.iacr.org/2014/848. (Cited on page 4.)

[CGG+20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. pages 1769–1787, 2020. (Cited on page 3, 4.)

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. pages 89–105, 1993. (Cited on page 23.)

[DEF+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, New York, NY, USA, 2021. Association for Computing Machinery. (Cited on page 2, 3, 4, 5, 8, 9, 10, 23.)

[DFL19] Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. pages 651–668, 2019. (Cited on page 4.)

[DMZ+21] Yi Deng, Shunli Ma, Xinxuan Zhang, Hailong Wang, Xuyang Song, and Xiang Xie. Promise σ-protocol: How to construct efficient threshold ecdsa from encryptions based on class groups. Springer-Verlag, 2021. (Cited on page 3, 4.)

[Dod03] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. pages 1–17, 2003. (Cited on page 3.)

[Ele13] Version bytes for BIP32 extended public and private keys. https://electrum.readthedocs.io/en/latest/xpub_version_bytes.html, 2013. (Cited on page 2.)

[ER22] Andreas Erwig and Siavash Riahi. Deterministic wallets for adaptor signatures. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian D. Jensen, and Weizhi Meng, editors, *Computer Security – ESORICS 2022*, pages 487–506, Cham, 2022. Springer Nature Switzerland. (Cited on page 4.)

[FKM+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 5.)

[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 16, 29.)

[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, New York, NY, USA, 2018. Association for Computing Machinery. (Cited on page 3, 4, 9, 24, 26, 27.)

[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)*, pages 88–102, 2021. (Cited on page 6, 11, 12, 19, 22, 23.)

[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 11.)

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004. (Cited on page 11.)

[GS15] Gus Gutoski and Douglas Stebila. Hierarchical deterministic bitcoin wallets that tolerate key leakage. pages 497–504, 2015. (Cited on page 4.)

[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.)

[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 1, 4.)

[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.)

[Lin17] Yehuda Lindell. Fast secure two-party ECDSA signing. pages 613–644, 2017. (Cited on page 4.)

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1837–1854, New York, NY, USA, 2018. Association for Computing Machinery. (Cited on page 4.)

[MPs19] Antonio Marcedone, Rafael Pass, and abhi shelat. Minimizing trust in hardware wallets with two factor signatures. pages 407–425, 2019. (Cited on page 1, 4.)

[MZW+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, New York, NY, USA, 2019. Association for Computing Machinery. (Cited on page 11.)

[SLL10] David Schultz, Barbara Liskov, and Moses Liskov. Mpss: Mobile proactive secret sharing. *ACM Trans. Inf. Syst. Secur.*, 13(4), dec 2010. (Cited on page 11.)

[Tre14] Trezor Wiki,Cryptocurrency standards,Hierachical deterministic wallets. https://wiki.trezor.io/Cryptocurrency_standards, 2014. (Cited on page 2.)

[Wik18] Bitcoin Wiki. BIP32 proposal. https://en.bitcoin.it/wiki/BIP_0032, 2018. (Cited on page 1.)

[YLY+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. https://eprint.iacr.org/2022/627. (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 ϵ, we have $\mathsf{RandPK}(\mathsf{pk}, \epsilon) = \mathsf{pk}$ and $\mathsf{RandSK}(\mathsf{sk}, \epsilon) = \mathsf{sk}$.

We further require:

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

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

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

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

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 sk, but it can also return signatures that were produced with keys that represent *honest* rerandomizations of 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 (ρ, m). The notion of **uf-cma-hrk1** for a rerandomizable signature scheme RSig is formally modeled in the form of a game $\textbf{uf-cma-hrk1}_{\mathsf{RSig}}$ 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 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 $\textbf{uf-cma-hrk1}_{\mathsf{RSig}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk1}_{\mathsf{RSig}}} := \Pr[\textbf{uf-cma-hrk1}^{\mathcal{A}}_{\mathsf{RSig}} = 1]$.

Game $\textbf{uf-cma-hrk1}_{\mathsf{RSig}}$:

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

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 $\mathsf{R} := \{(g, h, X, Y), x \mid X = g^x, Y = h^x\}$ 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 4 and the TVRF construction, which we denote by TVRF, in Figure 5. The threshold for the (t, n)-TVRF scheme is set to $t \leq \frac{n-1}{2}$.

$\mathsf{DLEq.Prove}(g^x, h^x, x)$
00 Sample $r \xleftarrow{\$} \mathbb{Z}_q$.
01 Compute $c \leftarrow \mathsf{H}(g^x, h^x, g^r, h^r)$
02 Compute $s = r + c \cdot x$.
03 Return $\pi := (c, s)$.

$\mathsf{DLEq.Verify}(g^x, h^x, \pi)$
00 Parse $\pi := (c, s)$.
01 $R \leftarrow g^s/(g^x)^c$.
02 $R' \leftarrow h^s/(h^x)^c$.
03 If $c \neq \mathsf{H}_1(g^x, h^x, R, R')$: Return 0.
04 Return 1

Figure 4: NIZK proof of equality of discrete logarithms with $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$.

$\mathsf{TVRF.Gen}(1^\kappa, t, n)$
00 Sample $a_i \xleftarrow{\$} \mathbb{Z}_q$ for $i \in [t] \cup \{0\}$
01 Let $F(x) := a_t x^t + \cdots + a_1 x + a_0$
02 $\mathsf{sk} := x \leftarrow a_0 \mod q$, $X \leftarrow g^x$
03 $\mathsf{sk}_i := x_i \leftarrow F(i) \mod q$, $X_i \leftarrow g^{x_i}$
04 $\mathsf{pk} := (X, \{X_1, \cdots, X_n\})$
05 Return $(\mathsf{pk}, \{\mathsf{sk}_1, \cdots, \mathsf{sk}_n\})$

$\mathsf{TVRF.Combine}(\mathsf{pk}, \mathcal{S}, \{(\phi_i, \pi_i)\}_{i \in \mathcal{S}})$
00 If $|\mathcal{S}| \leq t$: Return $\perp$.
01 Let $\mathcal{S}' := \emptyset$.
02 Parse $\mathsf{pk} := (X, \{X_1, \cdots, X_n\})$.
03 For $i \in \mathcal{S}$, if $\mathsf{DLEq.Verify}(\phi_i, X_i, \pi_i) = 1$:
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 (ϕ, π).

$\mathsf{TVRF.PEval}(m, \mathsf{sk}_i, \mathsf{pk})$
00 Parse $\mathsf{pk} := (X, \{X_1, \cdots, X_n\})$
01 $\phi_i \leftarrow \mathsf{H}_1(m)^{\mathsf{sk}_i}$.
02 $\pi_i \leftarrow \mathsf{DLEq.Prove}(\phi_i, X_i, \mathsf{sk}_i)$.
03 Return (ϕ_i, π_i).

$\mathsf{TVRF.Verify}(\mathsf{pk}, m, \phi, \pi)$
00 Parse $\mathsf{pk} := (X, \{X_1, \cdots, X_n\})$.
01 Parse $\pi := \{\phi_i, (\pi_i\}_{i \in \mathcal{S}'}$.
02 Let $\mathcal{S}' := \emptyset$.
03 For $i \in \mathcal{S}'$:
04 if $\mathsf{DLEq.Verify}(\phi_i, X_i, \pi_i) \neq 1$
05 return $\perp$.
06 If $\phi = \prod_{i \in \mathcal{S}'} \phi_i^{\lambda_i}$: Return 1.
07 Else return 0.

Figure 5: 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 $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}$.

A.3 ECDSA with Rerandomizable Keys

We briefly recall the standard ECDSA signature scheme in Figure 6 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 $\mathsf{ECDSA}[\mathsf{H}]$, where $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$ is a cryptographic hash function.

In Figure 7, we recall the ECDSA-based signature scheme with rerandomizable keys $\mathsf{rECDSA}[\mathsf{H}]$ as introduced in [DEF+21].

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

Figure 6: ECDSA signature scheme $\mathsf{ECDSA}[\mathsf{H}]$ instantiated with a cryptographic hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$.

$\mathsf{Sign}\,(\mathsf{sk}, m)$	$\mathsf{RandSK}\,(\mathsf{sk}, \rho)$
00 $m' \leftarrow (\mathsf{pk}, m)$	00 $\mathsf{sk}' \leftarrow (\mathsf{sk} + \rho) \mod q$
01 $\sigma \leftarrow \mathsf{ECDSA}[\mathsf{H}].\mathsf{Sign}\,(\mathsf{sk}, m')$	01 Return sk'
02 Return σ	
	$\mathsf{RandPK}\,(\mathsf{pk}, \rho)$
$\mathsf{Verify}\,(\mathsf{pk}, \sigma, m)$	02 $\mathsf{pk}' \leftarrow (\mathsf{pk} + g^\rho)$
03 $m' \leftarrow (\mathsf{pk}, m)$	03 Return pk'
04 Return $\mathsf{ECDSA}[\mathsf{H}].\mathsf{Verify}\,(\mathsf{pk}, m', \sigma)$	

Figure 7: Public key prefixed version of the ECDSA signature scheme with perfectly rerandomizable keys $\mathsf{rECDSA}[\mathsf{H}]$ based on the ECDSA signature scheme $\mathsf{ECDSA}[\mathsf{H}]$. Above $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_q$ 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 $\mathsf{ZK} := (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$, where (i) $\mathsf{Setup}(1^\kappa)$ outputs a common reference string crs; (ii) $\mathsf{Prove}(\mathsf{crs}, (Y, y))$ outputs a proof π for $(Y, y) \in \mathsf{R}$; (iii) $\mathsf{Verify}(\mathsf{crs}, Y, \pi)$ 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 $(Y, y) \in \mathsf{R}$, all $\kappa \in \mathbb{N}$ and $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$, it holds that $\mathsf{Verify}(\mathsf{crs}, Y, \mathsf{Prove}(\mathsf{crs}, (Y, y))) = 1$ except with negligible probability;
2. *Soundness*: For any $(Y, y) \notin \mathsf{R}$, all $\kappa \in \mathbb{N}$ and $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$, it holds that $\mathsf{Verify}(\mathsf{crs}, Y, \mathsf{Prove}(\mathsf{crs}, (Y, y))) = 0$ except with negligible probability;
3. *Zero knowledge*: For any PPT adversary $\mathcal{A}$, there exist a PPT algorithm $\pi_{\mathcal{S}} \leftarrow \mathcal{S}(\mathsf{crs}, Y)$ such that for all $\kappa \in \mathbb{N}$, all $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$ and all $(Y, y) \in \mathsf{R}$, the distributions $\{(\pi, Y) : \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, Y, y)\}$ and $\{(\pi_{\mathcal{S}}, Y) : \pi_{\mathcal{S}} \leftarrow \mathcal{S}(\mathsf{crs}, Y)\}$ 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 $\mathsf{CT} := (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Equivocate})$ where Gen gets as input the security parameter $\kappa \in \mathbb{N}$ and outputs public parameters par and a trapdoor τ; Com takes as input par and a message $m \in \{0,1\}^*$ and outputs a tuple (c, d); Open takes as input par and a tuple $(c, d) \in (\mathcal{C} \times \mathcal{O})$ and either outputs a message m or $\perp$; $\mathsf{Equivocate}$ takes as input a trapdoor τ, 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 $(\mathsf{par}, \tau) \leftarrow \mathsf{Gen}(1^\kappa)$, any two messages $m, m' \in \{0,1\}^*$ and $(c, d) \leftarrow \mathsf{Com}(\mathsf{par}, m)$ and $(c', d') \leftarrow \mathsf{Com}(\mathsf{par}, m')$, 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 $(\mathsf{par}, \tau) \leftarrow \mathsf{Gen}(1^\kappa)$, there exists no PPT adversary $\mathcal{A}$ which can output (c, d, d') such that $\mathsf{Open}(\mathsf{par}, c, d) \neq \mathsf{Open}(\mathsf{par}, c, d')$ and $\mathsf{Open}(\mathsf{par}, c, d) \neq \perp$ and $\mathsf{Open}(\mathsf{par}, c, d') \neq \perp$ except with negligible probability.

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

Finally, a commitment scheme is *non-malleable* if for all $\kappa \in \mathbb{N}$, all $(\mathsf{par}, \tau) \leftarrow \mathsf{Gen}(1^\kappa)$, any message $m \in \{0,1\}^*$ and $(c, d) \leftarrow \mathsf{Com}(\mathsf{par}, m)$, 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 $m' \leftarrow \mathsf{Open}(\mathsf{par}, c', d')$ the messages m and m' are related.

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

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

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 ν in the security parameter $\kappa \in \mathbb{N}$ s.t.:

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

B.2 Construction

The $\mathsf{GG}[\mathsf{H}_0]$ 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. $x = a_i \cdot b_j \mod q$ to transform a_i and b_j into additive shares of x, i.e., into shares α_i and β_j s.t. $x = \alpha_i + \beta_j$. We briefly recall this protocol here. We denote by 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 $(\mathsf{pk}_{i,\mathsf{PKE}}, \mathsf{sk}_{i,\mathsf{PKE}})$ the public/secret key pair of scheme PKE of P_i.

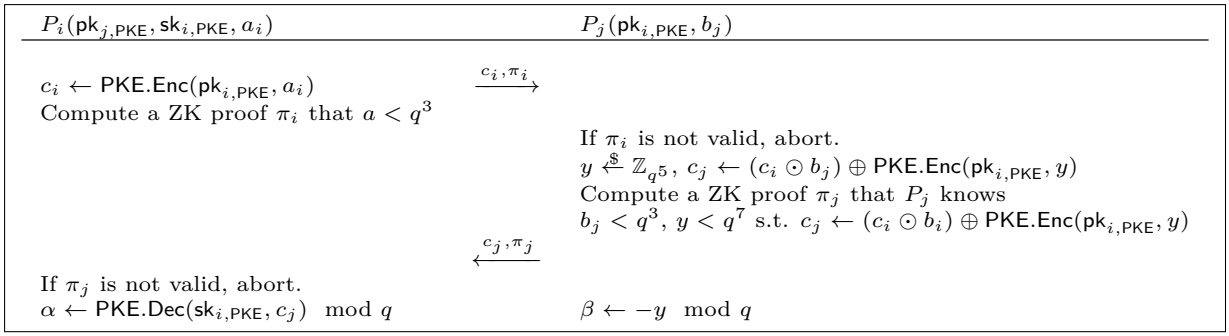

Figure 8: Multiplicative to additive share conversion protocol MtA.

Gennaro and Goldfeder also consider a slight adjustment of the above protocol which they call 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 $\mathsf{GG}[\mathsf{H}_0]$ scheme. For simplicity, we slightly deviate from the original $\mathsf{GG}[\mathsf{H}_0]$ scheme in two ways. We emphasize that these two changes have 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 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.

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

Figure 9: 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.

In Figure 10 we recall the signing procedure of the $\mathsf{GG}[\mathsf{H}_0]$ scheme. The procedure makes use of a non-malleable and equivocable commitment scheme $\mathsf{CT} := (\mathsf{Com}, \mathsf{Open})$ as well as a hash function $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{Z}_q$, a linearly homomorphic encryption scheme $\mathsf{PKE} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and a non-interactive zero-knowledge proof system 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 for the PKE scheme before Phase 1 of the signing procedure. The parties then broadcast the public key together with a zero-knowledge proof that the key was generated honestly. In more detail, before Phase 1 of the signing procedure, each party computes $(\mathsf{pk}_{i,\mathsf{PKE}}, \mathsf{sk}_{i,\mathsf{PKE}}) \leftarrow \mathsf{PKE.Gen}(1^\kappa)$ and $\pi_{i,\mathsf{PKE}} = \mathsf{ZK}_{\mathsf{PKE}_i}\{(\mathsf{pk}_{i,\mathsf{PKE}}, \mathsf{sk}_{i,\mathsf{PKE}}) : (\mathsf{pk}_{i,\mathsf{PKE}}, \cdot) \in \mathsf{PKE.Gen}(1^\kappa)\}$. Finally, each party broadcasts $(\mathsf{pk}_{i,\mathsf{PKE}}, \pi_{i,\mathsf{PKE}})$. The parties then engage in the signing procedure as specified in Figure 10.

Finally, in Figure 11 we recall the simulation of the signing procedure as provided 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.

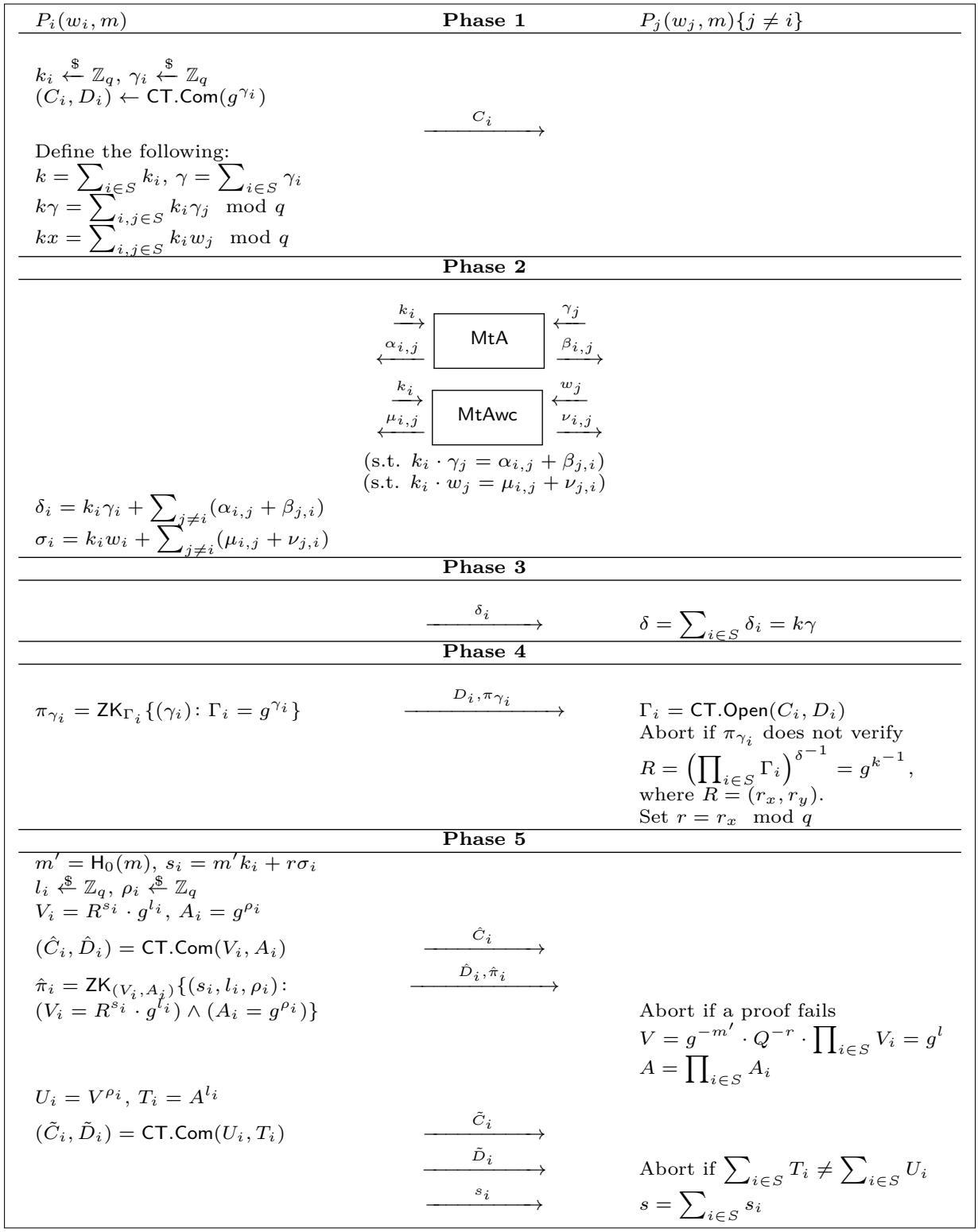

Figure 10: Interactive (t, n)-threshold ECDSA scheme by Gennaro and Goldfeder [GG18], where $|S| \subseteq [n], |S| = t + 1$. For all parties $\{P_i\}_{\{i \in [n]\}}$, x_i denotes secret share of the secret x. For all parties $P_{i\{i \in S\}}$, w_i represents the secret share of x due to $(t, t + 1)$-secret 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 $\mathsf{pk}_{1,\mathsf{PKE}}$ s.t. $(\mathsf{pk}_{1,\mathsf{PKE}}, \cdot) \in \mathsf{PKE.Gen}(1^\kappa)$ and simulates the zero-knowledge proof $\pi_{1,\mathsf{PKE}}$.

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^{γ_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 γ_1 and extracts the following values from the zero-knowledge proofs that are exchanged during the MtA protocol: k_i, γ_i, y_1 for $i > 1$. It then computes $\alpha_{1,j} = k_1\gamma_i + y_1 \mod q$ and $\tilde{k} = \sum_{i \in S} k_i \mod q$.

For the execution of the MtAwc 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 Γ_i. $\mathcal{F}$ extracts γ_j for all j_1 from the zero-knowledge proofs π_{γ_j} and computes $k = \delta \cdot \left(\sum_{i>0} \gamma_i\right)^{-1} \mod q$.

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 $R = g^{\mathsf{H}(m)s^{-1}} \cdot X^{rs^{-1}}$.

(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 11: 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.

B.3 Proof Sketch of Theorem 4.2

B.3.1 Case $b = 0$

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

The simulation of the oracles `Rand`, `RSign` and the random oracle H_0 happens in a straightforward way, i.e., $\mathcal{B}_0$ simply forwards queries from $\mathcal{A}$ in game $\textbf{unf-prand}_{\mathsf{TVRF\text{-}rGG}}$ to the corresponding oracle in game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}}$.

The simulation of the random oracle H_1 and the `REval` oracle is a bit more challenging as $\mathcal{B}_0$ does not have access to any such oracle in game $\textbf{th-ufcma-hrk1}_{\mathsf{rGG}}$. Upon a query from $\mathcal{A}$ to the random oracle H_1 on input a message m, $\mathcal{B}_0$ first checks if $\mathsf{H}_1(m)$ has been set already. If so, it simply returns $\mathsf{H}_1(m)$. Otherwise it samples a uniformly random value $r \xleftarrow{\$} \mathbb{Z}_q$ and sets $\mathsf{H}_1(m) := g^r$ and returns g^r. The simulation of the `REval` oracle then works as follows: On input a message m, an index $i \in [n]$ and a randomness $\rho \in \mathsf{RList}$, $\mathcal{B}_0$ first executes $\mathsf{pk}' \leftarrow \mathsf{TVRF\text{-}rGG.RandPK}(\mathsf{pk}, \rho)$ and parses $\mathsf{pk}' := (X', \{X'_1, \cdots, X'_n\})$. $\mathcal{B}_0$ then retrieves $r \leftarrow \mathsf{H}_1(m)$, sets $\phi_i := (X'_i)^r = \mathsf{H}_1(m)^{\mathsf{sk}'_i}$, simulates the corresponding zero-knowledge proof π_i and returns (ϕ_i, π_i).

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

B.3.2 Case $b = 1$

In this case, $\mathcal{B}$ executes $\mathcal{B}_1$ which plays in game $\textbf{th-prand}_{\mathsf{TVRF}}$. That is, upon $\mathcal{A}$ sending the list C of parties to corrupt in game $\textbf{unf-prand}_{\mathsf{TVRF\text{-}rGG}}$, $\mathcal{B}_1$ corrupts the same parties in $\textbf{th-prand}_{\mathsf{TVRF}}$ and forwards the resulting secret key shares and the public key to $\mathcal{A}$. The simulation of oracles `Rand`, `RSign`, H_0, 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 $\rho \xleftarrow{\$} \mathbb{Z}_q$, stores ρ in RList and returns ρ.

- **Oracle H_0:** Upon $\mathcal{A}$ querying 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 $m := (\mathsf{pk}', m')$ where $\mathsf{pk}' \leftarrow \mathsf{TVRF\text{-}rGG.RandPK}(\mathsf{pk}, \rho)$ for some $\rho \in \mathsf{RList}$. If so and if $\mathsf{H}_0(m)$ has already been set, then $\mathcal{B}_1$ aborts. Else, $\mathcal{B}_1$ executes $\mathcal{S}_{\mathsf{ECDSA}}$ as described in Figure 3 on input (X', m) with $\mathsf{pk}' := (X', \{X'_1, \cdots, X'_n\})$.

  If m is not public key prefixed, $\mathcal{B}_1$ simply samples a uniformly random value $r \xleftarrow{\$} \mathbb{Z}_q$, 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 $\mathcal{S}_{\mathsf{ECDSA}}$ programs the random oracle 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 $\rho \in \mathsf{RList}$, $\mathcal{B}_1$ simulates the signing procedure in the same way as described in Theorem 3.4. 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 H_0. Note that the simulator code from Theorem 3.4 does not program H_0 such that there is no conflict between the execution of the simulator code from Theorem 3.4 and $\mathcal{S}_{\mathsf{ECDSA}}$.

- **Oracle H_1:** Upon $\mathcal{A}$ querying 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, ρ), $\mathcal{B}_1$ queries its own oracle on input m and receives an evaluation share (ϕ_i, π_i) where $\phi_i = \mathsf{H}_1(m)^{\mathsf{sk}_i}$. $\mathcal{B}_1$ then computes $\phi'_i = \phi_i \cdot H_1(m)^{\rho_i} = H_1(m)^{\mathsf{sk}_i + \rho_i}$ (where ρ_i is the randomness share of ρ for party P_i according to the $\mathsf{TVRF\text{-}rGG.RandSK}$ algorithm), simulates a NIZK proof π'_i of the DLEq proof system (cf. Appendix A.2) and sends (ϕ'_i, π'_i) to $\mathcal{A}$.

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