The Exact Security of BIP32 Wallets

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

[1] TU Darmstadt, Germany
firstname.lastname@tu-darmstadt.de
[2] University of Maryland, USA
lossjulian@gmail.com

Abstract

In many cryptocurrencies, the problem of key management has become one of the most fundamental security challenges. Typically, keys are kept in designated schemes called *wallets*, whose main purpose is to store these keys securely. One such system is the BIP32 wallet (Bitcoin Improvement Proposal 32), which since its introduction in 2012 has been adopted by countless Bitcoin users and is one of the most frequently used wallet system today. Surprisingly, very little is known about the concrete security properties offered by this system. In this work, we propose the first formal analysis of the BIP32 system in its entirety and without any modification. Building on the recent work of Das et al. (CCS '19), we put forth a formal model for hierarchical deterministic wallet systems (such as BIP32) and give a security reduction in this model from the existential unforgeability of the ECDSA signature algorithm that is used in BIP32. We conclude by giving concrete security parameter estimates achieved by the BIP32 standard, and show that by moving to an alternative key derivation method we can achieve a tighter reduction offering an additional 20 bits of security (111 vs. 91 bits of security) at no additional costs.

Keywords: Wallets, cryptocurrencies, foundations, BIP32

1 Introduction

Decentralized cryptocurrencies such as Bitcoin or Ethereum have introduced a new digital payment paradigm which does not rely on a central authority such as a bank or a credit card company. The main building block used in many popular cryptocurrencies to facilitate secure transfer and holding of assets are digital signatures. Loosely speaking, a user Alice in the system is identified by her public key pk_A which she uses as her address for receiving and sending payments. If Alice wants to send c coins of the underlying currency to another user Bob with address pk_B, she creates a transaction tx saying "Send c coins from pk_A to pk_B" and signs tx using her secret key sk_A. She then uploads the transaction tx together with the signature σ to the public ledger (often also called blockchain) of the cryptocurrency. Once the tuple (tx, σ) is visible on the public ledger, the payment is completed meaning that now Bob owns an additional c coins of the underlying currency. Clearly, Alice's funds remain secure only as long as no one can forge a signature σ on her behalf that verifies under pk_A. On top of this, it is generally recommended to use a fresh signing key for every new transaction stored on the public ledger to avoid that all transactions are linkable to the same user Alice. In the cryptocurrency space, the management and storage of secret keys is typically carried out by so-called *wallets* – which are pivotal for the security of cryptocurrency funds. Indeed, cryptocurrency wallets are a highly attractive target for hackers as illustrated by spectacular attacks against common cryptocurrency projects. For example, in 2018 alone, hackers managed to steal more than one billion USD worth of cryptocurrency from wallets [Ske18, Blo18, Bit18].

While several recent works study the formal security properties of cryptocurrency wallets (see related work for a detailed discussion), one of the most widely used schemes – *the BIP32 wallet* [Wik18] – has not been formally analyzed so far. This is somewhat surprising as BIP32 became a standard for deterministic Bitcoin wallets in 2012, and has been widely adopted since then (e.g., it is used in the deployment of popular wallets [Ele13, Tre14, Led14]). In this work, we address this gap and provide the first comprehensive study of the security properties achieved by the BIP32 wallet standard.

1.1 Deterministic Wallets

As we have already pointed out, to improve privacy it is important to not re-use the same signing key for too many public transactions. To explain why privacy is also beneficial for security, let us consider a user Alice who holds a single secret/public key pair $(\mathsf{sk}_A, \mathsf{pk}_A)$, and that she receives multiple payments to her address pk_A. As we have explained, such transactions contain her public key pk_A and are posted to the public ledger. Hence, an attacker can easily extract Alice's balance via the public transaction ledger. Over time, pk_A's balance might grow, and at some point, the attacker may identify pk_A as a high-priority target. The obvious approach to thwart an attacker's attempts of linking Alice's transactions would be for Alice to keep a set of l one-time random key pairs $\{(\mathsf{sk}_1, \mathsf{pk}_1), \ldots, (\mathsf{sk}_l, \mathsf{pk}_l)\}$ within her wallet, where each key pair is used for a single transaction on the public ledger. However, this approach has the obvious downside of Alice having to store all of her keys on disk (as long as they still retain some amount of currency). This requires a lot of storage space and bears the risk of losing one of her keys, at which point the associated funds of that key are irrevocably lost. A simple approach to overcome these issues are *deterministic wallets*, proposed by Buterin [But13]. A deterministic wallet usually contains a pair of master keys $(\mathsf{msk}, \mathsf{mpk})$ and a seed ch, which is also referred to as the *chaincode*. For every new transaction, the wallet deterministically derives a fresh session key pair $(\mathsf{sk}, \mathsf{pk})$ from the master keys with the help of deterministic key derivation algorithms. More precisely, the public key derivation algorithm takes as input the master public key mpk, the chaincode ch and an identifier ID and deterministically computes a one-time public key $\mathsf{pk}_{\mathsf{ID}}$. An analogous secret key derivation algorithm takes in msk, ch, and ID and deterministically computes a one-time secret key $\mathsf{sk}_{\mathsf{ID}}$ that matches $\mathsf{pk}_{\mathsf{ID}}$ (given that the arguments ch and ID in both derivations are identical). Going back to the example of BIP32, $(\mathsf{msk}, \mathsf{mpk})$ are generated as ECDSA keys and public key derivation is done by computing the offset $\omega := \mathsf{H}(\mathsf{mpk}, \mathsf{ch}, \mathsf{ID})$, where $\mathsf{ID} \in [2^{32}]$ and then rerandomizing mpk to $\mathsf{pk}_{\mathsf{ID}}$ by computing $\mathsf{pk}_{\mathsf{ID}} := \mathsf{mpk} + G \cdot \omega$. Here, G denotes the base point of an elliptic curve group of prime order p. A matching secret key can be derived via $\mathsf{sk}_{\mathsf{ID}} := \mathsf{msk} + \omega \pmod{p}$.

Hot/Cold Wallets. A typical way of using deterministic wallets in practice is via the hot/cold wallet paradigm. With this approach, Alice maintains two wallets. The first wallet is referred to as the *cold wallet*. It keeps the master secret key msk as well as the chaincode. The cold wallet is usually implemented via some simple storage device that should be almost permanently disconnected from the internet, so as to minimize the risk of attack. The second wallet is the so-called *hot wallet*, which is permanently online and keeps the master public key as well as the chaincode. Using the deterministic key derivation procedures, the two wallets can independently derive matching keys to use for one-time transaction on the public ledger. In a bit more detail, Alice uses her hot wallet as a low-security spending wallet which, at any point in time, keeps only a small amount of currency. Whenever the funds stored on the hot wallet exceed a certain amount, Alice can use the public key derivation algorithm to derive a new public key $\mathsf{pk}_{\mathsf{ID}}$ on her hot wallet and transfer the excess funds to $\mathsf{pk}_{\mathsf{ID}}$. Note that this requires no interaction with the cold wallet. At a later point in time, the cold wallet can come online for a brief moment and spend the funds from $\mathsf{pk}_{\mathsf{ID}}$, using a matching secret key $\mathsf{sk}_{\mathsf{ID}}$ derived via the secret key derivation algorithm. As far as security goes, we would like to ensure two properties. First, *unlinkability* ensures that keys derived from the same master key pair are indistinguishable from random keys, given that the hot wallet has not leaked the chaincode to the attacker. Second, *unforgeability* ensures that even if the hot wallet leaks the chaincode (e.g., because it has been corrupted), signatures from derived keys should still remain unforgeable. While unlinkability is easy to achieve, unforgeability is a much more subtle issue in this setting, as the derived keys are all correlated once the chaincode has been revealed to the attacker. Hence, the standard unforgeability property of the underlying signature scheme is no longer sufficient to ensure unforgeability of signatures under these derived keys.

1.2 Limitations of Existing Works

The work of Das et al. [DFL19] was the first to provide a formal model to reason about the aforementioned security properties. It also showed how to achieve secure constructions in their proposed model from various different signature schemes used in practice, e.g., Schnorr, BLS, and ECDSA. Notably, the latter construction is very practical and can be integrated directly with the (unmodified) Bitcoin system. In spite of these achievements, their work makes no progress towards formally proving security properties for the BIP32 wallet standard that is widely used in many real-world systems. Let us discuss the reasons

for this in a little more detail.

First, the construction of Das et al. uses a *multiplicative rerandomization* to derive keys, in which keys for identity ID are computed from $\omega = \mathsf{H}(\mathsf{mpk}, \mathsf{ch}, \mathsf{ID})$ as $\mathsf{pk}_{\mathsf{ID}} := \mathsf{mpk} \cdot \omega$, and $\mathsf{sk}_{\mathsf{ID}} := \mathsf{msk} \cdot \omega \pmod p$. By comparison, as we saw above, BIP32 uses an *additive rerandomization*. Although this might look like a minor difference, we will see later that the proof technique and security guarantees achieved by the additive version differ significantly from the multiplicative one. Second, the work of Das et al. does not consider the hierarchical key derivation mechanism provided by BIP32. Hierarchical deterministic wallets allow for keys in the wallet to act simultaneously as signing keys *and* as parent (master) keys to derive new child keys in their own right. As a useful example, consider a company that wishes to delegate new signing key pairs to different entities within the company. Unfortunately, it cannot be guaranteed that all entities in the company store their keys securely and some of them might be leaked to the adversary over time. Such a strong adversary cannot be captured by the model and constructions of Das et. al. Since many wallets that are used in practice follow the BIP32 standard, it is crucial to provide a formal analysis of the scheme *as is*, meaning without any modifications to it.

1.3 Our Contributions

In this work we address the above shortcomings and provide, for the first time, a formal analysis of the full BIP32 specification in the hot/cold wallet setting. An important implication of our work is that we can establish the exact security that is achieved by the current standard, which also leads us to propose a minor modification that can significantly improve security without any additional costs.

Rerandomizing ECDSA. We begin by recalling the notion of *unforgeability under honestly rerandomized keys (UFCMA-HRK)* introduced by Das et al. [DFL19]. As this notion will serve as the basis of our wallet constructions, we review it in detail below. Compared to the standard notion of unforgeability under chosen message attacks (UFCMA), the adversary in the UFCMA-HRK game initially obtains a challenge public key pk and gets to query for rerandomizations of pk. The game returns the rerandomized public key $\tilde{\mathsf{pk}}$ together with the (uniformly chosen) randomness ρ that was used in the rerandomization process. The exact way that the rerandomization is actually done depends on the scheme; we are mostly interested in the case where ECDSA keys are additively rerandomized as $\mathsf{pk} + G \cdot \rho$. The game then allows the adversary to query for signatures relative to any of the rerandomized public keys that it has previously obtained from the game. It is considered successful if it can return a forgery relative to any of the requested keys $\tilde{pk}$ on a message for which it has not previously asked for a signature under $\tilde{pk}$. As observed by Das et al., this security notion is a weakened version of *unforgeability under rerandomized keys* [FKM+16] in which the adversary can choose the random coins ρ itself and provide them to the game. In Section 3, we prove that ECDSA with additive rerandomization satisfies UFCMA-HRK as long as *each message is signed only once per key*. A first attempt is to naively follow the approach of Das et al. who showed that ECDSA with multiplicative rerandomization satisfies UFCMA-HRK (without any restrictions on the number of signatures per message). The main idea of Das et al.'s reduction from UFCMA-HRK to UFCMA (both with respect to the ECDSA scheme) is to rely on a *related key attack* (RKA) that is present in the multiplicatively rerandomized version of the ECDSA scheme. Concretely, the RKA allows to transform a signature (r, s) on message m_0 relative to a key pk_0 into a signature $(r, s/\rho)$ on message m_1 that is valid under the related key $\mathsf{pk}_1 = \mathsf{pk}_0 \cdot \rho$, where ρ satisfies $\rho = \frac{\mathsf{H}(m_0)}{\mathsf{H}(m_1)}$. This attack can be leveraged by the reduction to answer all signing queries in the UFCMA-HRK game. More precisely, using the RKA, it is possible to transform signatures obtained from the signing oracle in the UFCMA game into signatures relative to any of the rerandomized keys in the UFCMA-HRK game (via programming of the random oracle). Hence, we are immediately faced with the following obstacle: this RKA does not work if keys are additively rerandomized.

Extending to Additive Rerandomization. To overcome this issue with the existing reduction, we present a new RKA which works for additively rerandomized ECDSA. The attack works as follows: given a signature (r, s) on m_0 relative to pk_0, (r, s) is also a valid signature relative to the public key $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$ on message m_1, given that $\rho = (\mathsf{H}(m_0) - \mathsf{H}(m_1))/r$. Rather surprisingly, considering ECDSA's huge popularity, we are not aware of this attack having been noticed previously. Using our new RKA, we are now able to (almost) make the simulation of signatures in Das et al.'s approach work. However, there is a further issue that comes from the structure of the additive RKA. Suppose that the reduction is directed to program the random oracle H on a message m so as to provide the attacker with a signature relative to a (rerandomized) public key $\tilde{\mathsf{pk}}$ in the UFCMA-HRK game. The above RKA

forces the reduction to program H on a value that depends on a *particular signature* (r, s) on m, which it obtains from the signing oracle in the underlying UFCMA game. Now, the only signature on m that the reduction can hand to the adversary under pk is (r, s). If the adversary requests another signature *on the same message* m, we are not able to reply with a fresh signature, as we can program H on m only a single time. For this reason, we have to restrict ourselves to one-per-message unforgeability. We emphasize, however, that this notion of security (one signature per-message) is sufficient in our setting, as transactions are identified by unique nonces in most cryptocurrencies (including Bitcoin) and hence never signed twice. An additional benefit of our new reduction (compared to [DFL19]) is that it only requires the weaker assumption that the underlying ECDSA scheme is one signature per-message unforgeable in its own right. This is worth noting, as the work of Fersch et al. shows that ECDSA achieves this property in the random oracle model [FKP17] (albeit with a very large security loss). By comparison, the unrestricted security (i.e., UFCMA) of ECDSA remains only a conjecture in the plain random oracle model. Our reduction also removes the need for the random salt present in Das et al.'s construction. This is an important improvement, as it allows using BIP32 without Bitcoin's scripting language, which was required by the construction of Das et al. due to their use of the salt. Finally, we remark that our reduction (by comparison to Das et al.) is *non-tight* and loses a factor proportional to the total number of keys derived in the UFCMA-HRK game. We provide further discussion on this issue in the next section and in Section 3.

Hierarchical Wallets. To complete the analysis of BIP32, the second part of our work focuses on formal security properties when supporting hierarchies in deterministic wallet constructions (as is the case for BIP32). As already hinted, the core difficulty in this setting is that some of the wallet's keys may be given to untrustworthy users who may leak their cold wallet keys to the adversary. If this happens, it is important to ensure that the adversary does not gain information about secret keys further up in the hierarchy. It is easy to see that this property is not achieved if all keys are derived using the derivation algorithms described so far: if the adversary learns $\mathsf{sk}_{\mathsf{ID}} = \mathsf{msk} + \rho \pmod p$, where ρ is computed as $\rho = \mathsf{H}(\mathsf{mpk}, \mathsf{ch}, \mathsf{ID})$, then it can recover msk as $\mathsf{msk} = \mathsf{sk}_{\mathsf{ID}} - \rho \pmod p$ and learn all cold wallet keys that were ever derived using msk. Because of this, BIP32 offers a second mode of deriving keys called *hardened key derivation*. Hardened keys are derived by changing the computation of the offset ρ above to $\rho = \mathsf{H}(\mathsf{msk}, \mathsf{ch}, \mathsf{ID})$. Now, even when learning $\mathsf{sk}_{\mathsf{ID}}$, it is not possible for the adversary to recover msk. The downside of hardened key derivation is that the hot and cold wallet can no longer independently derive keys (as the hot wallet does not know msk). Thus, this mode of derivation is not intended for use in the hot/cold wallet paradigm, but simply to create keys with a higher degree of security. These keys can either be stored (efficiently) as part of the main wallet or handed to users in the system without any concern for other cold wallet keys. In Section 4, we state the syntactical definition and correctness properties of a hierarchical deterministic wallet. We then introduce a security model that supports both types of key derivations (hardened and non-hardened), as well as secret key leakage of hardened keys. We refer to this notion of security as WUFCMA. In Section 5, we provide a generic construction HDWal that transforms a signature scheme satisfying UFCMA-HRK into a hierarchical deterministic wallet with WUFCMA security.[1] In this way, we are able to complete the analysis of BIP32 by instantiating HDWal with ECDSA using additive rerandomization.

On the Tightness of Our Construction. A particular focus of our work is to analyze the tightness and concrete security achieved by our constructions, most notably BIP32. We have already mentioned that our reduction from UFCMA-HRK to UFCMA of the ECDSA scheme with additive rerandomization is non-tight. More precisely, it loses a factor proportional to the number of keys derived by the adversary in the UFCMA-HRK game. Thus, our goal is to at least achieve the best possible tightness of our generic transform HDWal. To this end, let us first consider the possible options for potential security losses. From worst to best (excluding a tight reduction), the options are:

- Loss in the number of random oracle queries.
- Loss in the number of keys derived in the wallet (hardened or non-hardened).
- Loss in the number of signing oracle queries (assuming keys are used only once).
- Loss in the number of hardened keys leaked to the adversary.

[1] In case the underlying signature scheme has the one signature per message restriction, then the resulting wallet scheme also does.

The first three possibilities are quite catastrophic as the number of random oracle queries, signing oracle queries, or keys derived in practice could be quite high. On the other hand, we expect the number of *leaked keys* to be only a small portion of all the keys in a given wallet (we use 1% as an estimate in our calculations). We are able to prove that HDWal indeed achieves a multiplicative security loss proportional to only the hardened keys leaked to the adversary over the course of the lifetime of the wallet. Furthermore, we show that any *generic transform* from UFCMA-RK (a stronger notion than what is used in our construction) to WUFCMA *must lose at least this factor*. Hence, our construction HDWal achieves the *best possible parameters*. To prove our results, we adapt the reduction/metareduction techniques introduced by Coron in his seminal work [Cor02]. Given that his results deal with the tightness of unique signatures (which is very different from our setting), this requires careful insight into his technique in order to adapt it to our model.

Concrete Security Parameters. We conclude by giving a discussion of the concrete security levels achieved by BIP32 and the multiplicative ECDSA scheme of Das et al., when plugged into HDWal. We find that BIP32 gives roughly 94 bits of security according to our theorems and conservative choices of parameters. We find that by comparison, the multiplicative version of Das et al. gives 114 bits of security with a similarly efficient scheme. (We remark that using the techniques introduced in our paper, we can also remove the salt in the multiplicatively rerandomizable ECDSA version of Das et al.). Given these insights, we strongly recommend that the Bitcoin community switch rerandomizations in BIP32 from additive to multiplicative, in particular since these changes essentially come for free.

1.4 Related Work

The most relevant previous work for us is by Das et al. [DFL19] as mentioned previously. However, there have been other works which try to formalize cryptographic wallets. The work of Gutoski and Stebila [GS15] proposes an alternative construction for hierarchical wallets where up to d session keys can leak without the master secret key being compromised under the one-more discrete-log assumption. However, their security model is weaker than our model (or the security model of Das et al. on which we base our work). More precisely, in their model, the adversary cannot query the game for signatures under uncompromised wallet keys. Furthermore, instead of the traditional security model where the adversary wins if she can forge a signature, the adversary's goal in their security definition is to extract the master secret/public key pair. Another more recent work is by Luzio et al. [LFA20] where the authors design a new hierarchical wallet scheme by using (deterministic) hierarchical key assignment schemes [ABFF09]. Unfortunately, their solution is not compatible with cryptocurrencies such as Bitcoin since their solution requires a more sophisticated (signature) verification algorithm, where a certificate associated with the user needs to be verified along with the signature.

Turuani et al. [TVR16] analyzed the Bitcoin Electrum wallet using automated verification in the Dolev-Yao model. However, many automated verification models only consider "idealized" building blocks, i.e., cryptographic building blocks that are perfectly secure. Consequently, this type of analysis excludes weaknesses such as related key attacks, which are of fundamental relevance in the setting of deterministic wallets.

Another line of work has considered the security of hardware wallets [MPs19, AGKK19] and implementation bugs in wallets (such as weak randomness) [CEV14, BR18, BH19]. Additionally, there have been several works with focus on the use of threshold ECDSA signatures [KMOS19, GGN16, LN18, DKLs18] and multi-signatures [BDN18] in (and outside of) wallet systems.

In a recent work, Alkadri et al. [ADE+20] have shown how to realize deterministic wallets that are post-quantum secure. To this end, they suitably adapt the model and techniques of Das et al. by considering an adversary with quantum computing power.

The concept of rerandomizable signature schemes was first introduced by Fleischhacker et al. [FKM+16] and later used by [DFL19, ADE+20] for their wallet schemes. In addition, related key attacks have been studied for signature schemes such as Schnorr [Sch90] in many previous works [FF13, KMP16, ZCC+15]. For ECDSA, Das et al. leveraged related key attacks to achieve a multiplicatively rerandomizable ECDSA scheme which they prove secure w.r.t. the security notion of unforgeability under honestly rerandomizable keys. Finally, Fersch et. al. [FKP16] provided the first security analysis of ECDSA in an idealized model.

2 Preliminaries

Notation. We use the notation $s \xleftarrow{\$} H$ to denote the uniform sampling of a variable s from the set H. For an integer l, $[l]$ denotes the set of integers $\{1, \cdots, l\}$. We use upper case letters to denote algorithms. For an algorithm A, we write $y \xleftarrow{\$} A(x)$ to denote the execution of a randomized algorithm A on input x that outputs y. We write $y \leftarrow B(x; \rho)$ to denote the execution of an algorithm B that, on input x and randomness ρ, outputs y. Note that in this notation, B *is deterministic.* We use the notation $y \in A(x)$ to denote that y is in the set of possible outputs of A on input x.

In order to simplify our notation and definitions, we assume that public parameters par have been securely generated and can be used throughout the paper as input to algorithms. We generally assume that, initially, boolean variables are set to false, integers are set to 0, lists are set to $\emptyset$, and undefined entries of lists are set to $\perp$. For strings $a, b \in \{0,1\}^*$, we write $a = (b, \cdot)$ if b is a prefix of a and likewise, we write $a \neq (b, \cdot)$ if a is not prefixed by b. We denote by κ the security parameter throughout the paper.

We use standard code-based security games [Sho04]. A *game* $\mathbf{G}$ is an interactive probability experiment between an *adversary* $\mathcal{A}$ and an (implicit) *challenger* which provides answers to oracle queries posed by $\mathcal{A}$. The output of $\mathbf{G}$ when interacting with adversary $\mathcal{A}$ is denoted as $\mathbf{G}^{\mathcal{A}}$. Finally, the randomness in any probability term of the form $\Pr[\mathbf{G}^{\mathcal{A}} = 1]$ is assumed to be over all the random coins in game $\mathbf{G}$.

2.1 Signature Schemes

We now recall the definition of signature schemes and that of signature schemes with perfectly rerandomizable keys from [DFL19].

Definition 2.1 (Signature Scheme). A *signature scheme* is a tuple of algorithms $\mathsf{Sig} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ which are defined as follows:

- $\mathsf{Sig.Gen}(\mathsf{par})$: The randomized *key generation* algorithm $\mathsf{Sig.Gen}$ takes as input public parameters par and outputs a public/secret key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Sig.Sign}(\mathsf{sk}, m)$: The (possibly) randomized *signing* algorithm $\mathsf{Sig.Sign}$ takes as input a secret key sk and a message m and outputs a signature σ.
- $\mathsf{Sig.Verify}(m, \mathsf{pk}, \sigma)$: The deterministic *verification* algorithm $\mathsf{Sig.Verify}$ takes as input a public key pk, a signature σ, and a message m. It outputs either 1 (accept) or 0 (reject).

A signature scheme Sig is *correct* if the following holds: For all $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Sig.Gen}(\mathsf{par})$ and all $m \in \{0,1\}^*$ we have that

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

Definition 2.2 (Signature Scheme with Perfectly Rerandomizable Keys). A *signature scheme with perfectly rerandomizable keys* is a tuple of algorithms $\mathsf{RSig} = (\mathsf{RSig.Gen}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify}, \mathsf{RSig.RandSK}, \mathsf{RSig.RandPK})$ where $(\mathsf{RSig.Gen}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify})$ are the standard algorithms of a signature scheme. Moreover, we assume that the public parameters par define a randomness space $\mathcal{R} := \mathcal{R}(\mathsf{par})$. Then the algorithms $\mathsf{RSig.RandSK}$ and $\mathsf{RSig.RandPK}$ are defined as follows:

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

We make the convention that for the empty string ϵ, we have that $\mathsf{RSig.RandPK}(\mathsf{pk}; \epsilon) = \mathsf{pk}$ and $\mathsf{RSig.RandSK}(\mathsf{sk}; \epsilon) = \mathsf{sk}$.

We further require:

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

$$(\mathsf{sk}', \mathsf{pk}') \leftarrow (\mathsf{RSig.RandSK}(\mathsf{sk}; \rho), \mathsf{RSig.RandPK}(\mathsf{pk}; \rho)),$$
$$(\mathsf{sk}'', \mathsf{pk}'') \xleftarrow{\$} \mathsf{RSig.Gen}(\mathsf{par}).$$

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

$$\Pr_{\sigma \xleftarrow{\$} \mathsf{RSig.Sign}(\mathsf{sk}', m)} [\mathsf{RSig.Verify}\left(\mathsf{pk}', \sigma, m\right) = 1] = 1.$$

Security notion uf-cma1. In this work, we use the security notion of *one-per message existential unforgeability under chosen message attacks* (**uf-cma1**) [FKP17] which is a slightly weaker variant of the standard notion of existential unforgeability under chosen message attacks (**uf-cma**) security. In contrast to standard **uf-cma**, in **uf-cma1**, the adversary is restricted to querying the signing oracle at most once for each message. We formalize the **uf-cma1** notion for a signature scheme Sig in the form of a game $\mathbf{uf\text{-}cma1}_{\mathsf{Sig}}$ as follows.

Game $\mathbf{uf\text{-}cma1}_{\mathsf{Sig}}$:

- **Setup Phase:** The challenger initiates a list as $\mathsf{SigList} \leftarrow \{\epsilon\}$ for storing messages and samples a pair of keys $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Sig.Gen}(\mathsf{par})$. Then, $\mathcal{A}$ is run on input pk.
- **Online Phase:** $\mathcal{A}$ is given access to a signing oracle `Sign` which works as follows. On input a message m, if m was queried in a previous `Sign` query, i.e., if $m \in \mathsf{SigList}$, then $\perp$ is returned. Otherwise, `Sign` computes a signature on message m as $\sigma \xleftarrow{\$} \mathsf{Sig.Sign}(\mathsf{sk}, m)$. The message m is stored in the $\mathsf{SigList}$ and the signature σ is returned as the answer.
- **Output Phase:** Finally, $\mathcal{A}$ wins the game if it can provide a forgery σ^* on a message m^*, where (1) m^* is fresh, i.e., $m^* \notin \mathsf{SigList}$ and (2) σ^* is a valid forgery, i.e., $\mathsf{Sig.Verify}(\mathsf{pk}, \sigma^*, m^*) = 1$.

For an algorithm $\mathcal{A}$ we define $\mathcal{A}$'s advantage in the game $\mathbf{uf\text{-}cma1}_{\mathsf{Sig}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}cma1}_{\mathsf{Sig}}} = \Pr[\mathbf{uf\text{-}cma1}^{\mathcal{A}}_{\mathsf{Sig}} = 1]$.

Security notion uf-cma-hrk1. For signature schemes with perfectly rerandomizable keys, we introduce the notion of *one-per message existential unforgeability under honestly rerandomizable keys* (**uf-cma-hrk1**), which restricts the security notion of *existential unforgeability under honestly rerandomizable keys* (**uf-cma-hrk**) as introduced by Das et al. [DFL19]. In this security notion, 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. The term *honest* indicates that the randomness for the rerandomization is chosen uniformly at random from $\mathcal{R}$ (by the game itself). Our security notion of **uf-cma-hrk1** restricts the notion of **uf-cma-hrk** in the sense that the signing oracle returns at most one signature for each randomness/message pair (ρ, m). We formally model the notion of **uf-cma-hrk1** for a rerandomizable signature scheme RSig in the form of a game $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ as follows.

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

- **Setup Phase:** The challenger initializes two lists as $\mathsf{SigList} \leftarrow \{\epsilon\}$ and $\mathsf{RList} \leftarrow \{\epsilon\}$ and samples a pair of keys $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{RSig.Gen}(\mathsf{par})$. Then $\mathcal{A}$ is run on input pk.
- **Online Phase:**
  - $\mathcal{A}$ is given access to an oracle `Rand`, which, upon a query, samples a fresh random value from $\mathcal{R}$ as $\rho \xleftarrow{\$} \mathcal{R}$, stores ρ in the list RList, and returns ρ.
  - $\mathcal{A}$ is given access to a signing oracle `RSign` which works as follows. 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 return $\perp$. Otherwise, derive a pair of keys rerandomized with the randomness ρ, as $\mathsf{sk}' \leftarrow \mathsf{RSig.SKDer}(\mathsf{sk}; \rho)$ and $\mathsf{pk}' \leftarrow \mathsf{RSig.PKDer}(\mathsf{pk}; \rho)$. If $(\mathsf{pk}', m) \in \mathsf{SigList}$ then return $\perp$. Otherwise, a signature is derived on message m under the secret key sk' as $\sigma \leftarrow \mathsf{RSig.Sign}(\mathsf{sk}', m)$. The tuple (pk', m) is stored in the $\mathsf{SigList}$ and the signature σ is returned as the answer.
- **Output Phase:** $\mathcal{A}$ wins if it returns a forgery σ^* together with a message m^* and a public key $\mathsf{pk}^* \leftarrow \mathsf{RSig.PKDer}(\mathsf{pk}; \rho^*)$,[2] s.t. following holds: (1) the randomness ρ^* has been derived via a `Rand` query, i.e., $\rho^* \in \mathsf{RList}$, (2) (m^*, ρ^*) is fresh, i.e., $(\mathsf{pk}^*, m^*) \notin \mathsf{SigList}$, and (3) σ^* is a valid forgery, i.e., $\mathsf{RSig.Verify}(\mathsf{pk}^*, \sigma^*, m^*) = 1$.

[2] For simplicity, we tacitly assume that pk^* identifies ρ^*. This can easily be achieved using appropriate bookkeeping.

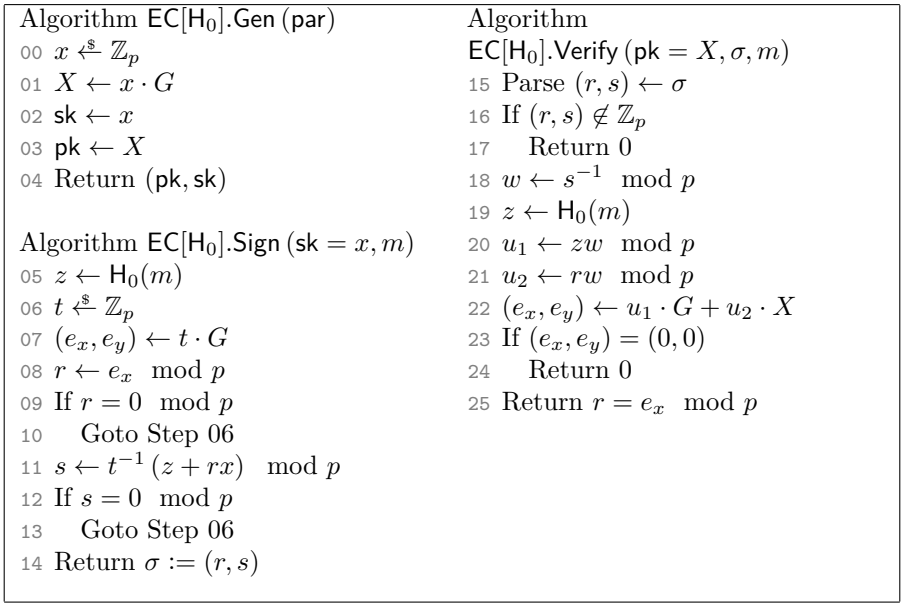

Figure 1: $\mathsf{EC}\,[\mathsf{H}_0] = (\mathsf{EC}[\mathsf{H}_0].\mathsf{Gen}, \mathsf{EC}[\mathsf{H}_0].\mathsf{Sign}, \mathsf{EC}[\mathsf{H}_0].\mathsf{Verify})$: ECDSA signature scheme over to elliptic curve $\mathbb{E}$ using hash function $\mathsf{H}_0 \colon \{0,1\}^* \to \mathbb{Z}_p$.

For an algorithm $\mathcal{A}$ we define $\mathcal{A}$'s advantage in game $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} = \Pr[\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{A}}_{\mathsf{RSig}} = 1]$.

Other than only allowing the adversary to ask for at most one signature per message, our definition deviates from the one presented in [DFL19] by storing the tuples (pk', m) in the list $\mathsf{SigList}$ instead of just storing m. This change allows an adversary in the **uf-cma-hrk1** game to query a signature for the same message but under different public keys.

3 Security Analysis of Additively Rerandomizable ECDSA

In the following discussion, let $\mathbb{E}(\mathsf{par})$ denote an elliptic curve with base point G and prime order p. Furthermore, assume hash functions $\mathsf{H}_0 \colon \{0,1\}^* \to \mathbb{Z}_p$, $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ (modeled as random oracles). In this section, we present a signature scheme with rerandomizable keys $\mathsf{REC}[\mathsf{H}_1]$ based on the standard ECDSA scheme which we denote by $\mathsf{EC}[\mathsf{H}_0]$ (cf. Figure 1). $\mathsf{REC}[\mathsf{H}_1]$, as illustrated in Figure 7, works in a similar way as $\mathsf{EC}[\mathsf{H}_0]$ with two main differences. (1) It is extended by two algorithms RandSK and RandPK for the key rerandomization and (2) it is designed for key-prefixed messages. First, the two algorithms RandSK and RandPK randomize a key pair by *adding* a random value to each key. This is in contrast to the signature scheme with *multiplicatively* rerandomizable keys based on ECDSA as presented by Das et al. [DFL19], where the rerandomization algorithms multiply a random value to each key. Second, $\mathsf{REC}[\mathsf{H}_1]$ is designed for key-prefixed messages, i.e., upon executing $\mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}, m)$ for a secret key sk and a message m, the message is first extended to a *key-prefixed message* $\mathsf{pm} \leftarrow (\mathsf{pk}, m)$ where pk represents the public key corresponding to sk. Then the prefixed message pm is signed under sk.

We prove that $\mathsf{REC}[\mathsf{H}_1]$ satisfies **uf-cma-hrk1** security by providing a reduction from the **uf-cma1** security of the standard ECDSA scheme $\mathsf{EC}[\mathsf{H}_0]$. An integral part of the reduction is the observation that there exists a so-called "related key attack" (RKA) in the scheme $\mathsf{EC}[\mathsf{H}_0]$. An RKA allows to transform a signature that is valid under a public key pk_0 into a signature that is valid under another public key pk_1 given there exists a specific relation between pk_1 and pk_0. The RKA in $\mathsf{EC}[\mathsf{H}_0]$ allows to use a signature σ that is valid under a public key pk_0 as a valid signature under a public key pk_1 in case pk_1 and pk_0 are related as $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$, where ρ must satisfy $\rho = \frac{H_0(m_0) - H_1(m_1)}{r}$. We formally describe this related key attack in the following Lemma.

Lemma 3.1 *Let* $\mathsf{H}_0,\ \mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ *be hash functions (modeled as random oracles). Suppose that* $\sigma = (r, s)$ *is a valid signature on message* $m_0 \in \{0,1\}^*$ *w.r.t.* $\mathsf{EC}[\mathsf{H}_0]$ *and public key* pk_0*, i.e.,*

Algorithm $\mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}\,(\mathsf{sk}, m)$	Algorithm $\mathsf{REC}[\mathsf{H}_1].\mathsf{RandSK}\,(\mathsf{sk}; \rho)$
00 $\mathsf{pm} \leftarrow (\mathsf{pk}, m)$	00 $\mathsf{sk}' \leftarrow (\mathsf{sk} + \rho) \mod p$
01 $\sigma \leftarrow \mathsf{EC}[\mathsf{H}_1].\mathsf{Sign}\,(\mathsf{sk}, \mathsf{pm})$	01 Return sk'
02 Return σ	
Algorithm $\mathsf{REC}[\mathsf{H}_1].\mathsf{Verify}\,(\mathsf{pk}, \sigma, m)$	Algorithm $\mathsf{REC}[\mathsf{H}_1].\mathsf{RandPK}\,(\mathsf{pk}; \rho)$
03 $\mathsf{pm} \leftarrow (\mathsf{pk}, m)$	02 $\mathsf{pk}' \leftarrow (\mathsf{pk} + \rho \cdot G)$
04 Return $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}\,(\mathsf{pk}, \sigma, \mathsf{pm})$	03 Return pk'

Figure 2: Key-prefixed version of the ECDSA signature scheme with perfectly rerandomizable keys $\mathsf{REC}[\mathsf{H}_1]$:= ($\mathsf{REC}[\mathsf{H}_1].\mathsf{Gen} = \mathsf{EC}[\mathsf{H}_1].\mathsf{Gen}$, $\mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}$, $\mathsf{REC}[\mathsf{H}_1].\mathsf{Verify}$, $\mathsf{REC}[\mathsf{H}_1].\mathsf{RandSK}$, $\mathsf{REC}[\mathsf{H}_1].\mathsf{RandPK}$) based on the ECDSA signature scheme $\mathsf{EC}[\mathsf{H}_1]$. Above $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ denotes a hash function.

$\mathsf{EC}[\mathsf{H}_0].\mathsf{Verify}(\mathsf{pk}_0, \sigma, m_0) = 1$. *Furthermore, let* $\rho = \frac{H_0(m_0) - H_1(m_1)}{r} \pmod p$. *Then* σ *is also a valid signature on message* $m_1 \in \{0,1\}^*$ *w.r.t.* $\mathsf{EC}[\mathsf{H}_1]$ *and public key* $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$, *i.e.,* $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m_1) = 1$.

Proof of Lemma 3.1. We have to show that $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m_1) = 1$ for $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$ and $\rho = \frac{H_0(m_0) - H_1(m_1)}{r} \pmod p$. Note that $\sigma = (r, s)$, where $s = t^{-1}(\mathsf{H}_0(m_0) + r\mathsf{sk}_0) \pmod p$ and r represents the x-coordinate of the elliptic curve point $t \cdot G$ for $t \xleftarrow{\$} \mathbb{Z}_p$. As shown in Figure 1, $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m_1)$ computes the following:

$$
\begin{aligned}
& u_1 \cdot G + u_2 \cdot \mathsf{pk}_1 \\
=& \mathsf{H}_1(m_1) \cdot s^{-1} \cdot G + r \cdot s^{-1} \cdot \left(\mathsf{pk}_0 + \frac{\mathsf{H}_0(m_0) - \mathsf{H}_1(m_1)}{r} \cdot G \right) \\
=& s^{-1} \cdot G \left(\mathsf{H}_1(m_1) + r \cdot \mathsf{sk}_0 + \mathsf{H}_0(m_0) - \mathsf{H}_1(m_1) \right) \\
=& s^{-1} \cdot G \left(r \cdot \mathsf{sk}_0 + \mathsf{H}_0(m_0) \right) \\
=& t \cdot (\mathsf{H}_0(m_0) + r\mathsf{sk}_0)^{-1} \cdot (\mathsf{H}_0(m_0) + r\mathsf{sk}_0) \cdot G = t \cdot G
\end{aligned}
$$

Since the x-coordinate of $t \cdot G$ equals $r \pmod p$, it holds that $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m_1) = 1$. ■

The RKA from Lemma 3.1 can be extended to an RKA between the schemes $\mathsf{EC}[\mathsf{H}_0]$ and $\mathsf{REC}[\mathsf{H}_1]$ such that a valid signature under pk_0 for a *prefixed* message $\mathsf{pm} \leftarrow (\mathsf{pk}_1, m)$ in $\mathsf{EC}[\mathsf{H}_0]$ is also valid in $\mathsf{REC}[\mathsf{H}_1]$ under pk_1 for message m. This RKA allows to transfer a valid signature from $\mathsf{EC}[\mathsf{H}_0]$ to a valid signature in $\mathsf{REC}[\mathsf{H}_1]$ and vice versa in case pk_0 and pk_1 satisfy the relation from Lemma 3.1. We formally present this RKA in the following Lemma.

Lemma 3.2 *Let* H_0, $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ *be hash functions (modeled as random oracles). Let* $m \in \{0,1\}^*$ *and suppose that* $\sigma = (r, s)$ *is a valid signature on message* $\mathsf{pm} \leftarrow (\mathsf{pk}_1, m)$ *w.r.t.* $\mathsf{EC}[\mathsf{H}_0]$ *and public key* pk_0, *i.e.,* $\mathsf{EC}[\mathsf{H}_0].\mathsf{Verify}(\mathsf{pk}_0, \sigma, \mathsf{pm}) = 1$. *Furthermore, suppose that* $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$ *where* $\rho = \frac{H_0(\mathsf{pm}) - H_1(\mathsf{pm})}{r} \pmod p$. *Then* σ *is also a valid signature on message* m *w.r.t.* $\mathsf{REC}[\mathsf{H}_1]$ *and public key* pk_1, *i.e.,* $\mathsf{REC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m) = 1$.

Proof of Lemma 3.2. We have to show that $\mathsf{REC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m) = 1$ for $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$ and $\rho = \frac{H_0(\mathsf{pm}) - H_1(\mathsf{pm})}{r} \pmod p$, where $\mathsf{pm} \leftarrow (\mathsf{pk}_1, m)$. Note that $\sigma = (r, s)$, where $s = t^{-1}(\mathsf{H}_0(\mathsf{pm}) + r\mathsf{sk}_0) \pmod p$ and r represents the x-coordinate of the elliptic curve point $t \cdot G$ for $t \xleftarrow{\$} \mathbb{Z}_p$. As shown in figure 7, $\mathsf{REC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m)$ first computes the prefixed message $\mathsf{pm} \leftarrow (\mathsf{pk}_1, m)$ and then runs $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, \mathsf{pm})$. The rest follows from the proof of Lemma 3.1 with $m_0 = m_1 = \mathsf{pm}$. ■

3.1 Security analysis of REC

In this section, we analyze the one-per message unforgeability of the honestly rerandomizable signature scheme, or in short the **uf-cma-hrk1** security of the scheme $\mathsf{REC}[\mathsf{H}_1]$. We prove the following theorem.

Theorem 3.3 *Let* $\mathsf{H}_0, \mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ *be hash functions (modeled as random oracles). Let* $\mathcal{A}$ *be an algorithm that plays in the game* $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$*. Then there exists an algorithm* $\mathcal{C}$ *running in roughly the same time as* $\mathcal{A}$*, such that*

$$\mathsf{Adv}^{\mathcal{C}}_{\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}} \geq \left(\mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}} - \frac{q^2_{\mathsf{H}_1}}{p} \right) \cdot \frac{1}{q}$$

where q_{H_1} *and* q *are the number of random oracle queries and* `Rand` *queries, respectively, that* $\mathcal{A}$ *makes.*

Before providing the full formal proof of Theorem 3.3, we give some intuition on how we overcome the main difficulties in our simulation. At a high level, the idea is to reduce the **uf-cma-hrk1** security of the additively rerandomizable ECDSA construction $\mathsf{REC}[\mathsf{H}_1]$ from the **uf-cma1** security of ECDSA construction $\mathsf{EC}[\mathsf{H}_0]$. Therefore, the proof essentially consists of building a reduction $\mathcal{C}$ trying to come up with a valid forgery to win the $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ game, by simulating the $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$ game to adversary $\mathcal{A}$ using the RKA from Lemma 3.2. In the $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ game, $\mathcal{C}$ obtains a public key $\mathsf{pk}_{\mathcal{C}}$ from its challenger. It can query an oracle `Sign` to get signatures w.r.t. $\mathsf{pk}_{\mathcal{C}}$. $\mathcal{C}$ also has access to a random oracle H_0. $\mathcal{C}$'s goal is to somehow embed its public key $\mathsf{pk}_{\mathcal{C}}$ in one of the rerandomized public keys pk^* under which $\mathcal{A}$ eventually returns a forgery $(\mathsf{pk}^*, \sigma^*, m^*)$. The hope is that $\mathcal{C}$ can use $(\mathsf{pk}^*, \sigma^*, m^*)$ to win its own game $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$.

In more detail, $\mathcal{C}$'s strategy works as follows. Instead of directly using $\mathsf{pk}_{\mathcal{C}}$, $\mathcal{C}$ generates the challenge public key for $\mathcal{A}$ by additively shifting $\mathsf{pk}_{\mathcal{C}}$ with a freshly sampled $\tilde{\rho} \xleftarrow{\$} \mathcal{R}$, i.e., $\mathsf{pk} \leftarrow \mathsf{pk}_{\mathcal{C}} - \tilde{\rho} \cdot G$. When $\mathcal{A}$ asks for a signature under a key $\mathsf{pk}' = \mathsf{pk} + \rho \cdot G$, $\mathcal{C}$ can simulate such signatures by querying its `Sign` oracle and employing the RKA from Lemma 3.2. This is because, to the adversary $\mathcal{A}$, pk' looks like a rerandomization of pk, while in fact, it is derived from $\mathsf{pk}_{\mathcal{C}}$ as $\mathsf{pk}' = \mathsf{pk} + \rho \cdot G = (\mathsf{pk}_{\mathcal{C}} - \tilde{\rho} \cdot G) + \rho \cdot G$. To make this simulation work, the random oracle H_1 must be carefully programmed by $\mathcal{C}$ such that the relation between ρ, H_0 and H_1 satisfies $\mathsf{H}_1(m) = \mathsf{H}_0(m) - r \cdot \rho \pmod p$ (according to Lemma 3.2), where $(r, s) := \sigma$ is the signature[3].Note that, due to the programming of the random oracle, the first simulated signature for every message and randomness pair (m, ρ) fully determines $\mathsf{H}_1(m)$. Hence, the simulated signing oracle in $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$ can be queried at most once on every input pair (m, ρ). $\mathcal{C}$'s strategy to win $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ is to embed $\tilde{\rho}$ at random as an answer to one of the `Rand` queries in $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$. For signing queries w.r.t. $\tilde{\mathsf{pk}}$, $\mathcal{C}$ does not reprogram H_1; instead, it uses H_0 and signatures obtained from the signing oracle in $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ directly. If $\mathcal{A}$ returns a valid forgery σ^* w.r.t. to $\mathsf{pk}^* = \tilde{\mathsf{pk}} = \mathsf{pk} + \tilde{\rho} \cdot G$, then $\mathcal{C}$ can simply use this forgery to win the $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ game. This is because $\mathsf{pk}^* = \mathsf{pk} + \tilde{\rho} \cdot G = \mathsf{pk}_{\mathcal{C}} - \tilde{\rho} \cdot G + \tilde{\rho} \cdot G = \mathsf{pk}_{\mathcal{C}}$. Note that pk^* is *the only key* for which the forgery σ^* is valid in game $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$. For any other key pk', the simulation of the signing oracle in $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$ requires to reprogram H_1 on any message that is prefixed with pk'. Since this involves a signing query on that very message to the signing oracle in $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$, the forgery would no longer be fresh in the latter game. This guessing on $\mathcal{C}$'s part is also the reason that our reduction is not tight.

We now provide the full formal proof.

Proof. For this proof, we consider an adversary $\mathcal{A}$ playing in the $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$ game relative to a random oracle H_1. Below, we present a series of games $\boldsymbol{G}_0$ to $\boldsymbol{G}_6$ where the following holds.

$$\mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}} = \Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G_6}^{\mathcal{A}} = 1] + \frac{{q_{\mathsf{H}_1}}^2}{p}$$

Game $\boldsymbol{G_0}$: This game is equivalent to the original game, namely $\textbf{uf-cma-hrk1}^{\mathcal{A}}_{\mathsf{REC}[\mathsf{H}_1]}$. In particular, a key pair $(\mathsf{sk}, \mathsf{pk})$ is sampled as $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{REC}[\mathsf{H}_1].\mathsf{Gen}(\mathsf{par})$. The adversary $\mathcal{A}$ is given pk as the challenge public key and oracle access to `Rand`, `RSign` and random oracle H_1. $\mathcal{A}$ can query `Rand` to receive a randomness ρ and make a follow-up query to `RSign` to receive a signature on message m with respect to the rerandomized key $\mathsf{pk}' \leftarrow \mathsf{pk} + \rho \cdot G$. In particular, $\mathcal{A}$ is allowed to query `RSign` on every input pair (m, ρ) at most once. Additionally, $\mathcal{A}$ can make direct queries to the random oracle H_1. Eventually, in order to win the game, $\mathcal{A}$ has to come up with a valid forgery σ^* on a

[3]An important aspect of this simulation is that $\mathcal{C}$ can program H_1 whenever it observes a query m to H_1 that is prefixed with a previously rerandomized key. In particular, this can be done *before* m is ever queried to the signing oracle in $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$.

new message m^* with respect to a randomness ρ^*. Since $\boldsymbol{G_0}$ proceeds as **uf-cma-hrk1** we have that $\Pr[\boldsymbol{G_0}^{\mathcal{A}} = 1] = \Pr[\textbf{uf-cma-hrk1}^{\mathcal{A}}_{\mathsf{REC}[\mathsf{H}_1]} = 1] = \mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}}$.

Game $\boldsymbol{G_1}$: This game is similar to game $\boldsymbol{G_0}$ with the following modification. $\mathcal{A}$ is now given a public key $\widetilde{\mathsf{pk}}$ instead of pk (which served as the challenge public key in $\boldsymbol{G}_0$) as the challenge public key. $\widetilde{\mathsf{pk}}$ is derived as $\widetilde{\mathsf{pk}} \leftarrow \mathsf{pk} - \tilde{\rho} \cdot G$ with a freshly sampled randomness $\tilde{\rho} \xleftarrow{\$} \mathcal{R}$. The corresponding secret key is obtained as $\widetilde{\mathsf{sk}} = \mathsf{sk} - \tilde{\rho}$.

Due to the perfect rerandomizablity of keys of the rerandomizable signature scheme REC, pk is indistinguishable from $\widetilde{\mathsf{pk}}$. Hence, we have $\Pr[\boldsymbol{G_0}^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G_1}^{\mathcal{A}} = 1]$.

Game $\boldsymbol{G_2}$: This game is similar to game $\boldsymbol{G_1}$ with the following modification in the `Rand` oracle. An index j is sampled uniformly at random from the set $\{1, \ldots, q\}$, where q is an upper bound on the number of queries to the oracle `Rand`. The game returns $\tilde{\rho}$ at the j^{th} `Rand` query. For all other queries, ρ is sampled randomly as $\rho \xleftarrow{\$} \mathcal{R}$.

Since both $\tilde{\rho}$ and ρ are sampled randomly from $\mathcal{R}$, the output distribution of the `Rand` oracle is the same in games $\boldsymbol{G_1}$ and $\boldsymbol{G_2}$. Hence, we have $\Pr[\boldsymbol{G_2}^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G_1}^{\mathcal{A}} = 1]$.

Game $\boldsymbol{G_3}$: This game behaves exactly like the game $\boldsymbol{G_2}$ with the following modifications: First, the game internally maintains a random oracle H_0 (in addition to H_1) in a straightforward manner, by storing a list H_0 of query/response pairs. Second, the game programs the oracle H_1 by maintaining three lists H_1, H_1' and Γ, where the first two will be used as possible replies to queries to H_1, and Γ stores pre-computed signatures. In the beginning of the game, H_1, H_1' and Γ are initially set to $\perp$ in each entry. Whenever $\mathcal{A}$ queries a message m to H_1, the values $H_1[m]$, $H_1'[m]$ and $\Gamma[m]$ are set in one of two ways depending on whether m is prefixed with a public key pk' or not. Here, pk' is a rerandomized form of the public key $\widetilde{\mathsf{pk}}$ (i.e., $\mathsf{pk}' \leftarrow \widetilde{\mathsf{pk}} + \rho \cdot G$ where $\rho \leftarrow$ `Rand` is a previous answer to any `Rand` oracle query), where $\widetilde{\mathsf{pk}} = \mathsf{pk} - \tilde{\rho} \cdot G$ (see Game $\boldsymbol{G}_1$). Concretely, on query m to H_1, the lists H_1, H_1' and Γ are maintained in the following way:

- If H_1 has already been programmed in a previous query, i.e., $H_1[m] \neq \perp$, return $H_1[m]$.
- Else $H_1[m] = \perp$, then sample uniformly at random $h \xleftarrow{\$} \mathbb{Z}_p$, set $H_1[m] = h$, and proceed as follows:
  - Case 1: m is of the form (pk', m'), where $\mathsf{pk}' = \widetilde{\mathsf{pk}} + \rho \cdot G = \mathsf{pk} + (\rho - \tilde{\rho}) \cdot G$, for $\rho \in \mathsf{RList}$. Derive a signature σ as $\sigma \leftarrow \mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}', m')$ for $\mathsf{sk}' = \widetilde{\mathsf{sk}} + \rho = \mathsf{sk} + (\rho - \tilde{\rho}) \pmod p$ and parse $\sigma := (r, s)$. Then set $H_1'[m] = H_0[m] - r \cdot (\rho - \tilde{\rho}) \pmod p$ and $\Gamma[m] = \sigma$. Finally return $H_1[m]$.
  - Case 2: m is not of the form (pk', m'). Set $\Gamma[m] = \epsilon$ and return $H_1[m]$.

In both the cases, the output of H_1 is uniformly distributed from $\mathcal{A}$'s point of view. It follows that $\Pr[\boldsymbol{G_2}^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G_3}^{\mathcal{A}} = 1]$.

Game $\boldsymbol{G_4}$: This game proceeds as the previous game with a modification in the `Rand` oracle. Upon $\mathcal{A}$ querying the `Rand` oracle, sample ρ as before, then compute the rerandomized public key $\mathsf{pk}' \leftarrow \widetilde{\mathsf{pk}} + \rho \cdot G$ and check if there exists a message m with prefix pk' such that $\Gamma[m] = \epsilon$. In that case, the game aborts.

Claim 3.4 Let E_1 be the event that the game $\boldsymbol{G_4}$ aborts during a `Rand` query. Then, we have that $\Pr[\mathsf{E}_1] \leq \frac{{q_{\mathsf{H}_1}}^2}{p}$.

Proof. Event E_1 can only occur if $\mathcal{A}$ has queried H_1 on input m with prefix $\mathsf{pk}' \leftarrow \widetilde{\mathsf{pk}} + \rho \cdot G$ prior to making a query to `Rand` that returns ρ. Since $\mathcal{A}$ makes at most q_{H_1} queries to H_1, for each query to `Rand` that the adversary $\mathcal{A}$ makes, we have that with probability $\frac{q_{\mathsf{H}_1}}{p}$ we receive a value ρ such that $\mathsf{pk}' \leftarrow \widetilde{\mathsf{pk}} + \rho \cdot G$ is a prefix of input m that was earlier made to H_1. Since there are at most q such queries to `Rand` by taking the union bound over q_{H_1} we obtain $\Pr[\mathsf{E}_1] = \sum_{i=1}^{q_{\mathsf{H}_1}} \frac{q_{\mathsf{H}_1}}{p} = \frac{{q_{\mathsf{H}_1}}^2}{p}$. ∎

From the above, we have that $\Pr[\boldsymbol{G_3}^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G_4}^{\mathcal{A}} = 1] + \frac{{q_{\mathsf{H}_1}}^2}{p}$.

Game $\boldsymbol{G_5}$: This game is similar to the game $\boldsymbol{G_4}$ except for a modification in the `RSign` oracle. Upon $\mathcal{A}'s$ query on input (m, ρ), the game simulates the `RSign` oracle in the following manner. It computes the rerandomized public key $\mathsf{pk}' \leftarrow \widetilde{\mathsf{pk}} + \rho \cdot G$ and creates the public key prefixed message $\mathsf{pm} \leftarrow (\mathsf{pk}', m)$. The signature is implicitly derived via querying the simulated random oracle H_1 (see Game $\boldsymbol{G_3}$ above) on

input the prefixed message pm. This results into $\Gamma[\mathsf{pm}] = \sigma = \mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}', m)$, which is returned as the response to the signature query.

Observe that all queries to $\mathtt{RSign}$ on input the tuple (m, ρ) output the same signature. However, since ECDSA signatures are randomized, the output of $\mathtt{RSign}$ should be different with overwhelming probability for each query on the same input tuples. Here, we exploit that $\mathcal{A}$ is allowed to query $\mathtt{RSign}$ at most once for the same input pair (m, ρ). Hence, the output distribution of $\mathtt{RSign}$ is identical to the distribution of the $\mathtt{RSign}$ oracle in the previous game and it holds that $\Pr[\boldsymbol{G_4}^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G_5}^{\mathcal{A}} = 1]$.

Game $\boldsymbol{G_6}$: This game is similar to game $\boldsymbol{G_5}$ except for the following changes: In the oracles $\mathtt{RSign}$ and H_1 the game uses $\mathsf{EC}[H_0].\mathsf{Sign}$ instead of $\mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}$ to compute the signatures stored in Γ (and in case of $\mathtt{RSign}$ this implicitly happens via H_1). More precisely, when H_1 is queried on $\mathsf{pm} = (\mathsf{pk}', m')$, where $\mathsf{pk}' = \widetilde{\mathsf{pk}} + \rho \cdot G = \mathsf{pk} + (\rho - \tilde{\rho}) \cdot G$ for $\rho \in \mathsf{RList}$, we derive $\sigma \leftarrow \mathsf{EC}[H_0].\mathsf{Sign}(\mathsf{sk}, \mathsf{pm})$, for $\mathsf{sk}' = \mathsf{sk} + (\rho - \tilde{\rho}) \pmod p$. Furthermore, upon H_1 being queried on m, H_1 returns $H_1'[m]$ instead of $H_1[m]$ whenever $\Gamma[m] \neq \perp$ and $\Gamma[m] \neq \epsilon$.

Claim 3.5 It holds that $\Pr[\boldsymbol{G_5}^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G_6}^{\mathcal{A}} = 1]$.

Proof. First, note that in this game, H_1 returns $H_0[m] - r \cdot (\rho - \tilde{\rho})$ on a message m for which a signature is stored in Γ. We have to show now that when H_1 is queried on $\mathsf{pm} = (\mathsf{pk}', m')$, where $\mathsf{pk}' = \mathsf{pk} + (\rho - \tilde{\rho}) \cdot G$ and $\mathsf{sk}' = \mathsf{sk} + (\rho - \tilde{\rho}) \pmod p$ for $\rho \in \mathsf{RList}$, we derive $\sigma \leftarrow \mathsf{EC}[H_0].\mathsf{Sign}(\mathsf{sk}, \mathsf{pm})$ (Game $\boldsymbol{G_6}$) instead of computing $\sigma \leftarrow \mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}', m')$ (Game $\boldsymbol{G_5}$).

To this end, we recall Lemma 3.2, which states that if $\sigma = (r, s)$ is a valid signature for $\mathsf{pm} \leftarrow (\mathsf{pk}', m')$ under pk w.r.t. $\mathsf{EC}[H_0]$, it is also a valid signature for m' under $\mathsf{pk}' \leftarrow \mathsf{pk} + (\rho - \tilde{\rho}) \cdot G$ w.r.t. $\mathsf{REC}[\mathsf{H}_1]$, if it holds that $\mathsf{H}_1(\mathsf{pm}) = \mathsf{H}_0(\mathsf{pm}) - r \cdot (\rho - \tilde{\rho}) \pmod p$. Note that we replaced the $\mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}$ procedure call on a message m' in $\boldsymbol{G_5}$ by a $\mathsf{EC}[\mathsf{H}_0].\mathsf{Sign}$ procedure call on a prefixed message $\mathsf{pm} \leftarrow (\mathsf{pk}', m')$, where $\mathsf{pk}' = \mathsf{pk} + (\rho - \tilde{\rho}) \cdot G$. It remains to show that the condition $\mathsf{H}_1(\mathsf{pm}) = \mathsf{H}_0(\mathsf{pm}) - r \cdot (\rho - \tilde{\rho}) \pmod p$ holds. But since $H_1'[\mathsf{pm}] = H_0[\mathsf{pm}] - r \cdot (\rho - \tilde{\rho}) \pmod p$ is programmed accordingly (latest when $\mathtt{RSign}$ is queried), this follows directly. ∎

Combining results from $\boldsymbol{G}_0$ to $\boldsymbol{G}_6$, we have that

$$\Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] + \frac{{q_{\mathsf{H}_1}}^2}{p}. \tag{1}$$

Reduction to uf-cma1 security. Having shown that the original $\textbf{uf-cma-hrk1}^{\mathcal{A}}_{\mathsf{REC}[\mathsf{H}_1]}$ game is indistinguishable from game $\boldsymbol{G_6}$, it remains to show that an adversary $\mathcal{A}$ winning in game $\boldsymbol{G}_6$ can be turned into an adversary $\mathcal{C}$ that wins $\textbf{uf-cma1}^{\mathcal{C}}_{\mathsf{EC}[\mathsf{H}_0]}$ game with related success probability. To this end, we construct $\mathcal{C}$ that runs in the game $\textbf{uf-cma1}^{\mathcal{C}}_{\mathsf{EC}[\mathsf{H}_0]}$ and simulates to $\mathcal{A}$ game $\boldsymbol{G}_6$. Thus, $\mathcal{C}$ proceeds as game $\boldsymbol{G}_6$ and leverages oracle access to its own signing oracle (with respect to its challenge public key) in the following way:

1. On input the challenge public key pk_C from $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$, the adversary $\mathcal{C}$ sets pk to $\mathsf{pk}_{\mathcal{C}}$. Note that this implicitly sets the challenge public key in $\mathcal{C}$'s simulation of $\boldsymbol{G}_6$ to $\widetilde{\mathsf{pk}} = \mathsf{pk}_{\mathcal{C}} - \tilde{\rho} \cdot G$. Hence, $\mathcal{C}$ runs $\mathcal{A}$ on input $\widetilde{\mathsf{pk}}$.

2. In case $\mathcal{A}$ returns a forgery (m^*, σ^*, ρ^*) with $\rho^* \neq \tilde{\rho}$, $\mathcal{C}$ aborts.

$\mathcal{C}$ perfectly simulates $\boldsymbol{G}_6$ for $\mathcal{A}$ except in case where it aborts. Moreover, note that in case there is no abort, we have that

$$\mathsf{pk}^* = \widetilde{\mathsf{pk}} + \rho^* \cdot G = \mathsf{pk}_{\mathcal{C}} - \tilde{\rho} \cdot G + \tilde{\rho} \cdot G = \mathsf{pk}_{\mathcal{C}}.$$

From the above programming strategy, we conclude that for $\mathcal{A}$'s queries to H_1 that are prefixed with pk^*, the oracles H_0 and H_1 are identical. It remains to calculate the success probability of $\mathcal{C}$ in winnning the $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ game in case $\mathcal{A}$ returns a valid forgery.

Claim 3.6 Let E_2 be the event that $\mathcal{A}$ outputs (m^*, σ^*, ρ^*) s.t. $(\mathsf{pm}^*, \sigma^*)$ constitutes a valid forgery in game $\textbf{uf-cma1}^{\mathcal{C}}_{\mathsf{EC}[\mathsf{H}_0]}$. Then, we have that $\Pr[\mathsf{E}_2 | \boldsymbol{G_6}^{\mathcal{A}} = 1] \geq \frac{1}{q}$, where q is the number of queries to the $\mathtt{Rand}$ oracle.

Proof. In order to prove this claim, we need to show that with probability $\frac{1}{q}$ it must hold that (1) $(\mathsf{pm}^*, \sigma^*)$ is a valid forgery in game $\mathbf{uf\text{-}cma1}^{\mathcal{C}}_{\mathsf{EC}[\mathsf{H}_0]}$ under public key $\mathsf{pk}_{\mathcal{C}}$ and (2) the $\mathtt{Sign}$ oracle of the $\mathbf{uf\text{-}cma1}^{\mathcal{C}}_{\mathsf{EC}[\mathsf{H}_0]}$ game has not been queried on input pm^*.
First, note that if σ^* is a valid signature for message (pk^*, m^*) under the public key pk^* relative to $\mathsf{REC}[\mathsf{H}_1]$, then σ^* is also a valid signature on pm^* under public key $\mathsf{pk}_{\mathcal{C}} = \mathsf{pk}^*$ relative to $\mathsf{EC}[\mathsf{H}_0]$, as H_0 and H_1 are identical for messages prefixed with pk^*. Since there are at most q possible values of ρ^* and $\mathcal{C}$ chooses one of them uniformly at random, the probability that $\mathcal{C}$'s guess is correct is at least $\frac{1}{q}$. Note that from the adversary's perspective, the public key generated at index j is no different than other public keys.
Second, since (m^*, σ^*, ρ^*) is a valid forgery in $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{A}}_{\mathsf{REC}[\mathsf{H}_1]}$, $\mathcal{A}$ has not previously queried the $\mathtt{RSign}$ oracle on input (m^*, ρ^*). Correspondingly, the $\mathtt{Sign}$ oracle of the $\mathbf{uf\text{-}cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ game has also not been queried on message pm^* and hence, $(\mathsf{pm}^*, \sigma^*)$ is a valid forgery in $\mathbf{uf\text{-}cma1}_{\mathsf{EC}[\mathsf{H}_0]}$. ∎

From Eq. 1 we get the following.

$$\mathsf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{REC}[\mathsf{H}_1]}} = \Pr[\boldsymbol{G}^{\mathcal{A}}_0 = 1] \leq \Pr[\boldsymbol{G}^{\mathcal{A}}_6 = 1] + \frac{q^2_{\mathsf{H}_1}}{p}$$

$$\text{or, } \Pr[\boldsymbol{G}^{\mathcal{A}}_6 = 1] \geq \mathsf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{REC}[\mathsf{H}_1]}} - \frac{q^2_{\mathsf{H}_1}}{p}$$

Since $\mathcal{C}$ can use a valid forgery by $\mathcal{A}$ in its own game whenever E_2 occurs,

$$\begin{aligned}\mathsf{Adv}^{\mathcal{C}}_{\mathbf{uf\text{-}cma}_{\mathsf{EC}}[\mathsf{H}_0]} &\geq \Pr[\boldsymbol{G}^{\mathcal{A}}_6 = 1] \cdot \Pr[\mathsf{E}_2 \mid \boldsymbol{G}^{\mathcal{A}}_6 = 1] = \Pr[\boldsymbol{G}^{\mathcal{A}}_6 = 1] \cdot \frac{1}{q} \\ &\geq \left(\mathsf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{REC}[\mathsf{H}_1]}} - \frac{q^2_{\mathsf{H}_1}}{p}\right) \cdot \frac{1}{q}\end{aligned}$$

∎

4 A Model for Hierarchical Deterministic Wallets

In this section, we introduce a formal model for hierarchical deterministic wallets. This model closely reflects the BIP32 specification [Wik18] with only minor differences which we list in Section 6. At a high level, a hierarchical deterministic wallet scheme can be visualized as a tree, where every node in the tree corresponds to a wallet. As is usual in a tree structure, the scheme originates from a root node, which contains a pair of master keys - a master public key mpk and a master secret key msk as well as a seed $\mathsf{ch}_{0,0}$ which we will refer to as chaincode from now on. We say that the root node is located at level 0 of the tree. The root can create a child node at level 1 and position t by deriving a new key pair $(\mathsf{pk}_{1,t}, \mathsf{sk}_{1,t})$ and a chaincode $\mathsf{ch}_{1,t}$ from its master keys and chaincode $\mathsf{ch}_{0,0}$. This child node represents a new wallet that is initiated with the key pair $(\mathsf{pk}_{1,t}, \mathsf{sk}_{1,t})$ and chaincode $\mathsf{ch}_{1,t}$ and using these values it can in turn create a child node for level 2. This child creation process can continue recursively. Note, however, that a node at level i can only create children for the *immediate* lower level, i.e., for level $i + 1$.

In our model, we distinguish between two different kinds of nodes, namely *non-hardened* and *hardened* nodes. Non-hardened nodes are, in essence, the nodes as discussed above, i.e., nodes that can be used for child creation at the next lower level. We assume that the public key and the chaincode of a non-hardened node can be corrupted by an adversary, whereas the secret key remains protected. One might think of non-hardened nodes as wallets in the hot/cold wallet setting, where the hot wallet stores the public key, the cold wallet stores the secret key and the chaincode is provided to both wallets. While the hot wallet is permanently online and thereby vulnerable to attacks, the cold wallet stays offline for the majority of the time and is therefore protected against attacks. To create a non-hardened child node at level i and at position t, its parent must generate the child node's key pair $(\mathsf{pk}_{i,t}, \mathsf{sk}_{i,t})$ and chaincode $\mathsf{ch}_{i,t}$. We model the derivation of these values in such a way that the derivation process of $\mathsf{sk}_{i,t}$ involves the parent's secret key, while the derivation of $\mathsf{pk}_{i,t}$ and $\mathsf{ch}_{i,t}$ requires only the parent's public key and chaincode (i.e., it is independent of the parent's secret key).

Hardened nodes, on the other hand, represent the leaves of the tree, i.e., we do not consider any child derivation from hardened nodes[4]. However, in comparison to non-hardened nodes we allow secret key leakage, along with public key and chaincode leakage for hardened nodes. That is, we consider full corruption of hardened nodes. Our security goal is that the secret key leakage of a hardened node does not affect the security of any other node in the tree. As opposed to non-hardened nodes, the creation process of a hardened child node requires the secret key of the parent node, i.e., even for the derivation of the child's public key and chaincode . The tree structure of a hierarchical deterministic wallet scheme, containing hardened as well as non-hardened nodes can be found in Figure 3.

While hardened nodes clearly exhibit stronger security guarantees than non-hardened nodes, the advantage of non-hardened nodes lies in the child creation process. We will illustrate this advantage in the following example. In a company there might be trusted and untrusted employees. Trusted employees operate a non-hardened node, as they are trusted to properly protect their secret key, e.g., by storing it in a cold wallet. On the other hand, untrusted employees have to operate a hardened node as they might leak their secret key or simply get compromised. Assume a trusted employee maintains a non-hardened node with key pair $(\mathsf{pk}_{i,t}, \mathsf{sk}_{i,t})$ and chaincode $\mathsf{ch}_{i,t}$. Further assume that the node is operated in a hot/cold wallet setting, i.e., the tuple $(\mathsf{sk}_{i,t}, \mathsf{ch}_{i,t})$ is stored in a cold wallet and the tuple $(\mathsf{pk}_{i,t}, \mathsf{ch}_{i,t})$ is stored in a hot wallet. If the employee wishes to receive payments to different public addresses, it can simply generate these addresses by deriving non-hardened child public keys using only the information stored in its hot wallet. In particular, the cold wallet can remain offline during this process. Only when the employee wants to spend the coins it received, it has to use $\mathsf{sk}_{i,t}$ from the cold wallet to generate the secret keys corresponding to the public addresses it generated earlier.

Another example for the usefulness of non-hardened nodes is the following. Consider a company A that operates a non-hardened node with key pair $(\mathsf{pk}_{i,t}, \mathsf{sk}_{i,t})$ and chaincode $\mathsf{ch}_{i,t}$ only to receive payments from a company B. In this case, company A can simply share $\mathsf{pk}_{i,t}$ and $\mathsf{ch}_{i,t}$ with company B, which can then by itself generate non-hardened child public keys and make the payments to those addresses. Note that in this case, company A does not have to be involved in the payment process at all.

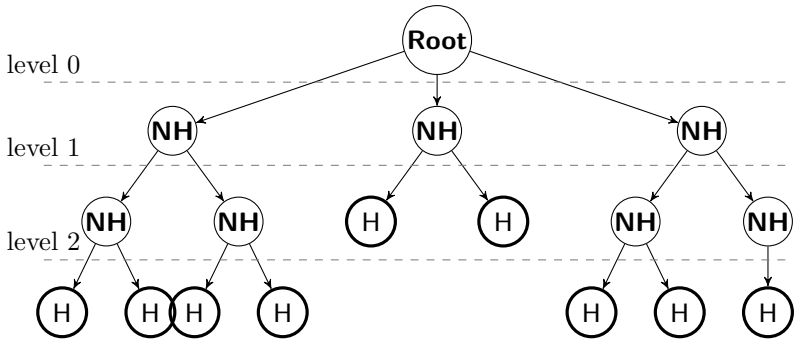

Figure 3: Tree structure of a hierarchical deterministic wallet scheme. Hardened nodes are denoted by H while non-hardened nodes are denoted by NH.

Flat Vs Hierarchical Deterministic Wallets. Let us now briefly discuss the main difference between the model for hierarchical deterministic wallets and the setting originally analyzed by Das et. al [DFL19] which we denote as *the flat model*. The key derivation process in the flat model works in the same way as the non-hardened key derivation in the hierarchical model with the difference that the flat model allows to derive keys only directly from the master key pair. Hardened nodes are not considered in the flat model. Therefore, the flat model basically represents a hierarchical wallet structure with non-hardened leaf nodes at level 1 (see Figure 4). Since the flat model allows only for non-hardened key derivation, the essential difference to the hierarchical model is that the flat model cannot allow for any secret key leakage as this would render the entire scheme insecure. Hierarchical wallets, on the other hand, introduce hardened nodes whose secret keys can be leaked without affecting the security of any other node in the tree.

In the following, we refer to a *tree* as a tuple $(h, n_{0,0}, \mathcal{N}, \mathcal{E})$ if $(\mathcal{N}, \mathcal{E})$ defines a tree of height h with node set $\mathcal{N}$ and edge set $\mathcal{E}$, and a root node $n_{0,0} \in \mathcal{N}$. We denote a directed path p_i^t of length i from the

[4]We show in Appendix A that child derivation of hardened nodes is possible under certain conditions.

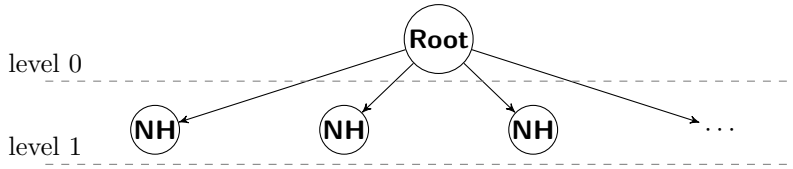

Figure 4: Tree structure of a deterministic wallet scheme in the flat setting.

root to a node $n_{i,t} \in \mathcal{N}$ at level i and position t in the tree as the corresponding ordered sequence of edges $p_i^t = (\mathsf{e}_1, \cdots, \mathsf{e}_i) \in \mathcal{E}^i$. A path of length 1 from a node $n_{i-1,s} \in \mathcal{N}$ to a node $n_{i,t} \in \mathcal{N}$ consists of only one edge which we denote as $\mathsf{e}_i^{s,t} \in \mathcal{E}$.

Definition 4.1 (Address Structure). Let $\mathcal{T} = (h, n_{0,0}, \mathcal{N}, \mathcal{E})$ be a tree. Define a labeling of the nodes in $\mathcal{N}$ as follows.

- The root node $n_{0,0}$ is labeled by an address $\mathbf{addr}_{0,0}$.
- For $1 \leq t < |\mathcal{N}|$ and $0 \leq i \leq h$, a node $n_{i,t} \in \mathcal{N}$ is labeled by an address $\mathbf{addr}_{i,t} := (\mathbf{addr}_{0,0}, p_i^t)$.

A tuple $(\mathcal{T}, \mathbf{Addr})$ is said to be an *address structure (with respect to $\mathcal{T}$)* if $\mathbf{Addr}$ consists of a set of labels for the nodes in $\mathcal{N}$ that meets the above requirements. A prefix address $\mathbf{addr}_{i,t}^j$ for a node $n_{i,t} \in \mathcal{N}$ with $0 \leq j < i \leq h$ and $t < |N|$ is a vector of length $j+1$ consisting of the first $j+1$ components of $\mathbf{addr}_{i,t} \in \mathbf{Addr}$.

We are now ready to define hierarchical deterministic wallets. In short, these schemes consist of a Setup algorithm, which initializes the root node, hardened and non-hardened secret and public key derivation algorithms $\mathsf{SKDer_H}, \mathsf{PKDer_H}$ and $\mathsf{SKDer_{NH}}, \mathsf{PKDer_{NH}}$ and finally signing and signature verification algorithms Sign and and Verify. We assume that public parameters par are known to all parties and we define appropriate secret and public key sets $\mathcal{SK}$ and $\mathcal{PK}$ respectively. We assume there exists a function $\mathsf{ToPubKey} : \mathcal{SK} \to \mathcal{PK}$ that on input a secret key from $\mathcal{SK}$ outputs the corresponding public key in $\mathcal{PK}$. Formally we have:

Definition 4.2 (Hierarchical Deterministic Wallets). Let $\mathcal{T} = (h, n_{0,0}, \mathcal{N}, \mathcal{E})$ be a tree. A *hierarchical deterministic wallet* scheme is defined w.r.t. an address structure $(\mathcal{T}, \mathbf{Addr})$ and consists of seven algorithms $\mathsf{HDWal} = (\mathsf{Setup}, \mathsf{SKDer_H}, \mathsf{SKDer_{NH}}, \mathsf{PKDer_H}, \mathsf{PKDer_{NH}}, \mathsf{Sign}, \mathsf{Verify})$ which are defined as follows:

- $\mathsf{Setup}(1^\kappa)$: The probabilistic *setup* algorithm takes as input a security parameter 1^κ and outputs a non-hardened master key pair $(\mathsf{msk}_{0,0}, \mathsf{mpk}_{0,0})$ with $\mathsf{msk}_{0,0} \in \mathcal{SK}$, $\mathsf{mpk}_{0,0} \in \mathcal{PK}$ and a chaincode $\mathsf{ch}_{0,0}$.
- $\mathsf{SKDer_H}(\mathsf{sk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$: The deterministic *hardened secret key derivation* algorithm takes as input a secret key $\mathsf{sk}_{i,s} \in \mathcal{SK}$, a chaincode $\mathsf{ch}_{i,s}$, an address $\mathbf{addr}_{i,s} \in \mathbf{Addr}$ for level $i < h$, positions s, t, as well as an edge $\mathsf{e}_{i+1}^{s,t} \in \mathcal{E}$. It outputs a secret key $\mathsf{sk}_{i+1,t} \in \mathcal{SK}$, a chaincode $\mathsf{ch}_{i+1,t}$ and an address $\mathbf{addr}_{i+1,t} \in \mathbf{Addr}$ for level $i+1$ and position t.
- $\mathsf{SKDer_{NH}}(\mathsf{sk}_{i,s}, \mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$: The deterministic *non-hardened secret key derivation* algorithm takes as input a secret key $\mathsf{sk}_{i,s} \in \mathcal{SK}$, a public key $\mathsf{pk}_{i,s} \in \mathcal{PK}$, a chaincode $\mathsf{ch}_{i,s}$, an address $\mathbf{addr}_{i,s} \in \mathbf{Addr}$ for level $i < h$, positions s, t, as well as an edge $\mathsf{e}_{i+1}^{s,t} \in \mathcal{E}$. It outputs a secret key $\mathsf{sk}_{i+1,t} \in \mathcal{SK}$, a chaincode $\mathsf{ch}_{i+1,t}$ and an address $\mathbf{addr}_{i+1,t} \in \mathbf{Addr}$ for level $i+1$ and position t.
- $\mathsf{PKDer_H}(\mathsf{sk}_{i,s}, \mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$: The deterministic *hardened public key derivation* algorithm takes as input a secret key $\mathsf{sk}_{i,s} \in \mathcal{SK}$, a public key $\mathsf{pk}_{i,s} \in \mathcal{PK}$, a chaincode $\mathsf{ch}_{i,s}$, an address $\mathbf{addr}_{i,s} \in \mathbf{Addr}$ for level $i < h$, positions s, t, as well as an edge $\mathsf{e}_{i+1}^{s,t} \in \mathcal{E}$. It outputs a public key $\mathsf{pk}_{i+1,t} \in \mathcal{PK}$, a chaincode $\mathsf{ch}_{i+1,t}$ and an address $\mathbf{addr}_{i+1,t} \in \mathbf{Addr}$ for level $i+1$ and position t.
- $\mathsf{PKDer_{NH}}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$: The deterministic *non-hardened public key derivation* algorithm takes as input a public key $\mathsf{pk}_{i,s} \in \mathcal{PK}$, a chaincode $\mathsf{ch}_{i,s}$, an address $\mathbf{addr}_{i,s} \in \mathbf{Addr}$ for level $i < h$, positions s, t, as well as an edge $\mathsf{e}_{i+1}^{s,t} \in \mathcal{E}$. It outputs a public key $\mathsf{pk}_{i+1,t} \in \mathcal{PK}$, a chaincode $\mathsf{ch}_{i+1,t}$ and an address $\mathbf{addr}_{i+1,t} \in \mathbf{Addr}$ for level $i+1$ and position t.

- $\mathsf{Sign}(\mathsf{sk}_{i,s}, m)$: The probabilistic *signing* algorithm takes as input a secret key $\mathsf{sk}_{i,s}$ and a message m. It outputs a signature σ.
- $\mathsf{Verify}(\mathsf{pk}_{i,s}, m, \sigma)$: The probabilistic *verification* algorithm takes as input a public key $\mathsf{pk}_{i,s}$, a message m and a signature σ. It outputs 0 or 1.

A hierarchical deterministic wallet is *correct*, if a secret and public key pair is derived correctly using the algorithms $\mathsf{SKDer}_{\mathsf{H}}, \mathsf{PKDer}_{\mathsf{H}}$ or $\mathsf{SKDer}_{\mathsf{NH}}, \mathsf{PKDer}_{\mathsf{NH}}$, the keys represent a valid signing key pair.

We denote keys with subscript nh (e.g., $\mathsf{sk}_{\mathsf{nh},\cdot,\cdot}$ or $\mathsf{pk}_{\mathsf{nh},\cdot,\cdot}$) as *non-hardened* keys and keys with subscript h (e.g., $\mathsf{sk}_{\mathsf{h},\cdot,\cdot}$ or $\mathsf{pk}_{\mathsf{h},\cdot,\cdot}$) as *hardened* keys. A key without the subscript nh or h indicates that it can be both a non-hardened or hardened key.

Definition 4.3 (Correctness of Hierarchical Deterministic Wallets). Let HDWal be a hierarchical deterministic wallet scheme with respect to an address structure $(\mathcal{T}, \mathbf{Addr})$. For any $\mathsf{e}_1^{0,s} \in \mathcal{E}$ and any $(\mathsf{ch}_{0,0}, \mathsf{msk}_{\mathsf{nh},0,0}, \mathsf{mpk}_{\mathsf{nh},0,0}) \in \mathsf{Setup}(1^\kappa)$, we define tuples $(\mathsf{sk}_{\mathsf{h},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s})$ and $(\mathsf{pk}_{\mathsf{h},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s})$ as

$$(\mathsf{sk}_{\mathsf{h},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s}) := \mathsf{SKDer}_{\mathsf{H}}(\mathsf{msk}_{\mathsf{nh},0,0}, \mathsf{ch}_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}_1^{0,s})$$
$$(\mathsf{pk}_{\mathsf{h},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s}) := \mathsf{PKDer}_{\mathsf{H}}(\mathsf{msk}_{\mathsf{nh},0,0}, \mathsf{ch}_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}_1^{0,s})$$

and tuples $(\mathsf{sk}_{\mathsf{nh},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s})$ and $(\mathsf{pk}_{\mathsf{nh}1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s})$ as

$$(\mathsf{sk}_{\mathsf{nh},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s}) := \mathsf{SKDer}_{\mathsf{NH}}(\mathsf{msk}_{\mathsf{nh},0,0}, \mathsf{ch}_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}_1^{0,s})$$
$$(\mathsf{pk}_{\mathsf{nh},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s}) := \mathsf{PKDer}_{\mathsf{NH}}(\mathsf{mpk}_{\mathsf{nh},0,0}, \mathsf{ch}_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}_1^{0,s}).$$

Further, for any $\mathbf{addr}_{i-1,s} \in \mathbf{Addr}$, and any edge $\mathsf{e}_i^{s,t} \in \mathcal{E}$ we define the tuples $(\mathsf{sk}_{\mathsf{h},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t})$ and $\big(\mathsf{pk}_{\mathsf{h},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}\big)$ recursively as

$$(\mathsf{sk}_{\mathsf{h},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}) := \mathsf{SKDer}_{\mathsf{H}}(\mathsf{sk}_{\mathsf{nh},i-1,s}, \mathsf{ch}_{i-1,s}, \mathbf{addr}_{i-1,s}, \mathsf{e}_i^{s,t})$$
$$\big(\mathsf{pk}_{\mathsf{h},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}\big) := \mathsf{PKDer}_{\mathsf{H}}(\mathsf{sk}_{\mathsf{nh},i-1,s}, \mathsf{ch}_{i-1,s}, \mathbf{addr}_{i-1,t}, \mathsf{e}_i^{s,t})$$

and tuples $(\mathsf{sk}_{\mathsf{nh},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t})$ and $\big(\mathsf{pk}_{\mathsf{nh},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}\big)$ as

$$(\mathsf{sk}_{\mathsf{nh},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}) := \mathsf{SKDer}_{\mathsf{NH}}(\mathsf{sk}_{\mathsf{nh},i-1,s}, \mathsf{ch}_{i-1,s}, \mathbf{addr}_{i-1,s}, \mathsf{e}_i^{s,t})$$
$$\big(\mathsf{pk}_{\mathsf{nh},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}\big) := \mathsf{PKDer}_{\mathsf{NH}}(\mathsf{pk}_{\mathsf{nh},i-1,s}, \mathsf{ch}_{i-1,s}, \mathbf{addr}_{i-1,s}, \mathsf{e}_i^{s,t})$$

HDWal is *correct* if for all $m \in \{0,1\}^*$, all $1 \le i \le h$, all $1 \le t \le (1-d^{h+1})/(1-d)$, and all $(\mathsf{ch}_{0,0}, \mathsf{msk}_{\mathsf{nh},0,0}, \mathsf{mpk}_{\mathsf{nh},0,0}) \in \mathsf{Setup}(1^\kappa)$ it holds that

$$\Pr_{\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{h},i,t}, m)}[\mathsf{Verify}(\mathsf{pk}_{\mathsf{h},i,t}, \sigma, m) = 1] = 1$$
$$\wedge \Pr_{\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{nh},i,t}, m)}[\mathsf{Verify}(\mathsf{pk}_{\mathsf{nh},i,t}, \sigma, m) = 1] = 1.$$

4.1 Oracles

Let us now describe the general capability and influence that the adversary has over the hierarchical wallet schemes. An adversary is allowed to create new hardened and non-hardened nodes in the tree. Furthermore, the adversary can corrupt the hot wallet of all non-hardened nodes, thereby learning the public key and the chaincode of these nodes, as well as learning the secret key and chaincode of the hardened nodes. As we mentioned earlier, since hardened keys are given to untrustworthy nodes, the adversary is able to corrupt both their hot and cold wallets and as such, we do not consider the hardened nodes to derive new children. One way to look at hardened nodes, is that such nodes are the root of a new tree. We will later show in App. A that an adversary cannot distinguish hardened key pairs from freshly generated keys except with negligible probability. Therefore, our model can be recursively extended to

consider settings where the hardened nodes can also derive new keys. Finally, the adversary can query any node on a freely chosen message m and receive a signature for this message. To model the above mentioned capabilities, we describe the oracles which the adversary gets access to in the unlinkability game $\mathbf{unl}_{\mathsf{HDWal}}$ and the unforgeability game $\mathbf{wufcma1}_{\mathsf{HDWal}}$.

Initially, two lists $\mathsf{SK} = \emptyset$ and $\mathsf{CH} = \emptyset$ are initialized. These are used throughout the oracles to bookkeep which secret keys and chaincodes have been leaked to the adversary. In the following, we consider a fixed address structure $(\mathcal{T}, \mathbf{Addr})$.

- **Hardened Child Creation** `HChild0`**:** On inputs an address $\mathbf{addr}_{i,s}$ and an edge $\mathsf{e}_{i+1}^{s,t}$ from $\mathcal{A}$, return $\perp$ if the address $\mathbf{addr}_{i,s}$ belongs to a hardened node or the address $\mathbf{addr}_{i,s}$ is not valid (i.e., $\mathbf{addr}_{i,s} \notin \mathbf{Addr}$). Further, return $\perp$, if the address $\mathbf{addr}_{i+1,t}$ exists already. Otherwise, compute the keys and chaincode $(\mathsf{sk}_{\mathsf{h},i,s}, \mathsf{pk}_{\mathsf{h},i,s})$ and $\mathsf{ch}_{i,s}$ for the node $\mathbf{addr}_{i,s}$ by recursively deriving keys along the path in the tree, starting from the first node in the path that has already been assigned a key. Create a hardened child with address $\mathbf{addr}_{i+1,t}$ as follows. Generate keypair $(\mathsf{sk}_{\mathsf{h},i+1,t}, \mathsf{pk}_{\mathsf{h},i+1,t})$ by executing both secret and public key derivation algorithms.

$$(\mathsf{sk}_{\mathsf{h},i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t}) \leftarrow \mathsf{SKDer}_{\mathsf{H}}(\mathsf{sk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$$
$$(\mathsf{pk}_{\mathsf{h},i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t}) \leftarrow \mathsf{PKDer}_{\mathsf{H}}(\mathsf{sk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t}).$$

  Return $\mathsf{pk}_{\mathsf{h},i+1,t}$.

- **Non-Hardened Child Creation** `NHChild0`**:** On inputs an address $\mathbf{addr}_{i,s}$ and an edge $\mathsf{e}_{i+1}^{s,t}$ from $\mathcal{A}$, return $\perp$ if the address $\mathbf{addr}_{i,s}$ belongs to a hardened node or the address $\mathbf{addr}_{i,s}$ is not valid (i.e., $\mathbf{addr}_{i,s} \notin \mathbf{Addr}$). Further, return $\perp$, if the address $\mathbf{addr}_{i+1,t}$ exists already. Otherwise, compute the keys and chaincode $(\mathsf{sk}_{\mathsf{h},i,s}, \mathsf{pk}_{\mathsf{h},i,s})$ and $\mathsf{ch}_{i,s}$ for the node $\mathbf{addr}_{i,s}$ by recursively deriving keys along the path in the tree, starting from the first node in the path that has already been assigned a key. Create a non-hardened child with address $\mathbf{addr}_{i+1,t}$ as follows. Generate keypair $(\mathsf{sk}_{\mathsf{nh},i+1,t}, \mathsf{pk}_{\mathsf{nh},i+1,t})$ by executing both key derivation algorithms

$$(\mathsf{sk}_{\mathsf{nh},i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t}) \leftarrow \mathsf{SKDer}_{\mathsf{NH}}(\mathsf{sk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$$
$$(\mathsf{pk}_{\mathsf{nh},i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t}) \leftarrow \mathsf{PKDer}_{\mathsf{NH}}(\mathsf{pk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t}).$$

  Return $\mathsf{pk}_{\mathsf{nh},i+1,t}$.

- **Signing** `HDSign0`**:** On input message m and an address $\mathbf{addr}_{i,s}$ from $\mathcal{A}$, proceed as follows. Return $\perp$ if the address $\mathbf{addr}_{i,s}$ is not valid (i.e., $\mathbf{addr}_{i,s} \notin \mathbf{Addr}$). Further, check if $\mathbf{addr}_{i,s}$ has already been queried to either `NHChild0` or `HChild0` and return $\perp$ if this is not the case. Let $\mathsf{sk}_{i,s}$ be the secret key for the node with address $\mathbf{addr}_{i,s}$. Then compute a signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{i,s}, m)$, add m to the message list $\mathsf{SigList}[\mathbf{addr}_{i,s}]$ and return σ.[5]

- **Chaincode Leakage** `CHLeak0`**:** On input an address $\mathbf{addr}_{i,s}$ from $\mathcal{A}$, check if $\mathbf{addr}_{i,s}$ has already been queried to either `NHChild0` or `HChild0` and return $\perp$ if this is not the case. Set $\mathsf{CH}[\mathbf{addr}_{i,s}] = 1$ to denote that the chaincode $\mathsf{ch}_{i,s}$ of address $\mathbf{addr}_{i,s}$ has been leaked and return $(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s})$.

- **Secret Key Leakage (for hardened node)** `SKLeak0`**:** On input an address $\mathbf{addr}_{i,s}$ from $\mathcal{A}$, check if the address is that of the root, i.e., $\mathbf{addr}_{i,s} = \mathbf{addr}_{0,0}$ or if the address belongs to a non-hardened node; in this case, return $\perp$. Further, check if $\mathbf{addr}_{i,s}$ has already been queried to either `NHChild0` or `HChild0` and return $\perp$ if this is not the case. Else, set $\mathsf{SK}[\mathbf{addr}_{i,s}] = 1$ and $\mathsf{CH}[\mathbf{addr}_{i,s}] = 1$ to denote that the secret key $\mathsf{sk}_{h,i,s}$ and the chaincode $\mathsf{ch}_{i,s}$ of address $\mathbf{addr}_{i,s}$ have been leaked and return $(\mathsf{sk}_{h,i,s}, \mathsf{ch}_{i,s})$.

4.2 Unlinkability

Intuitively, the notion of unlinkability for hierarchical deterministic wallets guarantees that public keys in the tree, i.e., public keys that have been derived directly or indirectly from the master key of the tree

[5] In case of one-per message unforgeability, the oracle aborts if it has been queried previously on input $(m, \mathbf{addr}_{i,s})$.

root, cannot be distinguished from from a freshly generated public key. More concretely, the distribution of public keys from the tree should be computationally indistinguishable from a distribution of public keys that have been derived from an independently chosen master key. While this is a valuable privacy notion, it does not quite model practical scenarios in the hot/cold wallet setting. Recall that this setting assumes public keys and chaincodes to be stored in hot wallets, which are prone to corruptions. Therefore, we extend the unlinkability notion as described above in the following way. We consider hot wallet corruption upon which the public key and chaincode of the corrupted wallet are leaked. This extended notion gives more power to the adversary and is more close to the capabilities that an adversary has in real life scenarios. Naturally, the adversary can distinguish the distribution of keys derived from public keys of corrupted hot wallets from a distribution of public keys that have been derived from an independently chosen master key. Therefore, in our new unlinkability notion the adversary should not be able to distinguish the distribution of keys derived from non-compromised hot wallets and keys derived from independently chosen master keys.

In the following we describe the unlinkability game $\mathbf{unl}_{\mathsf{HDWal}}$ with respect to a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. In the first step of the game, the challenger generates a fresh master key pair and a chaincode via the execution of $\mathsf{Setup}(1^\kappa)$. The adversary receives the master public key as input and obtains access to all oracles as described in subsection 4.1. At some point, the adversary outputs an address $\mathbf{addr}_{i,s}$ and an edge $\mathsf{e}_{i+1}^{s,t}$ and receives a public key from the challenger. This public key is either the correct key for the node at address $\mathbf{addr}_{i,s}$ or a public key derived for a random address from a fresh master public key. $\mathcal{A}$ wins the game if it can successfully distinguish these two scenarios. In the following we give a detailed description of the game $\mathbf{unl}_{\mathsf{HDWal}}$:

Game $\mathbf{unl}_{\mathsf{HDWal}}$:

- **Setup Phase:** The challenger computes $(\mathsf{ch}_{0,0}, \mathsf{msk}_{0,0}, \mathsf{mpk}_{0,0}) \leftarrow \mathsf{Setup}(1^\kappa)$ and sends $\mathsf{mpk}_{0,0}$ to $\mathcal{A}$.
- **Online Phase:** On input the security parameter and the master public key $\mathsf{mpk}_{0,0}$, the adversary $\mathcal{A}$ is allowed to make queries to the oracles as explained in subsection 4.1.
- **Output Phase:** Eventually, $\mathcal{A}$ chooses an address $\mathbf{addr}_{i,s}$, an edge $\mathsf{e}_{i+1}^{s,t}$ and a value $c \in \{\mathsf{h}, \mathsf{nh}\}$ and sends them to the challenger. Let $(\mathsf{sk}_{i,s}, \mathsf{pk}_{i,s})$ be the key pair and $\mathsf{ch}_{i,s}$ the chaincode of the node at address $\mathbf{addr}_{i,s}$. If the address $\mathbf{addr}_{i,s}$ belongs to a hardened node, $\mathcal{C}$ returns $\perp$. Otherwise, the challenger chooses a bit $b \xleftarrow{\$} \{0,1\}$ and generates a public key $\mathsf{pk}_{i+1,t}$ as follows:
  - If $b = 0$:
    - If $c = \mathsf{h}$: $\mathcal{C}$ computes $(\mathsf{pk}_{\mathsf{h},i+1,t}, \cdot, \cdot) \leftarrow \mathsf{PKDer}_{\mathsf{H}}(\mathsf{sk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$.
    - If $c = \mathsf{nh}$: If the chaincode for $\mathbf{addr}_{i,s}$ or any of its prefix addresses has been leaked, i.e., $\mathsf{CH}[\mathbf{addr}_{i,s}^j] = 1$, for any $j < i$, then $\mathcal{C}$ returns $\perp$. Else, $\mathcal{C}$ computes $(\mathsf{pk}_{\mathsf{nh},i+1,t}, \cdot, \cdot) \leftarrow \mathsf{PKDer}_{\mathsf{NH}}(\mathsf{pk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$.
  - If $b = 1$: The challenger computes $(\mathsf{ch}'_{0,0}, \mathsf{msk}'_{0,0}, \mathsf{mpk}'_{0,0}) \leftarrow \mathsf{Setup}(1^\kappa)$.
    - If $c = \mathsf{h}$: $\mathcal{C}$ derives a public key $\mathsf{pk}'_{\mathsf{h},1,t} \leftarrow \mathsf{PKDer}_{\mathsf{H}}(\mathsf{msk}'_{0,0}, \mathsf{ch}'_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}_1^{0,t})$.
    - If $c = \mathsf{nh}$: If the chaincode for $\mathbf{addr}_{i,s}$ or any of its prefix addresses has been leaked, i.e., $\mathsf{CH}[\mathbf{addr}_{i,s}^j] = 1$, for any $j < i$, then $\mathcal{C}$ returns $\perp$. Else $\mathcal{C}$ derives a public key $\mathsf{pk}'_{\mathsf{nh},1,t} \leftarrow \mathsf{PKDer}_{\mathsf{NH}}(\mathsf{mpk}'_{0,0}, \mathsf{ch}'_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}_1^{0,t})$.
  - Based on the value of b and c, the challenger sends to the adversary either $\mathsf{pk}_{\mathsf{h},i+1,t}$ or $\mathsf{pk}_{\mathsf{nh},i+1,t}$ or $\mathsf{pk}'_{\mathsf{h},1,t}$ or $\mathsf{pk}'_{\mathsf{nh},1,t}$.
- The adversary can continue to make oracle queries under the restrictions as mentioned above.
- Eventually, $\mathcal{A}$ outputs a bit b' and wins the game if $b = b'$.

We define the advantage of an adversary $\mathcal{A}$ in $\mathbf{unl}_{\mathsf{HDWal}}$ as

$$\mathsf{Adv}^{\mathcal{A}}_{\mathbf{unl}_{\mathsf{HDWal}}} := \left| \Pr[\mathbf{unl}^{\mathcal{A}}_{\mathsf{HDWal}} = 1] - \frac{1}{2} \right|.$$

On Forward Unlinkability The model of hierarchical wallets as defined in Definition 4.2 in Section 4 is stateless. In other words, each node in the tree maintains a fixed chaincode $\mathsf{ch}_{i,s}$ which is used as an input parameter for the child key derivation algorithms. If the (non-hardened) public key $\mathsf{pk}_{i,s}$ as well as the chaincode $\mathsf{ch}_{i,s}$ of a node are leaked (e.g., due to a hot wallet corruption of the node in the hot/cold wallet setting), then the adversary can as well compute the non-hardened keys in the entire sub-tree under $\mathsf{pk}_{i,s}$. Consequently, unlinkability of the sub-tree is lost. To enhance the unlinkability property, we can extend our model to a stateful variant where, each node maintains a state $\mathsf{St}^t_{i,s}$. On every child key derivation, the state of the node is refreshed to a new state $\mathsf{St}^{t+1}_{i,s}$. As a result of this modification, we can guarantee *forward unlinkability* for hierarchical wallets, which is similar to the standard notion of *forward security*. Precisely, on a hot wallet corruption, the adversary learns the *current state* $\mathsf{St}^t_{i,s}$ and the public key $\mathsf{pk}_{i,s}$ of a node. However, the existing children of this node were derived from earlier states $\mathsf{St}^{t'}_{i,s}$, for $t' < t$ - which are not known to the adversary. Thus it can no longer break the unlinkability of the existing child keys in the sub-tree under $\mathsf{pk}_{i,s}$. However, it would be able to link any future child keys derived from $\mathsf{pk}_{i,s}$.

4.3 Unforgeability

The notion of unforgeability for hierarchical deterministic wallets in the hot/cold wallet setting guarantees that an adversary cannot forge a signature of any uncorrupted node in the tree. In our model, non-hardened keys are always stored in hot/cold wallets, i.e., the secret keys are secured in the cold wallet storage, which cannot be corrupted by an adversary. Hardened keys, on the other hand, can be stored on any device and are thereby prone to corruption. Therefore, we allow an adversary to corrupt hardened secret keys, while non-hardened secret keys must remain uncorrupted.

In more detail, the unforgeability game proceeds as follows. The challenger generates a master key pair and a chaincode via the execution of $\mathsf{Setup}(1^\kappa)$. The adversary receives the master public key and obtains access to the oracles as described in subsection 4.1. Eventually, the adversary outputs a forgery, i.e., a message and a signature for a specific node in the tree. The adversary wins the game, if the signature is valid, the message has not been queried to the signing oracle `HDSign0` for this specific node before and the cold wallet of the node is uncorrupted. We note that a slightly weaker variant of unforgeability for hierarchical deterministic wallets is the notion of *one-per message unforgeability*, where the security game proceeds exactly as the game of the unforgeability notion with the difference that the adversary is allowed to query the `HDSign0` oracle only once for each message/address pair. We now give a detailed description of the unforgeability game $\mathbf{wufcma1}_{\mathsf{HDWal}}$.

Game $\mathbf{wufcma1}_{\mathsf{HDWal}}$:

- **Setup Phase:** The challenger computes $(\mathsf{ch}_{0,0}, \mathsf{msk}_{0,0}, \mathsf{mpk}_{0,0}) \leftarrow \mathsf{Setup}(1^n)$ and sends $\mathsf{ch}_{0,0}$ and $\mathsf{mpk}_{0,0}$ to $\mathcal{A}$.
- **Online Phase:** On input the security parameter, the adversary $\mathcal{A}$ is allowed to make queries to the oracles as explained in subsection 4.1.
- **Output Phase:** Eventually, $\mathcal{A}$ outputs a public key pk_{i^*,s^*}, a message m^*, an address $\mathbf{addr}_{i^*,s^*}$ and a signature σ^*. $\mathcal{A}$ wins if all of the following conditions hold,
  - $\mathsf{Verify}(pk_{i^*,s^*}, \sigma^*, m^*) = 1$
  - $m^* \notin \mathsf{SigList}[\mathbf{addr}_{i^*,s^*}]$
  - Either $\mathbf{addr}_{i^*,s^*}$ belongs to a non-hardened node or $\mathbf{addr}_{i^*,s^*}$ belongs to a hardened node and its secret key has not been corrupted, i.e., $\mathsf{SK}[\mathbf{addr}_{i^*,s^*}] = 0$.

We define the advantage of an adversary $\mathcal{A}$ in $\mathbf{wufcma1}_{\mathsf{HDWal}}$ as

$$\mathsf{Adv}^{\mathcal{A}}_{\mathbf{unl}_{\mathsf{HDWal}}} := \Pr[\mathbf{wufcma1}^{\mathcal{A}}_{\mathsf{HDWal}} = 1].$$

5 Generic Construction

In this section, we first show how to generically construct a hierarchical deterministic wallet scheme HDWal from a signature scheme with perfectly rerandomizable keys $\mathsf{RSig} = (\mathsf{RSig.Gen}, \mathsf{RSig.RandSK}, \mathsf{RSig.RandPK}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify})$. We denote the construction of HDWal with respect to a signature scheme with rerandomizable keys RSig by $\mathsf{HDWal}[\mathsf{RSig}]$. Our generic construction $\mathsf{HDWal}[\mathsf{RSig}]$ uses internally a hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{R} \times \{0,1\}^\kappa$. We detail our construction in Figure 5. Subsequently, we analyze the security of our generic construction by proving the unlinkability and the unforgeability properties of $\mathsf{HDWal}[\mathsf{RSig}]$. We defer the full proof for unlinkability of $\mathsf{HDWal}[\mathsf{RSig}]$ to Appendix A. In the following subsection, we present the theorem that states that $\mathsf{HDWal}[\mathsf{RSig}]$ satisfies $\mathbf{wufcma1}_{\mathsf{HDWal}}$ security with a loss in the security reduction. We then show that this loss is indeed unavoidable which means that our security reduction is optimal.

Algorithm $\mathsf{HDWal}[\mathsf{RSig}].\mathsf{Setup}(\mathsf{par})$
00 $\mathsf{ch}_{0,0} \xleftarrow{\$} \{0,1\}^\kappa$
01 $(\mathsf{msk}_{0,0}, \mathsf{mpk}_{0,0}) \xleftarrow{\$} \mathsf{RSig.Gen}(\mathsf{par})$
02 Return $(\mathsf{msk}_{0,0}, \mathsf{mpk}_{0,0}, \mathsf{ch}_{0,0})$

Algorithm $\mathsf{HDWal}[\mathsf{RSig}].\mathsf{Sign}(\mathsf{sk}_{i,s}, m)$
00 $\sigma \leftarrow \mathsf{RSig.Sign}(\mathsf{sk}_{i,s}, m)$
01 Return σ

Algorithm $\mathsf{HDWal}[\mathsf{RSig}].\mathsf{Verify}(\mathsf{pk}_{i,s}, \sigma, m)$
00 $0/1 \leftarrow \mathsf{RSig.Verify}(\mathsf{pk}_{i,s}, \sigma, m)$
01 Return $0/1$

Algorithm $\mathsf{HDWal}[\mathsf{RSig}].\mathsf{SKDer}_{\mathsf{H}}(\mathsf{sk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$
00 $(\omega, \mathsf{ch}_{i+1,t}) \leftarrow \mathsf{H}(\mathsf{sk}_{i,s}, \mathsf{ch}_{i,s}, \mathsf{e}_{i+1}^{s,t})$
01 $\mathsf{sk}_{i+1,t} \leftarrow \mathsf{RSig.RandSK}(\mathsf{sk}_{i,s}; \omega)$
02 $\mathbf{addr}_{i+1,t} \leftarrow \mathbf{addr}_{i,s} \parallel \mathsf{e}_{i+1}^{s,t}$
03 Return $(\mathsf{sk}_{i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t})$

Algorithm $\mathsf{HDWal}[\mathsf{RSig}].\mathsf{SKDer}_{\mathsf{NH}}(\mathsf{sk}_{i,s}, \mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$
00 $(\omega, \mathsf{ch}_{i+1,t}) \leftarrow \mathsf{H}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathsf{e}_{i+1}^{s,t})$
01 $\mathsf{sk}_{i+1,t} \leftarrow \mathsf{RSig.RandSK}(\mathsf{sk}_{i,s}; \omega)$
02 $\mathbf{addr}_{i+1,t} \leftarrow \mathbf{addr}_{i,s} \parallel \mathsf{e}_{i+1}^{s,t}$
03 Return $(\mathsf{sk}_{i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t})$

Algorithm $\mathsf{HDWal}[\mathsf{RSig}].\mathsf{PKDer}_{\mathsf{H}}(\mathsf{sk}_{i,s}, \mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$
00 $(\omega, \mathsf{ch}_{i+1,t}) \leftarrow \mathsf{H}(\mathsf{sk}_{i,s}, \mathsf{ch}_{i,s}, \mathsf{e}_{i+1}^{s,t})$
01 $\mathsf{pk}_{i+1,t} \leftarrow \mathsf{RSig.RandPK}(\mathsf{pk}_{i,s}; \omega)$
02 $\mathbf{addr}_{i+1,t} \leftarrow \mathbf{addr}_{i,s} \parallel \mathsf{e}_{i+1}^{s,t}$
03 Return $(\mathsf{pk}_{i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t})$

Algorithm $\mathsf{HDWal}[\mathsf{RSig}].\mathsf{PKDer}_{\mathsf{NH}}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$
00 $(\omega, \mathsf{ch}_{i+1,t}) \leftarrow \mathsf{H}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathsf{e}_{i+1}^{s,t})$
01 $\mathsf{pk}_{i+1,t} \leftarrow \mathsf{RSig.RandPK}(\mathsf{pk}_{i,s}; \omega)$
02 $\mathbf{addr}_{i+1,t} \leftarrow \mathbf{addr}_{i,s} \parallel \mathsf{e}_{i+1}^{s,t}$
03 Return $(\mathsf{pk}_{i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t})$

Figure 5: Generic construction of a hierarchical deterministic wallet scheme $\mathsf{HDWal}[\mathsf{RSig}]$ from a signature with perfectly rerandomizable keys RSig. $\mathsf{HDWal}[\mathsf{RSig}]$ is defined w.r.t. an address structure $(\mathcal{T}, \mathbf{Addr})$, where $\mathcal{T} = (h, n_{0,0}, \mathcal{N}, \mathcal{E})$, such that $\mathbf{addr}_{i,s} \in \mathbf{Addr}$ and $\mathsf{e}_i^{s,t} \in \mathcal{E}$ for $0 \le i \le h$ and $1 \le s, t \le |\mathcal{N}|$. We denote by $(\mathsf{pk}_{i,s}, \mathsf{sk}_{i,s})$ and $\mathsf{ch}_{i,s}$ the public/secret key pair and chaincode of the node with address $\mathbf{addr}_{i,s}$. We denote by H a hash function $\mathsf{H}\colon \{0,1\}^* \to \mathcal{R} \times \{0,1\}^\kappa$.

5.1 Unforgeability of Generic Construction

We now analyze the unforgeability property of our generic construction $\mathsf{HDWal}[\mathsf{RSig}]$ of a hierarchical wallet. We require the following properties from the underlying signature scheme RSig. RSig must satisfy (1) the definition of a signature scheme with rerandomizable keys as well as (2) a transitive property of the keys. We formally define the latter below.

Definition 5.1 (Transitive Rerandomization). Let $\mathsf{RSig} = (\mathsf{RSig.Gen}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify}, \mathsf{RSig.RandSK}, \mathsf{RSig.RandPK})$ be a signature scheme with perfectly rerandomizable keys. We say that RSig *transitively rerandomizes* if there exists an operation $\odot : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ s.t. for all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{RSig.Gen}(\mathsf{par})$ and all

$(\rho, \rho') \in \mathcal{R} \times \mathcal{R}$, the values $(\mathsf{sk}', \mathsf{pk}'), (\mathsf{sk}'', \mathsf{pk}''), \tilde{\rho}$ which are defined as

$$
\begin{aligned}
&(\mathsf{sk}', \mathsf{pk}') \leftarrow (\mathsf{RSig.RandSK}(\mathsf{sk}; \rho), \mathsf{RSig.RandPK}(\mathsf{pk}; \rho)) \\
&(\mathsf{sk}'', \mathsf{pk}'') \leftarrow (\mathsf{RSig.RandSK}(\mathsf{sk}'; \rho'), \mathsf{RSig.RandPK}(\mathsf{pk}'; \rho')), \\
&\tilde{\rho} = \rho \odot \rho' \text{ satisfy} \\
&(\mathsf{sk}'', \mathsf{pk}'') = (\mathsf{RSig.RandSK}(\mathsf{sk}; \tilde{\rho}), \mathsf{RSig.RandPK}(\mathsf{pk}; \tilde{\rho})).
\end{aligned}
$$

Definition 5.2 (Invertible Rerandomization). Let $\mathsf{RSig} = (\mathsf{RSig.Gen}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify}, \mathsf{RSig.RandSK}, \mathsf{RSig.RandPK})$ be a signature scheme with perfectly rerandomizable keys. We say that RSig has *invertible rerandomization* if there exist (efficient) algorithms RandSK^{-1} and RandPK^{-1} s.t. for all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{RSig.Gen}(\mathsf{par})$ and all $\rho \in \mathcal{R}$ it holds

$$
\begin{aligned}
\mathsf{sk} &= \mathsf{RandSK}^{-1}(\mathsf{RSig.RandSK}(\mathsf{sk}; \rho); \rho) \\
\mathsf{pk} &= \mathsf{RandPK}^{-1}(\mathsf{RSig.RandPK}(\mathsf{pk}; \rho); \rho)
\end{aligned}
$$

We note that the signature schemes with rerandomizable keys based on Schnorr [FKM+16], BLS [DFL19] and ECDSA (additive variant presented in Section 3 of this work and multiplicative variant presented in [DFL19]) all satisfy the properties of transitive rerandomization and invertible rerandmization as defined in Definitions 5.1, 5.2. For the Schnorr, BLS and additive ECDSA based schemes, the $\odot$ operation is a simple addition, while for the multiplicative ECDSA scheme it is a multiplication (modulo the group order p). Below we state our theorem for the one-per message unforgeability property of $\mathsf{HDWal}[\mathsf{RSig}]$.

Theorem 5.3 *Let* $\mathsf{HDWal}[\mathsf{RSig}]$ *be the construction defined in Figure 5, let* $\mathsf{H}\colon \{0,1\}^* \to \mathcal{R} \times \{0,1\}^\kappa$ *be a hash function modeled as a random oracle and let* RSig *be a signature scheme with rerandomizable keys that satisfies the property of transitive rerandomization and invertible rerandomization as in Definitions 5.1, 5.2. Let* $\mathcal{A}$ *be an adversary playing in the game* $\mathbf{wufcma1}^{\mathcal{A}}_{\mathsf{HDWal}[\mathsf{RSig}]}$*, then there exists an algorithm* $\mathcal{C}$ *running in roughly the same time as* $\mathcal{A}$*, and that makes as many queries to the oracle* `Rand` *in* **uf-cma-hrk1** *as* $\mathcal{A}$ *makes queries to* `NHChildO`/`HChildO` *such that*

$$
\mathsf{Adv}^{\mathcal{C}}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \frac{1}{4e(q_{sk}+1)} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{RSig}]}}.
$$

where q_{sk} *is the number of* `SKLeakO` *oracle queries from* $\mathcal{A}$*.*

We stated Theorem 5.3 w.r.t. the one-per message unforgeability notions of hierarchical deterministic wallet schemes and signature schemes with reradomizable keys, because these notions are sufficient in the setting of deterministic wallets. This is because wallets sign each unique transaction at most once. However, we note that we can likewise state and prove the above theorem with respect to the standard unforgeability notions, i.e., the notions that do not restrict the adversary to obtain at most one signature on a specific message.

In the following, we provide the full formal proof of Theorem 5.3.

Proof. The proof of Theorem 5.3 exhibits an adversary $\mathcal{C}$ who uses the adversary $\mathcal{A}$ who plays in game $\mathbf{wufcma1}^{\mathcal{A}}_{\mathsf{HDWal}[\mathsf{RSig}]}$ to win its own game $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{C}}_{\mathsf{RSig}}$. The main idea of our proof is that $\mathcal{C}$ guesses in advance which hardened nodes $\mathcal{A}$ might corrupt (i.e., calls the `SKLeakO` oracle on). In case the guess of $\mathcal{C}$ is wrong, $\mathcal{C}$ cannot answer all `SKLeakO` oracle queries from $\mathcal{A}$ and therefore has to abort. This leads to a polynomial loss in the number of `SKLeakO` oracle queries (i.e., q_{sk}) in $\mathcal{C}$'s advantage in its $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{C}}_{\mathsf{RSig}}$ game. We use Coron's technique as presented in [Cor02] to bound this loss.

We now provide the formal proof via a series of games $\boldsymbol{G}_0$ to $\boldsymbol{G}_6$.

Game $\boldsymbol{G}_0$: This is the regular $\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{RSig}]}$ game at the beginning of which a key pair $(\mathsf{pk}, \mathsf{sk})$ is generated and the adversary $\mathcal{A}$ is given as input pk and oracle access to the following oracles: `HChildO`, `NHChildO`, `HDSignO`, `CHLeakO` and `SKLeakO` oracles and a random oracle H. The random oracle H is internally programmed in a straight forward manner, by maintaining a list H. In particular, on input s, if $H[s] \neq \bot$, then return $H[s]$. Otherwise, sample a fresh randomness $\rho \xleftarrow{\$} \mathcal{R}$ and a fresh value as $\psi \xleftarrow{\$} \{0,1\}^\kappa$ and set $(\rho, \psi) =: H[s]$ and return $H[s]$. In addition, the game keeps a list R in which it

stores the randomness used to derive the keys at position s and level i at entry $R[i,s]$. We have that $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal[RSig]}}} = \Pr[\mathbf{wufcma1}^{\mathcal{A}}_{\mathsf{HDWal[RSig]}} = 1] = \Pr[\boldsymbol{G}^{\mathcal{A}}_0 = 1]$.

Game $\boldsymbol{G}_1$: Upon generating the key pair $(\mathsf{pk}, \mathsf{sk})$, the game chooses a fresh chaincode $\mathsf{ch}_{0,0} \xleftarrow{\$} \{0,1\}^\kappa$ and fresh randomness $\rho \xleftarrow{\$} \mathcal{R}$. Then it derives the root public key for the $\mathbf{wufcma1}_{\mathsf{HDWal[RSig]}}$ game as $\mathsf{mpk}_{0,0} \xleftarrow{\$} \mathsf{RSig.RandPK}(\mathsf{pk}; \rho)$, stores ρ in a list as $R[0,0] = \rho$. The game sends $\mathsf{ch}_{0,0}$ and $\mathsf{mpk}_{0,0}$ to $\mathcal{A}$.

Since the randomness ρ is chosen uniformly at random from $\mathcal{R}$, the rerandomizability of keys property of the signature scheme RSig holds. This implies that the distributions of $(\cdot, \mathsf{mpk}_{0,0})$ and $(\cdot, \mathsf{mpk}'_{0,0}) \xleftarrow{\$} \mathsf{RSig.Gen}(\mathsf{par})$ are identical. Therefore, it holds that $\Pr[\boldsymbol{G}^{\mathcal{A}}_1 = 1] = \Pr[\boldsymbol{G}^{\mathcal{A}}_0 = 1]$.

Game $\boldsymbol{G}_2$: This game behaves like $\boldsymbol{G}_1$ with a modification in the `NHChildO` oracle. Upon an oracle query on input $(\mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$ the `NHChildO` oracle executes $\mathsf{PKDer}_{\mathsf{NH}}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$ and creates the public key $\mathsf{pk}_{\mathsf{nh},i+1,t}$ at level $i+1$ and position t as $\mathsf{pk}_{\mathsf{nh},i+1,t} \leftarrow \mathsf{RandPK}(\mathsf{pk}; \omega \odot R[i,s])$, i.e., the public key $\mathsf{pk}_{\mathsf{nh},i+1,t}$ is derived directly from pk with randomness $\omega \odot R[i,s]$, where $(\omega, \cdot) \leftarrow \mathsf{H}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathsf{e}^{s,t}_{i+1})$. The game then sets the list $R[i+1,t] = \omega \odot R[i,s]$. If any of the values $(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, R[i,s])$ is not defined yet, the game recursively derives the path from the root node up to $(\mathsf{pk}_{i,s}, \mathbf{addr}_{i,s})$ and updates the list up to $R[i,s]$.

Note that $\mathsf{RandPK}(\mathsf{pk}; \omega \odot R[i,s])$ and $\mathsf{RandPK}(\mathsf{pk}_{\mathsf{nh},i,s}; \omega)$ derive the same key $\mathsf{pk}_{\mathsf{nh},i+1,t}$, due to the transitive property of rerandomizable keys. Since ω and $R[i,s]$ are uniformly at random from $\mathcal{R}$, we have that $\Pr[\boldsymbol{G}^{\mathcal{A}}_2 = 1] = \Pr[\boldsymbol{G}^{\mathcal{A}}_1 = 1]$.

Game $\boldsymbol{G}_3$: This game proceeds similarly to the previous game with a modification in the random oracle. The game aborts upon the adversary querying the random oracle on input $(\mathsf{sk}_{\mathsf{nh},i,s}, \cdot, \cdot)$ where $\mathsf{sk}_{\mathsf{nh},i,s}$ is either a non-hardened secret key that corresponds to a public key $\mathsf{pk}_{\mathsf{nh},i,s}$ previously output by the `NHChildO` oracle or $\mathsf{sk}_{\mathsf{nh},i,s}$ is the master secret key $\mathsf{msk}_{0,0}$ corresponding to $\mathsf{mpk}_{0,0}$.

Claim 5.4 Let ϵ be the probability that game $\boldsymbol{G}_3$ aborts during a random oracle query. Then there exists an algorithm $\mathcal{C}_1$ playing in game $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ such that $\mathsf{Adv}^{\mathcal{C}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \epsilon$.

Proof. We prove this claim by providing a reduction to the **uf-cma-hrk1** security of RSig. More concretely, we show that there exists an algorithm $\mathcal{C}_1$ with $\mathsf{Adv}^{\mathcal{C}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \epsilon$ assuming $\mathcal{C}_1$ has access to an adversary $\mathcal{A}$ that causes $\boldsymbol{G}_3$ to abort with probability ϵ. Initially, $\mathcal{C}_1$ receives as input a public key pk from the $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ game and chooses at random a chaincode $\mathsf{ch} \xleftarrow{\$} \{0,1\}^\kappa$. From pk and ch, $\mathcal{C}_1$ can honestly simulate the `NHChildO` and `CHLeakO` oracles to $\mathcal{A}$. The simulation of the random oracle H works as described in $\boldsymbol{G}_3$ with the exception that instead of sampling the randomness $\rho \xleftarrow{\$} \mathcal{R}$ uniformly at random from $\mathcal{R}$, $\mathcal{C}_1$ calls the `Rand` oracle in game $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ to obtain the randomness ρ. A query from $\mathcal{A}$ to the `HDSignO` oracle on input $(m, \mathbf{addr}_{\cdot,\cdot})$ is forwarded to the `RSign` oracle on input m and the randomness corresponding to $\mathbf{addr}_{\cdot,\cdot}$ of the $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{C}_1}_{\mathsf{RSig}}$ game. For a `HChildO` oracle query on input $(\mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$, $\mathcal{C}_1$ chooses a fresh key pair (independently of pk) $(\mathsf{sk}', \mathsf{pk}') \xleftarrow{\$} \mathsf{RSig.Gen}(\mathsf{par})$, assigns $(\mathsf{sk}_{\mathsf{h},i+1,t}, \mathsf{pk}_{\mathsf{h},i+1,t}) :- (\mathsf{sk}', \mathsf{pk}')$ and returns $\mathsf{pk}_{\mathsf{h},i+1,t}$. The `SKLeakO` oracle is then simulated by returning $\mathsf{sk}_{\mathsf{h},i+1,t}$ on input $\mathbf{addr}_{i+1,t}$. The simulation of the `HChildO` and `HDSignO` oracles cannot be distinguished by $\mathcal{A}$ from the oracles in $\boldsymbol{G}_3$ due to the rerandomizability of keys property of RSig. The only way in which $\mathcal{A}$ could detect the difference between $\boldsymbol{G}_3$ and the reduction provided by $\mathcal{C}_1$ would be if the following event occurs. $\mathcal{A}$ makes a random oracle query of the form $(\mathsf{sk}_{\mathsf{nh},i,s}, \cdot, \cdot)$ where $\mathsf{sk}_{\mathsf{nh},i,s}$ is either a non-hardened secret key that corresponds to a public key $\mathsf{pk}_{\mathsf{nh},i,s}$ previously output by the `NHChildO` oracle or $\mathsf{sk}_{\mathsf{nh},i,s}$ is the secret key corresponding to pk (if $\mathsf{sk}_{\mathsf{nh},i,s}$ belongs to a public key $\mathsf{pk}_{\mathsf{nh},i,s}$ can be efficiently checked via the function $\mathsf{ToPubKey}(\mathsf{sk}_{\mathsf{nh},i,s})$). By Claim 5.4, this event happens with probability ϵ. However, when this event occurs, $\mathcal{C}_1$ learns the secret key $\mathsf{sk}_{\mathsf{nh},i,s}$ which it can use to compute the secret key sk of the $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ game. This is due to the transitivity and invertible rerandomization property of RSig. $\mathcal{C}_1$ can then use sk to create a valid forgery in the $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{C}_1}_{\mathsf{RSig}}$ game. Therefore, we have that $\mathsf{Adv}^{\mathcal{C}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \epsilon$. ∎

It follows that $\Pr[\boldsymbol{G}^{\mathcal{A}}_2] \leq \Pr[\boldsymbol{G}^{\mathcal{A}}_3] + \epsilon$.

Game $\boldsymbol{G}_4$: This game works like the previous game with a modification to the `HChildO` oracle which works as follows. Let q_{sk} be the number of hardened nodes that $\mathcal{A}$ corrupts via the `SKLeakO` oracle. Upon

$\mathcal{A}$ querying the HChildO oracle, with probability $\frac{1}{q_{\mathsf{sk}}+1}$, the address of this node is added to a list L. Let Bad define the event that a node corresponding to an address in L is corrupted.

Since the change in this game is only syntactical, $\mathcal{A}$'s winning probability is not affected by whether Bad occurs. It follows that $\Pr[\boldsymbol{G}_3^{\mathcal{A}}] = \Pr[\boldsymbol{G}_4^{\mathcal{A}}]$.

Game $\boldsymbol{G}_5$: This game works like the previous game with the only difference that $\boldsymbol{G}_5$ aborts in case event Bad occurs.

Lemma 5.5 $\Pr[\boldsymbol{G}_4^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G}_5^{\mathcal{A}} = 1] \cdot e$.

Proof. $\mathcal{A}$ can distinguish $\boldsymbol{G}_5$ from the previous game if the game aborts i.e., when the event Bad happens. This event happens for each SKLeakO query, independently, with probability $\frac{1}{q_{\mathsf{sk}}+1}$. With probability $(1 - \frac{1}{q_{\mathsf{sk}}+1})$, a SKLeakO oracle query does not lead to an abort. Hence, the overall probability with which the game does not abort on any SKLeakO oracle query can be lower bounded by $(1 - \frac{1}{q_{\mathsf{sk}}+1})^{q_{\mathsf{sk}}} \geq e^{-1}$, i.e., Bad occurs with probability at most $1 - e^{-1}$. As we have argued that Bad occurs in $\boldsymbol{G}_4$ independently of the event $\boldsymbol{G}_4 = 1$, we have that $\Pr[\boldsymbol{G}_4^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G}_5^{\mathcal{A}} = 1] \cdot 1/\Pr[\neg\mathsf{Bad}] \leq \Pr[\boldsymbol{G}_5^{\mathcal{A}} = 1] \cdot e$. ■

Game $\boldsymbol{G}_6$: This game works like the previous game with a modification to the HChildO oracle which works as follows. For the nodes that are chosen to be added to the list L, the game derives the public key of that node as a public key of a non-hardened node. The rest of the hardened nodes are generated as $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{RSig.Gen}(\mathsf{par})$ and assigned $(\mathsf{sk}_{\mathsf{h},i+1,t}, \mathsf{pk}_{\mathsf{h},i+1,t}) := (\mathsf{sk}, \mathsf{pk})$.

Lemma 5.6 $\Pr[\boldsymbol{G}_5^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1]$.

Proof. $\mathcal{A}$ can distinguish $\boldsymbol{G}_6$ from the previous game if it corrupts a hardened node which is simulated as a non-hardened node i.e., one of the nodes in the list L. The only other way for $\mathcal{A}$ to distinguish these two games would be if $\mathcal{A}$ was able to query the random oracle on input the secret key of a non-hardened node as this would allow to recursively compute the secret key of the corresponding child hardened node. This case is, however, has already been excluded in $\boldsymbol{G_3}$. As explained in game $\boldsymbol{G}_5$, upon $\mathcal{A}$ making a corruption query for a node in list L, the game aborts. Therefore, the adversary cannot distinguish this game from the previous game. ■

By the transition from game $\boldsymbol{G}_0$ to game $\boldsymbol{G}_6$, we get that

$$\mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal[RSig]}}} = \Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1] \leq \big(\Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] \cdot e\big) + \epsilon$$
$$\text{or, } \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] \geq \frac{1}{e} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal[RSig]}}} - \frac{1}{e} \cdot \epsilon$$

Reduction to uf-cma-hrk security. Having shown that the transition from game $\mathbf{wufcma1}^{\mathcal{A}}_{\mathsf{HDWal[RSig]}}$ to the game $\boldsymbol{G}_6$ is indistinguishable, it remains to show that there exists a challenger $\mathcal{C}_2$ that simulates $\boldsymbol{G}_5$ and uses $\mathcal{A}$ to win its own game $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$. The challenger code is same as $\boldsymbol{G}_6$ with the following changes: (1) The sampling of $\rho \xleftarrow{\$} \mathcal{R}$ within the programming of H is replaced by a call to the oracle Rand (2) pk is replaced by the challenge public key $\mathsf{pk}_{\mathcal{C}_2}$ from the underlying game $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{C}_2}_{\mathsf{RSig}}$. Since the above changes are trivially indistinguishable to $\mathcal{A}$, we move on to analyze $\mathcal{C}_2$'s probability to win the $\mathbf{uf\text{-}cma\text{-}hrk}^{\mathcal{C}_2}_{\mathsf{RSig}}$ game using the forgery of $\mathcal{A}$. There are two possibilities for $\mathcal{A}$; either to output a forgery for a non-hardened node or for a hardened node. We analyze each case separately and show that for both cases our simulator can win its game with non-negligible probability.

- Adversary outputs a forgery for a non-hardened node: If the adversary provides a forgery for a non-hardened node, $\mathcal{C}_2$ can always use this forgery to win the $\mathbf{uf\text{-}cma\text{-}hrk}^{\mathcal{C}_2}_{\mathsf{RSig}}$ game. Therefore, the overall probability of $\mathcal{C}_2$ winning the game in case of $\mathcal{A}$ generating a forgery for a non-hardened node is:

$$\mathsf{Adv}^{\mathcal{C}_2}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] \geq \frac{1}{e} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal[RSig]}}} - \frac{\epsilon}{e}$$

- Adversary outputs a forgery for a hardened node: We now compute the probability that the game aborts in case the adversary generates a forgery for a hardened node.

  Let i^* be the index of the hardened node for which the adversary outputs a forgery. In this case $\mathcal{C}_2$ needs to abort if i^* was sampled randomly. Recall, the probability that i^* has been sampled at random is $1 - \frac{1}{q_{\mathsf{sk}}+1}$. Therefore, the overall probability of the simulator winning the game in case of $\mathcal{A}$ generating a forgery for a hardened node is:

$$\mathsf{Adv}^{\mathcal{C}_2}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \Pr[\boldsymbol{G}^{\mathcal{A}}_6 = 1] \cdot \frac{1}{q_{\mathsf{sk}}+1} \geq \left(\frac{1}{e} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{RSig}]}} - \frac{\epsilon}{e}\right) \cdot \frac{1}{q_{\mathsf{sk}}+1}$$

We can now compose a challenger $\mathcal{C}$ from the challengers $\mathcal{C}_1$ of Claim 5.4 and $\mathcal{C}_2$, such that $\mathcal{C}$ uses adversary $\mathcal{A}$ to win in its game $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{C}}_{\mathsf{RSig}}$. $\mathcal{C}$ executes either of $\mathcal{C}_1$ and $\mathcal{C}_2$ with probability $\frac{1}{2}$. In order to compute $\mathcal{C}$'s advantage $\mathsf{Adv}^{\mathcal{C}}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}}$, we distinguish the following two cases:

- Case $\epsilon \geq \frac{1}{2}\mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}}}$: In this case, we have by claim 5.4 that

$$\mathsf{Adv}^{\mathcal{C}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \epsilon \geq \frac{1}{2}\mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}}}.$$

  Therefore we can lower bound $\mathcal{C}$'s advantage by

$$\mathsf{Adv}^{\mathcal{C}}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \frac{1}{2}\mathsf{Adv}^{\mathcal{C}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \frac{\mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}}}}{4}.$$

- Case $\epsilon < \frac{1}{2}\mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}}}$: In this case, we can lower bound $\mathcal{C}_2$'s advantage by

$$\begin{aligned}
\mathsf{Adv}^{\mathcal{C}_2}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} &\geq \left(\mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{RSig}]}} \cdot \frac{1}{e} - \frac{\epsilon}{e}\right) \cdot \frac{1}{q_{\mathsf{sk}}+1} \\
&\geq \left(\mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{RSig}]}} \cdot \frac{1}{e} - \frac{1}{2e} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{RSig}]}}\right) \cdot \frac{1}{q_{\mathsf{sk}}+1} \\
&= \frac{1}{2e(q_{sk}+1)} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{RSig}]}}
\end{aligned}$$

  Hence, $\mathcal{C}$'s overall advantage can be lower bounded by

$$\mathsf{Adv}^{\mathcal{C}}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \min\left(\frac{1}{2}\mathsf{Adv}^{\mathcal{C}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}}, \frac{1}{2}\mathsf{Adv}^{\mathcal{C}_2}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}}\right) \geq \frac{1}{4e(q_{sk}+1)} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{RSig}]}}.$$

■

The proof of Theorem 5.3 incurs a polynomial loss in the number of `SKLeak0` oracle queries (i.e., q_{sk}) in $\mathcal{C}$'s advantage in its $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{C}}_{\mathsf{RSig}}$ game. Interestingly, the following theorem states that this loss is inherent and that, in fact, there does not exist a tighter security reduction. In Appendix B, we recall the security notion of *unforgeability under rerandomized keys* $\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}$ for a signature scheme with rerandomizable keys RSig as introduced in [FKM+16] and prove Theorem 5.7. Below, we denote as $\mathcal{A}_1^{\mathcal{A}_2}$ that $\mathcal{A}_1$ has black-box access to $\mathcal{A}_2$. In particular, it does not rewind $\mathcal{A}_2$.

Theorem 5.7 *Let* HDWal *be an algorithm such that for any signature scheme with rerandomizable keys* RSig, $\mathsf{HDWal}^{\mathsf{RSig}}$ *is a hierarchical deterministic wallet scheme. Moreover, suppose that there is a reduction* $\mathcal{R}$ *such that for every signature scheme with rerandomizable keys* RSig *and every adversary* $\mathcal{A}$ *running in time* $t_{\mathcal{A}}$ *with* $\epsilon_{\mathcal{A}} = \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}}$, *it holds that* $\mathsf{Adv}^{\mathcal{R}^{\mathcal{A}}}_{\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}} \geq \epsilon_{\mathcal{R}}$ *and* $\mathcal{R}^{\mathcal{A}}$ *runs in time* $t_{\mathcal{R}}$. *Then there exists an algorithm* $\mathcal{M}$ *running in time* $t_{\mathcal{M}} \leq 2 \cdot t_{\mathcal{R}}$ *s.t.* $\mathsf{Adv}^{\mathcal{M}}_{\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}} \geq \epsilon_{\mathcal{R}} - \epsilon_{\mathcal{A}} \cdot \frac{2\exp(-1)}{q_{\mathsf{sk}}}$.

Theorem 5.7 implies that if there exists a reduction from $\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}$ to $\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}$ for a signature scheme with rerandomizable keys RSig s.t. the reduction loses less than a factor proportional to q_{sk}, then there exists an efficient algorithm $\mathcal{M}$ that can break the $\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}$ security. We formulate and prove this result w.r.t. a reduction from the strongest possible security notion of signature schemes with rerandomizable keys (i.e., **uf-cma-rk**) to the restricted notion of one-per message wallet unforgeability (i.e., $\mathbf{wufcma1}_{\mathsf{HDWal}}$). Clearly, this implies that the result from Theorem 5.7 also holds for the weaker notion of **uf-cma-hrk1** for signature schemes with rerandomizable keys which we use in Theorem 5.3. We note that Theorem 5.7 can likewise be stated and proved with respect to the standard unforgeability notion of hierarchical deterministic wallet schemes, i.e., the notion that does not restrict the adversary to obtain at most one signature on a specific message.

6 Discussion

On Security Parameters We instantiate our generic hierarchical deterministic wallet construction $\mathsf{HDWal}[\mathsf{RSig}]$ with two schemes, namely $\mathsf{REC}[\mathsf{H}_1]$ (Figure 7) and $\mathsf{REC}'[\mathsf{H}_1]$ (Figure 6). Note that $\mathsf{HDWal}[\mathsf{REC}[\mathsf{H}_1]]$ corresponds to the BIP32 wallet, while $\mathsf{HDWal}[\mathsf{REC}'[\mathsf{H}_1]]$ is instantiated from the multiplicatively rerandomized construction $\mathsf{REC}'[\mathsf{H}_1]$ from [DFL19], we will refer to it as BIP32-m.

First, let us recall, how to compute the bit security level of a scheme. A hierarchical wallet scheme HDWal is said to have a bit security level of κ bits, if any algorithm $\mathcal{A}$ with running time t and advantage ϵ in $\mathbf{wufcma1}_{\mathsf{HDWal}}$ takes *expected* running time $\frac{t}{\epsilon} \geq 2^{\kappa}$ to break the scheme for the first time. (The security level for a conventional signature scheme is defined analogously). From our Theorems C.1, C.2, we compute the bit security level of our schemes, considering an algorithm $\mathcal{A}$ with parameters t', ϵ' (in game $\mathbf{wufcma1}_{\mathsf{HDWal}}$), where $t' \approx t$ and $\epsilon' = \epsilon \cdot Q$ for some $Q \geq 1$ and where t, ϵ denote the runtime and advantage of the related forger $\mathcal{C}$ in game $\mathbf{uf\text{-}cma1}_{\mathsf{EC}}$. By assumption, EC satisfies $\kappa = 128$ bits of security, hence $\frac{t}{\epsilon} \geq 2^{128}$. Thus, we obtain $\frac{t'}{\epsilon'} = \frac{t}{\epsilon \cdot Q} \geq \frac{2^{128}}{Q} = 2^{\kappa - \log Q}$. Our results are reported in Table 1, where we took an estimate of the practical parameters as follows: the total number of keys is $q = 2^{20}$, the number of q_{sk} of secret keys leaked is roughly 1% of the total number of keys q, i.e., $q_{\mathsf{sk}} \approx 2^{14}$.

Scheme	Theorem Ref.	Bit Security with $\kappa = 128$
BIP32	Thm C.1	$\log(Q) = \log(q \cdot 4e \cdot q_{\mathsf{sk}}) \approx 37, \kappa - \log(Q) = 91$
BIP32-m	Thm C.2	$\log(Q) = \log(4e \cdot q_{\mathsf{sk}}) \approx 17, \kappa - \log(Q) = 111$

Table 1: Bit Security Level of BIP32 and BIP32-m, relying on **uf-cma1** of $\mathsf{EC}[\mathsf{H}_0]$

On BIP32 Parameters. Our construction of $\mathsf{HDWal}[\mathsf{REC}[\mathsf{H}_1]]$ gives us the BIP32 construction as specified in [Wik18]. Here we list the exact parameters used in BIP32 and minor differences of BIP32 with our construction $\mathsf{HDWal}[\mathsf{REC}[\mathsf{H}_1]]$.

- Each node can derive at most 2^{32} children nodes.
- $\mathsf{e}^{\cdot,\cdot}$ is chosen from $\{0,1\}^{32}$, which allows each non-hardened node to generate 2^{31} non-hardened and 2^{31} hardened child keys.
- A child key is derived as a hardened or a non-hardened node based on whether $\mathsf{e}^{\cdot,\cdot} \geq 2^{31}$ or $\leq 2^{31}$ respectively. However, this is syntactical, and does not affect our security analysis.
- Although at each level, the total number of derived keys can be at most $(2^{32}) \cdot p$, where p is the number of parent nodes in the immediate upper level, we do not imagine that all of these keys are derived at every level. As can be seen, this would already exceed our parameter $q = 2^{20}$, as selected above.
- The chaincode $\mathsf{ch}_{\cdot,\cdot}$ is chosen from $\{0,1\}^{256}$.
- The input parameter $\mathbf{addr}_{\cdot,\cdot}$ to the key derivation algorithms is set to an empty string. We use this parameter to indicate the position in the tree, at which the child key is derived and to ensure that the actual BIP32 derivation algorithms are called on the proper inputs for this position.
- The input parameter $\mathbf{addr}_{\cdot,\cdot}$ to the key derivation algorithms is set to an empty string λ. Let us briefly explain this syntactical difference. In our Definition 4.2, $\mathbf{addr}_{\cdot,\cdot} \neq \lambda$ is provided as input. This

makes the user aware of the position in the tree, at which the child key is derived and makes sure that the actual BIP32 derivation algorithms are called on the proper inputs for this position in the tree.

Open Questions Finally, let us mention some interesting open questions that can be answered in future works:

- Is it possible to remove the one per-message restriction and prove the security of the additively rerandomizable ECDSA scheme in the **uf-cma-hrk** notion? Additionally, is there a tight reduction to **uf-cma-hrk**?
- Can we improve the tightness of **uf-cma1** security [FKP17] of ECDSA from the semi-logarithm problem?

Acknowledgments

The authors are grateful to the anonymous reviewers for their valuable comments. 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

[ABFF09] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3), January 2009. (Cited on page 5.)

[ADE+20] Nabil Alkeilani Alkadri, Poulami Das, Andreas Erwig, Sebastian Faust, Juliane Krämer, Siavash Riahi, and Patrick Struck. Deterministic wallets in a quantum world. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20: 27th Conference on Computer and Communications Security*, pages 1017–1031, Virtual Event, USA, November 9–13, 2020. ACM Press. (Cited on page 5.)

[AGKK19] Myrto Arapinis, Andriana Gkaniatsou, Dimitris Karakostas, and Aggelos Kiayias. A formal treatment of hardware wallets. In Ian Goldberg and Tyler Moore, editors, *FC 2019: 23rd International Conference on Financial Cryptography and Data Security*, volume 11598 of *Lecture Notes in Computer Science*, pages 426–445, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Heidelberg, Germany. (Cited on page 5.)

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany. (Cited on page 5.)

[BH19] Joachim Breitner and Nadia Heninger. Biased nonce sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies. In Ian Goldberg and Tyler Moore, editors, *FC 2019: 23rd International Conference on Financial Cryptography and Data Security*, volume 11598 of *Lecture Notes in Computer Science*, pages 3–20, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Heidelberg, Germany. (Cited on page 5.)

[Bit18] BitcoinExchangeGuide. CipherTrace Releases Report Exposing Close to $1 Billion Stolen in Crypto Hacks During 2018. https://coinexchangeguide.com/ciphertrace-releases-report-exposing-close-to-1-billion-stolen-in_-crypto-hacks-during-2018/, 2018. (Cited on page 1.)

[Blo18] Bloomberg. How to Steal $500 Million in Cryptocurrency. http://fortune.com/2018/01/31/coincheck-hack-how/, 2018. (Cited on page 1.)

[BR18] Michael Brengel and Christian Rossow. Identifying key leakage of bitcoin users. In Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis, editors, *Research in Attacks, Intrusions, and Defenses*, pages 623–643, Cham, 2018. Springer International Publishing. (Cited on page 5.)

[But13] Vitalik Buterin. Deterministic Wallets, Their Advantages and their Understated Flaws. https://bitcoinmagazine.com/articles/deterministic-wallets-advantages-flaw-1385450276/, 2013. (Cited on page 2.)

[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 5.)

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany. (Cited on page 5, 21, 31, 32.)

[DFL19] Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 651–668. ACM Press, November 11–15, 2019. (Cited on page 2, 3, 4, 5, 6, 7, 8, 14, 21, 25, 33.)

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pages 980–997, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press. (Cited on page 5.)

[Ele13] Version bytes for BIP32 extended public and private keys. https://electrum.readthedocs.io/en/latest/xpub_version_bytes.html, 2013. (Cited on page 1.)

[FF13] Marc Fischlin and Nils Fleischhacker. Limitations of the meta-reduction technique: The case of Schnorr signatures. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 444–460, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. (Cited on page 5.)

[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. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 9614 of *Lecture Notes in Computer Science*, pages 301–330, Taipei, Taiwan, March 6–9, 2016. Springer, Heidelberg, Germany. (Cited on page 3, 5, 21, 24, 30.)

[FKP16] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the provable security of (EC)DSA signatures. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1651–1662, Vienna, Austria, October 24–28, 2016. ACM Press. (Cited on page 5.)

[FKP17] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the one-per-message unforgeability of (ec)dsa and its variants. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 519–534, Cham, 2017. Springer International Publishing. (Cited on page 4, 7, 26.)

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16: 14th International Conference on Applied Cryptography and Network Security*, volume 9696 of *Lecture Notes in Computer Science*, pages 156–174, Guildford, UK, June 19–22, 2016. Springer, Heidelberg, Germany. (Cited on page 5.)

[GS15] Gus Gutoski and Douglas Stebila. Hierarchical deterministic bitcoin wallets that tolerate key leakage. In Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015: 19th International Conference on Financial Cryptography and Data Security*, volume 8975 of *Lecture Notes in Computer Science*, pages 497–504, San Juan, Puerto Rico, January 26–30, 2015. Springer, Heidelberg, Germany. (Cited on page 5.)

[KK18] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. *Journal of Cryptology*, 31(1):276–306, January 2018. (Cited on page 31.)

[KMOS19] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. Refresh when you wake up: Proactive threshold wallets with offline devices. Cryptology ePrint Archive, Report 2019/1328, 2019. https://eprint.iacr.org/2019/1328. (Cited on page 5.)

[KMP16] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 33–61, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. (Cited on page 5.)

[Led14] Ledger Support,Ledger Nano OS. https://support.ledger.com/hc/en-us/articles/115005297709-Export-your-accounts, 2014. (Cited on page 1.)

[LFA20] Adriano Di Luzio, Danilo Francati, and Giuseppe Ateniese. Arcula: A secure hierarchical deterministic wallet for multi-asset blockchains. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20: 19th International Conference on Cryptology and Network Security*, volume 12579 of *Lecture Notes in Computer Science*, pages 323–343, Vienna, Austria, December 14–16, 2020. Springer, Heidelberg, Germany. (Cited on page 5.)

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1837–1854, Toronto, ON, Canada, October 15–19, 2018. ACM Press. (Cited on page 5.)

[MPs19] Antonio Marcedone, Rafael Pass, and abhi shelat. Minimizing trust in hardware wallets with two factor signatures. In Ian Goldberg and Tyler Moore, editors, *FC 2019: 23rd International Conference on Financial Cryptography and Data Security*, volume 11598 of *Lecture Notes in Computer Science*, pages 407–425, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Heidelberg, Germany. (Cited on page 5.)

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany. (Cited on page 5.)

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. https://ia.cr/2004/332. (Cited on page 6.)

[Ske18] Rhys Skellern. Cryptocurrency Hacks: More Than $2b USD lost between 2011-2018. https://medium.com/ecomi/cryptocurrency-hacks-more-than-2b-usd-lost-between-2011-2018_-67054b342219, 2018. (Cited on page 1.)

[Tre14] Trezor Wiki,Cryptocurrency standards,Hierachical deterministic wallets. https://wiki.trezor.io/Cryptocurrency_standards, 2014. (Cited on page 1.)

[TVR16] Mathieu Turuani, Thomas Voegtlin, and Michaël Rusinowitch. Automated verification of electrum wallet. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 27–42, Christ Church, Barbados, February 26, 2016. Springer, Heidelberg, Germany. (Cited on page 5.)

[Wik18] Bitcoin Wiki. BIP32 proposal. https://en.bitcoin.it/wiki/BIP_0032, 2018. (Cited on page 1, 13, 25.)

[ZCC+15] Zongyang Zhang, Yu Chen, Sherman S. M. Chow, Goichiro Hanaoka, Zhenfu Cao, and Yunlei Zhao. Black-box separations of hash-and-sign signatures in the non-programmable random oracle model. In Man Ho Au and Atsuko Miyaji, editors, *ProvSec 2015: 9th International Conference on Provable Security*, volume 9451 of *Lecture Notes in Computer Science*, pages 435–454, Kanazawa, Japan, November 24–26, 2015. Springer, Heidelberg, Germany. (Cited on page 5.)

A Unlinkability Proof of Generic Construction

Theorem A.1 *Let* $\mathsf{HDWal}[\mathsf{RSig}]$ *be the construction defined in Figure 5. Then for any adversary* $\mathcal{A}$ *playing in game* $\mathbf{unl}^{\mathcal{A}}_{\mathsf{HDWal}[\mathsf{RSig}]}$ *there exists an adversary* $\mathcal{A}_1$ *that plays in the game* $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ *such that*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathbf{unl}_{\mathsf{HDWal}[\mathsf{RSig}]}} \leq \frac{q_{\mathsf{H}}(q_C+1)}{2^{\kappa}} + \mathsf{Adv}^{\mathcal{A}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}}$$

where q_{H} *and* q_C *are the number of random oracle and child creation queries from* $\mathcal{A}$*, respectively.*

Proof. Consider the $\mathbf{unl}^{\mathcal{A}}_{\mathsf{HDWal}[\mathsf{RSig}]}$ game for an adversary $\mathcal{A}$. In the beginning, the challenger generates a fresh master key pair and chaincode

$$(\mathsf{msk}_{\mathsf{nh},0,0}, \mathsf{mpk}_{\mathsf{nh},0,0}, \mathsf{ch}_{0,0}) \leftarrow \mathsf{HDWal}[\mathsf{RSig}].\mathsf{Setup}(\mathsf{par})$$

and runs $\mathcal{A}$ on inputs the security parameter and the master public key $\mathsf{mpk}_{\mathsf{nh},0,0}$. During the output phase of the $\mathbf{unl}^{\mathcal{A}}_{\mathsf{HDWal}[\mathsf{RSig}]}$ game, $\mathcal{A}$ outputs a tuple $(\mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1}, c)$, where $\mathsf{e}^{s,t}_{i+1}$ is the edge from the node with address $\mathbf{addr}_{i,s}$ to the challenge node with address $\mathbf{addr}_{i+1,t}$ and c indicates if $\mathbf{addr}_{i+1,t}$ is a hardened or non-hardened node. In case $\mathbf{addr}_{i,s}$ is a hardened node, the game aborts and hence we have that the adversary's advantage is 0, i.e., $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{unl}_{\mathsf{HDWal}[\mathsf{RSig}]}} = 0$. Likewise, if $\mathcal{A}$ has previously queried the `CHLeakO` oracle on address $\mathbf{addr}_{i,s}$ or any of its prefix addresses, i.e., $\mathsf{CH}[\mathbf{addr}^{j}_{i,s}] = 1$ for any $j < i$, and if the challenge node is non-hardened (i.e., $c = \mathsf{nh}$) then the game aborts and we have that $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{unl}_{\mathsf{HDWal}[\mathsf{RSig}]}} = 0$.

Let $\mathsf{pk}_{\mathsf{nh},i+1,t}$ and $\mathsf{pk}_{\mathsf{nh},1,t}$ denote the challenge public keys in case $\mathcal{A}$ challenges a non-hardened node (i.e., $c = \mathsf{nh}$) with address $\mathbf{addr}_{i+1,t}$. Further, let $(\mathsf{pk}_{j,\cdot}, \mathsf{ch}_{j,\cdot})$ for $1 \leq j \leq i$ denote the public key and chaincode pair of all nodes in the prefix address of $\mathbf{addr}_{i+1,t}$. Recall that the non-hardened public keys are computed as follows:

$$\begin{aligned}(\omega, \mathsf{ch}_{j+1,t}) &\leftarrow \mathsf{H}(\mathsf{pk}_{j,s}, \mathsf{ch}_{j,s}, \mathsf{e}^{s,t}_{j+1}),\\ \mathsf{pk}_{j+1,t} &\leftarrow \mathsf{RSig.RandPK}(\mathsf{pk}_{j,s}; \omega)\end{aligned}$$

According to the (perfect) rerandomizability of keys property (cf. Def 2.2) the public keys derived via the $\mathsf{RSig.RandPK}$ algorithm are identically distributed to freshly generated keys from $\mathcal{A}$'s view as long as ω is uniformly random. Therefore, the challenge public keys $\mathsf{pk}_{\mathsf{nh},i+1,t}$ and $\mathsf{pk}_{\mathsf{nh},1,t}$ are identically distributed from $\mathcal{A}$'s point of view as long as $\mathcal{A}$ has not previously queried the random oracle H on input $(\mathsf{pk}_{j,\cdot}, \mathsf{ch}_{j,\cdot}, \mathsf{e}^{\cdot,\cdot}_{j+1})$. If $\mathcal{A}$ makes one of the aforementioned queries, it can recursively compute the public key of the challenge node, thereby trivially winning the $\mathbf{unl}^{\mathcal{A}}_{\mathsf{HDWal}[\mathsf{RSig}]}$ game. By assumption, $\mathcal{A}$ makes at most q_C queries to the child creation oracles. Therefore, there are at most $q_C + 1$ potential chaincodes

that $\mathcal{A}$ can guess correctly and query the random oracle on. For each of these, the probability of correctly guessing it is $\frac{1}{2^\kappa}$ and thereby the probability of correctly guessing any of the chaincodes is at most $\frac{q_C+1}{2^\kappa}$ during any particular random oracle query. Since $\mathcal{A}$ makes at most q_H calls to H, the overall probability of querying the random oracle on an input as above is $\frac{q_\mathsf{H}(q_C+1)}{2^\kappa}$.

It remains to show $\mathcal{A}$'s probability of winning the $\mathbf{unl}^{\mathcal{A}}_{\mathsf{HDWal[RSig]}}$ game in case the adversary challenges a hardened node with address $\mathbf{addr}_{i+1,t}$. In this case, let $\mathsf{pk}_{\mathsf{h},i+1,t}$ and $\mathsf{pk}_{\mathsf{h},1,t}$ denote the challenge public keys and let $(\mathsf{pk}_{j,\cdot}, \mathsf{ch}_{j,\cdot})$ for $1 \leq j \leq i$ denote the public key and chaincode pair of all nodes in the prefix address of $\mathbf{addr}_{i+1,t}$.

$\mathcal{A}$ is allowed to query the `CHLeakO` oracle for parent nodes, thereby eliminating the need to correctly guess a relevant chaincode. Recall that hardened public keys are derived as follows:

$$\begin{aligned}(\omega, \mathsf{ch}_{j+1,t}) &\leftarrow \mathsf{H}(\mathsf{sk}_{j,s}, \mathsf{ch}_{j,s}, \mathsf{e}^{s,t}_{j+1})\\ \mathsf{pk}_{j+1,t} &\leftarrow \mathsf{RSig.RandPK}(\mathsf{pk}_{j,s}; \omega)\end{aligned}$$

Hence, having access to the `CHLeakO` oracle does not reveal all required inputs to the random oracle, i.e., the secret key of the parent node is still unknown to the adversary. As such, according to the (perfect) rerandomizability of keys property (cf. Def 2.2), $\mathcal{A}$ can distinguish $\mathsf{pk}_{\mathsf{h},i+1,t}$ from $\mathsf{pk}_{\mathsf{h},1,t}$ only if it is able to compute the secret key of one of the challenge nodes' parents. Let E be the event that $\mathcal{A}$ can compute a secret key $\mathsf{sk}_{j,\cdot}$ that corresponds to any of the public keys $\mathsf{pk}_{j,\cdot}$ and calls the random oracle on input $(\mathsf{sk}_{j,\cdot}, \cdot, \cdot)$. Then we can upper bound the probability that event E occurs as follows:

Claim A.2 There exists an algorithm $\mathcal{A}_1$ such that

$$\mathsf{Adv}^{\mathcal{A}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \Pr[\mathsf{E}].$$

Proof. The proof of this claim corresponds to the proof of claim 5.4 in Section 5. ∎

Therefore, the adversary's advantage in case of a hardened challenge node can be upper bounded by $\mathsf{Adv}^{\mathcal{A}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}}$ and $\mathcal{A}$'s overall advantage in game $\mathbf{unl}^{\mathcal{A}}_{\mathsf{HDWal[RSig]}}$ can be upper bounded by $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{unl}_{\mathsf{HDWal[RSig]}}} \leq \frac{q_\mathsf{H}(q_C+1)}{2^\kappa} + \mathsf{Adv}^{\mathcal{A}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}}$.

∎

Indistinguishability of Hardened nodes Recall that in our construction $\mathsf{HDWal[RSig]}$, a hardened key pair $(\mathsf{sk}_{\mathsf{h},(i+1),t}, \mathsf{pk}_{\mathsf{h},(i+1),t})$ is derived via $\mathsf{SKDer}_\mathsf{H}$ and $\mathsf{PKDer}_\mathsf{H}$ as follows:

$$\begin{aligned}(\omega, \mathsf{ch}_{(i+1),t}) &\leftarrow \mathsf{H}(\mathsf{sk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathsf{e}^{s,t}_{i+1})\\ \mathsf{sk}_{\mathsf{h},(i+1),t} &\leftarrow \mathsf{RSig.RandSK}(\mathsf{sk}_{\mathsf{nh},i,s}; \omega)\\ \mathsf{pk}_{\mathsf{h},(i+1),t} &\leftarrow \mathsf{RSig.RandPK}(\mathsf{pk}_{\mathsf{nh},i,s}; \omega)\end{aligned}$$

Due to the key rerandomizability property of the underlying signature scheme RSig, $\mathcal{A}$ can only distinguish $(\mathsf{sk}_{\mathsf{h},(i+1),t}, \mathsf{pk}_{\mathsf{h},(i+1),t})$ from a fresh key pair if it can distinguish ω from random. Since we model H as a random oracle, this happens only if $\mathcal{A}$ has previously queried H on the same input, i.e., $(\mathsf{sk}_{\mathsf{nh},i,s}, \mathsf{St}_{i,s}, \mathsf{e}_{i+1,t})$. Since our model excludes secret key leakage of non-hardened nodes, the adversary cannot distinguish the output of H from a random value except if it correctly guesses $\mathsf{sk}_{\mathsf{nh},i,s}$ or any parent secret key that $\mathsf{sk}_{\mathsf{nh},i,s}$ has been (directly or indirectly) derived from.

B Impossibility of a tighter bound

In this section, we first recall the security notion of unforgeability under rerandomized keys for signature schemes with rerandomizable keys as introduced in [FKM+16]. This notion is stronger than the notion of unforgeability under *honestly* rerandomized keys in the sense that an adversary is not restricted to randomness chosen uniformly at random from the randomness space $\mathcal{R}$ for the key rerandomization. We recall the following security game:

Game $\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}$:

- **Setup Phase:** The challenger initializes the list $\mathsf{SigList} \leftarrow \{\epsilon\}$ and samples a pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{RSig.Gen}(\mathsf{par})$. Then the public key pk is sent to the adversary $\mathcal{A}$.
- **Online Phase:** $\mathcal{A}$ is given access to a signing oracle $\mathtt{RSign}$ which works as follows. On input a message m and a randomness ρ, derive a pair of keys rerandomized with the randomness ρ, as $\mathsf{sk}' \leftarrow \mathsf{RSig.SKDer}(\mathsf{sk}, \rho)$ and $\mathsf{pk}' \leftarrow \mathsf{RSig.PKDer}(\mathsf{pk}, \rho)$. A signature is then derived on message m under the secret key sk' as $\sigma \leftarrow \mathsf{RSig.Sign}(\mathsf{sk}', m)$. The message m is stored in the $\mathsf{SigList}$ and eventually the signature σ is returned as the answer.
- **Output Phase:** Finally, the adversary $\mathcal{A}$ wins the game if it can provide a signature σ^* for a message m^* relative to randomness ρ^*, where the following holds: (1) the message m^* has not been queried before, i.e., $m^* \notin \mathsf{SigList}$, and (2) σ^* is a valid forgery, i.e., $\mathsf{RSig.Verify}(\mathsf{pk}^*, \sigma^*, m^*) = 1$, where $\mathsf{pk}^* \leftarrow \mathsf{RSig.PKDer}(\mathsf{pk}, \rho^*)$.

For an algorithm $\mathcal{A}$ we define $\mathcal{A}$'s advantage in game $\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}} = \Pr[\mathbf{uf\text{-}cma\text{-}rk}^{\mathcal{A}}_{\mathsf{RSig}} = 1]$.

We now show the proof of Theorem 5.7 as presented in Section 5. Concretely, we show that for any signature scheme with rerandomizable keys RSig that satisfies **uf-cma-rk** security and for any generic transformation from RSig to a hierarchical deterministic wallet scheme $\mathsf{HDWal}^{\mathsf{RSig}}$ there exists no reduction from $\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}$ to $\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}$ that does not incur a loss polynomial in the number of $\mathtt{SKLeak0}$ oracle queries q_{sk}. In particular, this shows that the reduction in our proof of Theorem 5.3 is optimal and cannot be improved even assuming a generic transformation $\mathsf{HDWal}^{\mathsf{RSig}}$ from a **uf-cma-rk** secure signature scheme with rerandomizable keys. We show this result by assuming a reduction $\mathcal{R}$ that reduces $\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}$ to $\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}$ and by providing a metareduction $\mathcal{M}$ that uses $\mathcal{R}$ to win its own $\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}$ game. We show that the advantage of $\mathcal{M}$ in game $\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}$ has a polynomial loss in the number of $\mathtt{SKLeak0}$ oracle queries q_{sk}. Our proof proceeds in a similar fashion as the proofs in [KK18, Theorem 2] and [Cor02, Theorem 4].

We now provide the full formal proof of Theorem 5.7.

Proof. We describe a metareduction $\mathcal{M}$ that plays in the game $\mathbf{uf\text{-}cma\text{-}rk}^{\mathcal{M}}_{\mathsf{RSig}}$ and simulates the game $\mathbf{uf\text{-}cma\text{-}rk}^{\mathcal{R}}_{\mathsf{RSig}}$ to $\mathcal{R}$. Additionally, $\mathcal{M}$ simulates an adversary in game $\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}$ to $\mathcal{R}$. $\mathcal{M}$ receives a public key $\mathsf{pk}_{\mathcal{M}}$ from its challenger, and access to a signing oracle $\mathtt{RSign}$. The goal of $\mathcal{M}$ is to come up with a valid forgery in the $\mathbf{uf\text{-}cma\text{-}rk}^{\mathcal{M}}_{\mathsf{RSig}}$ game. The metareduction proceeds as follows:

1. $\mathcal{M}$ runs the reduction $\mathcal{R}$ with public key $\mathsf{pk}_{\mathcal{M}}$ as input and simulates game $\mathbf{uf\text{-}cma\text{-}rk}^{\mathcal{R}}_{\mathsf{RSig}}$ to $\mathcal{R}$ by simply forwarding $\mathcal{R}$'s queries to its own challenger. $\mathcal{R}$ sends a public key pk to $\mathcal{M}$ in the game $\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}$.
2. Assume that $\mathcal{M}$ in game $\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}$ has made q queries to the $\mathtt{HChild0}$ oracle on input pairs $(\mathbf{addr}_{\cdot,\cdot}, \mathsf{e}_{\cdot,\cdot})$ and let $\mathcal{X}$ be a set consisting of the q addresses that $\mathcal{M}$ has queried the $\mathtt{HChild0}$ oracle on (for simplicity we write $\mathcal{X} = \{\mathbf{addr}_1, \cdots, \mathbf{addr}_q\}$). Let $q_{\mathsf{sk}} \leq \lfloor q/2 \rfloor$ be the number of addresses, for which $\mathcal{M}$ invokes the Secret Key Leakage oracle. $\mathcal{M}$ picks $i \xleftarrow{\$} \{1, \ldots, q_{\mathsf{sk}}\}$, chooses $\mathbf{addr}^* \xleftarrow{\$} \mathcal{X}$ and $(\mathbf{addr}_1, \cdots, \mathbf{addr}_{q_{\mathsf{sk}}}) \xleftarrow{\$} (\mathcal{X} \setminus \{\mathbf{addr}^*\})^{q_{\mathsf{sk}}}$. This defines the following two sequences:
$$\mathcal{X}_s := (\mathbf{addr}_1, \ldots, \mathbf{addr}_{i-1}, \mathbf{addr}^*)$$
$$\mathcal{X}'_s := (\mathbf{addr}_1, \cdots, \mathbf{addr}_{q_{\mathsf{sk}}})$$
3. $\mathcal{M}$ queries the $\mathtt{SKLeak0}$ oracle on addresses in the set $\mathcal{X}_s$ and receives the corresponding secret keys as answers from $\mathcal{R}$. In particular, since $\mathbf{addr}^* \in \mathcal{X}_s$, $\mathcal{M}$ knows the secret key sk^*.
4. $\mathcal{R}$ is then rewound to the initial state. Then $\mathcal{M}$, in game $\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}$, queries the $\mathtt{SKLeak0}$ oracle on addresses from the set $\mathcal{X}'_s$. Since $\mathbf{addr}^* \notin \mathcal{X}'_s$, $\mathcal{M}$ has not corrupted the node with $\mathbf{addr}^*$.
5. $\mathcal{M}$ now tosses a biased coin τ with probability $\epsilon_{\mathcal{A}}$ of outputting 1. If $\tau = 0$, $\mathcal{M}$ sends $\perp$ to $\mathcal{R}$ in the $\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}$ game. If $\tau = 1$, $\mathcal{M}$ samples a random message m, creates a signature σ on m with secret key sk^* and returns (σ, m) as a valid forgery. This execution is done in time $t_{\mathcal{A}}$ such that $\mathcal{M}$ correctly simulates an adversary in game $\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}$.

6. Since $\mathcal{R}$ was rewound, sk^* was not revealed and (σ, m) constitutes a valid forgery. $\mathcal{R}$ derives a signature (σ', m') corresponding to challenge key $\mathsf{pk}_{\mathcal{M}}$ and returns it to $\mathcal{M}$. $\mathcal{M}$ can return (σ', m') to the $\textbf{uf-cma-rk}^{\mathcal{M}}_{\mathsf{RSig}}$ game.

Success probability of $\mathcal{M}$ We now analyze the probability with which $\mathcal{M}$ can win the $\textbf{uf-cma-rk}^{\mathcal{M}}_{\mathsf{RSig}}$ game. Let $\mathcal{Q}$ be a set of sequences of addresses such that for any sequence $(\mathbf{addr}_1, \cdots, \mathbf{addr}_j) \in \mathcal{Q}$, the corresponding `SKLeakO` oracle queries are answered correctly by $\mathcal{R}$. Additionally, it holds that if $(\mathbf{addr}_1, \cdots, \mathbf{addr}_j) \in \mathcal{Q}$, then also $(\mathbf{addr}_1, \cdots, \mathbf{addr}_{j-1}) \in \mathcal{Q}$. Let us now consider a (possibly unbounded) real adversary $\mathcal{A}$ (i.e., $\mathcal{A}$ is not simulated by $\mathcal{M}$), who issues queries to the `SKLeakO` oracle on inputs $\mathbf{addr}_j \in \mathcal{X}'_s$ and eventually outputs a valid forgery (σ, m) with success probability $\epsilon_{\mathcal{A}}$. The view of $\mathcal{R}$ is exactly the same when interacting with the real adversary $\mathcal{A}$ or with the adversary who is simulated by $\mathcal{M}$ (which we denote by $\mathcal{A}_{\mathcal{M}}$) except if the following bad event occurs: $\mathcal{X}_s \notin \mathcal{Q}$ but $\mathcal{X}'_s \in \mathcal{Q}$. In this case, the reduction $\mathcal{R}$ did not answer all `SKLeakO` oracle queries correctly in the interaction with $\mathcal{A}_{\mathcal{M}}$ before the rewind but did so after the rewind. If this event occurs, the real adversary $\mathcal{A}$ would output a valid forgery, while the simulated adversary $\mathcal{A}_{\mathcal{M}}$ would not. Hence, the reduction $\mathcal{R}$ would be able to distinguish the real from the simulated execution.

Let $\mathcal{R}^{\mathcal{A}}$ and $\mathcal{R}^{\mathcal{A}_{\mathcal{M}}}$ denote the execution of the reduction w.r.t. the real and simulated adversary, respectively. The executions $\mathcal{R}^{\mathcal{A}}$ and $\mathcal{R}^{\mathcal{A}_{\mathcal{M}}}$ are identical, except if the following bad events occur in $\mathcal{R}^{\mathcal{A}_{\mathcal{M}}}$: $\mathcal{X}'_s \in \mathcal{Q}$ and $\mathcal{X}_s \notin \mathcal{Q}$ and $\tau = 1$. Therefore, we get:

$$\begin{aligned}|\Pr[(\sigma', m') \leftarrow \mathcal{R}^{\mathcal{A}_{\mathcal{M}}}(\mathsf{pk}_{\mathcal{M}}) \wedge\ (\sigma' \text{ is valid on } m')] - \Pr[(\sigma', m') \leftarrow \mathcal{R}^{\mathcal{A}}(\mathsf{pk}_{\mathcal{M}}) \wedge\ (\sigma' \text{ is valid on } m')]| \\ \leq \epsilon_{\mathcal{A}} \cdot \Pr[\mathcal{X}'_s \in \mathcal{Q} \wedge \mathcal{X}_s \notin \mathcal{Q}].\end{aligned}$$

We recall the following lemma due to Coron [Cor02].

Lemma B.1 *Let $\mathcal{Q}$ be a set of sequences of at most q_{sk} integers in $\mathcal{X}$, such that for any sequence $(\mathbf{addr}_1, \cdots, \mathbf{addr}_j) \in \mathcal{Q}$, we have $(\mathbf{addr}_1, \cdots, \mathbf{addr}_{j-1}) \in \mathcal{Q}$. Then:*

$$\Pr_{\substack{i \xleftarrow{\$} \{1, \cdots, q_{\mathsf{sk}}\} \\ (\mathbf{addr}_1, \cdots, \mathbf{addr}_{q_{\mathsf{sk}}}, \mathbf{addr}^*) \xleftarrow{\$} \mathcal{X}^{q_{\mathsf{sk}}+1}}} \left[\begin{array}{l} (\mathbf{addr}_1, \cdots, \mathbf{addr}_{q_{\mathsf{sk}}}) \in \mathcal{Q} \\ \wedge (\mathbf{addr}_1, \cdots, \mathbf{addr}_{i-1}, \mathbf{addr}^*) \notin \mathcal{Q} \end{array} \right] \leq \frac{\exp(-1)}{q_{\mathsf{sk}}}.$$

From lemma B.1, representing addresses as integers, we get that

$$\Pr[\mathcal{X}'_s \in \mathcal{Q} \wedge \mathcal{X}_s \notin \mathcal{Q}] \leq \frac{\exp(-1)}{q_{\mathsf{sk}}} \left(1 - \frac{q_{\mathsf{sk}}}{q}\right)^{-1}.$$

Note that the additional term $\left(1 - \frac{q_{\mathsf{sk}}}{q}\right)^{-1}$ comes from the fact that we chose all $\mathbf{addr}_i$ from the set $\mathcal{X} \setminus \{\mathbf{addr}^*\}$ instead of $\mathcal{X}$. Hence, we need to consider the probability that for all $\mathbf{addr}_i$ it holds that $\mathbf{addr}_i \neq \mathbf{addr}^*$.

From this, we obtain the success probability $\mathsf{Adv}^{\mathcal{M}}_{\textbf{uf-cma-rk}_{\mathsf{RSig}}}$ for $\mathcal{M}$ as follows:

$$\begin{aligned}\mathsf{Adv}^{\mathcal{M}}_{\textbf{uf-cma-rk}_{\mathsf{RSig}}} &= \Pr[(\sigma', m') \leftarrow \mathcal{R}^{\mathcal{A}_{\mathcal{M}}}(\mathsf{pk}_{\mathcal{M}}) \wedge\ (\sigma' \text{ is valid on } m')] \\ &\geq \Pr[(\sigma', m') \leftarrow \mathcal{R}^{\mathcal{A}}(\mathsf{pk}_{\mathcal{M}}) \wedge\ (\sigma' \text{ is valid on } m')] - \epsilon_{\mathcal{A}} \cdot \frac{\exp(-1)}{q_{\mathsf{sk}}} \left(1 - \frac{q_{\mathsf{sk}}}{q}\right)^{-1} \\ &\geq \Pr[(\sigma', m') \leftarrow \mathcal{R}^{\mathcal{A}}(\mathsf{pk}_{\mathcal{M}}) \wedge\ (\sigma' \text{ is valid on } m')] - \epsilon_{\mathcal{A}} \cdot \frac{2\exp(-1)}{q_{\mathsf{sk}}} \\ &\geq \epsilon_{\mathcal{R}}(\epsilon_{\mathcal{A}}) - \epsilon_{\mathcal{A}} \cdot \frac{2\exp(-1)}{q_{\mathsf{sk}}}\end{aligned}$$

Note that, since $\mathcal{M}$ rewinds the reduction $\mathcal{R}$ once, the running time of $\mathcal{M}$ can be upper bounded by $t_{\mathcal{M}} \leq 2 \cdot t_{\mathcal{R}}(t_{\mathcal{A}})$.

■

Algorithm $\mathsf{REC}'[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}, m)$
00 $\psi \xleftarrow{\$} \{0,1\}^\kappa$
01 $\hat{m} \leftarrow (\mathsf{pk}, \psi, m)$
02 $\sigma' \leftarrow \mathsf{EC}[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}, \hat{m})$
03 Return $\sigma = (\psi, \sigma')$

Algorithm
$\mathsf{REC}'[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}, \sigma, m)$
04 $(\psi, \sigma') \leftarrow \sigma$
05 $\hat{m} \leftarrow (\mathsf{pk}, \psi, m)$
06 Return $\mathsf{EC}[\mathsf{H}_1].(\mathsf{pk}, \sigma', \hat{m})$

Algorithm
$\mathsf{REC}'[\mathsf{H}_1].\mathsf{RandSK}(\mathsf{sk}; \rho)$
00 $\mathsf{sk}' \leftarrow \mathsf{sk} \cdot \rho \bmod p$
01 Return sk'

Algorithm
$\mathsf{REC}'[\mathsf{H}_1].\mathsf{RandPK}(\mathsf{pk}; \rho)$
02 $\mathsf{pk}' \leftarrow \mathsf{pk} \cdot \rho$
03 Return pk'

Figure 6: Salted and key-prefixed version of the ECDSA signature scheme with perfectly rerandomizable keys $\mathsf{REC}'[\mathsf{H}_1] := (\mathsf{REC}'[\mathsf{H}_1].\mathsf{Gen} = \mathsf{EC}[\mathsf{H}_1].\mathsf{Gen}, \mathsf{REC}'[\mathsf{H}_1].\mathsf{Sign}, \mathsf{REC}'[\mathsf{H}_1].\mathsf{Verify}, \mathsf{REC}'[\mathsf{H}_1].\mathsf{RandSK}, \mathsf{REC}'[\mathsf{H}_1].\mathsf{RandPK})$ from the ECDSA signature scheme $\mathsf{EC}[\mathsf{H}_1]$. $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ denotes a hash function.

C Discussion (contd.)

The following Theorem follows from Theorems 3.3 and 5.3.

Theorem C.1 *Let* $\mathsf{H}_0 \colon \{0,1\}^* \to \mathbb{Z}_p$, $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ *be a hash function modeled as a random oracle. Let* $\mathsf{REC}[\mathsf{H}_1]$ *be the scheme as defined in Figure 7. Let* $\mathsf{HDWal}[\mathsf{RSig}]$ *be the construction as defined in Figure 5. We define* $\mathsf{HDWal}[\mathsf{REC}[\mathsf{H}_1]]$ *as the construction of* $\mathsf{HDWal}[\mathsf{RSig}]$*, instantiated with* $\mathsf{RSig} = \mathsf{REC}[\mathsf{H}_1]$*. Let* $\mathcal{A}$ *be an algorithm that plays in the game* $\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{REC}[\mathsf{H}_1]]}$*, then there exists an algorithm* $\mathcal{C}$ *running in roughly the same time as* $\mathcal{A}$ *such that*

$$\mathsf{Adv}^{\mathcal{C}}_{\mathbf{uf\text{-}cma1}_{\mathsf{EC}}[\mathsf{H}_0]} \geq \left(\frac{1}{4e(q_{sk}+1)} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{RSig}]}} - \frac{q^2_{\mathsf{H}_1}}{p} \right) \cdot \frac{1}{q},$$

where q_{H_1} *is the number of random oracle queries,* q *is the total number of* `HChildO` *and* `NHChildO` *oracle queries and* q_{sk} *is the number of queries to the* `SKLeakO` *oracle.*

The following Theorem follows from Theorem 5.3 and Theorem 5.1 from [DFL19].

Theorem C.2 *Let* $\mathsf{H}_0 \colon \{0,1\}^* \to \mathbb{Z}_p$, $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ *be hash functions modeled as a random oracles. Let* $\mathsf{REC}'[\mathsf{H}_1]$ *be the scheme as defined in Figure 6. Let* $\mathsf{HDWal}[\mathsf{RSig}]$ *be the construction as defined in Figure 5. We define* $\mathsf{HDWal}[\mathsf{REC}'[\mathsf{H}_1]]$ *as the construction of* $\mathsf{HDWal}[\mathsf{RSig}]$*, instantiated with* $\mathsf{RSig} = \mathsf{REC}'[\mathsf{H}_1]$*. Let* $\mathcal{A}$ *be an algorithm that plays in the game* $\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{REC}'[\mathsf{H}_1]]}$*, then there exists an algorithm* $\mathcal{C}$ *running in roughly the same time as* $\mathcal{A}$ *such that*

$$\mathsf{Adv}^{\mathcal{C}}_{\mathbf{uf\text{-}cma1}_{\mathsf{EC}}[\mathsf{H}_0]} \geq \frac{1}{4e(q_{sk}+1)} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{RSig}]}} - \frac{3{q_{\mathsf{H}_1}}^2}{p},$$

where q_{H_1} *is the number of random oracle queries and* q_{sk} *is the number of queries to the* `SKLeakO` *oracle.*

Lemma C.3 *Consider the algorithm* $\mathsf{Trf}[\mathsf{H}_0, \mathsf{H}_1]_{\mathsf{EC}}$ *in Figure 8. Suppose that:*

- $\omega = \frac{\mathsf{H}_1(m_1)}{\mathsf{H}_0(m_0)} \in \mathbb{Z}_p$,
- $X_0, X_1 \in \mathbb{E}$ *s.t.* $X_0 = x_0 \cdot G$ *and* $X_1 = \omega \cdot X_0$,
- $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(X_1, \sigma_1, m_1) = 1$,
- $\sigma_0 \leftarrow \mathsf{Trf}[\mathsf{H}_0, \mathsf{H}_1]_{\mathsf{EC}}(m_0, m_1, \sigma_1, \omega, X_0, X_1)$.

Then $\mathsf{EC}[\mathsf{H}_0].\mathsf{Verify}(X_0, \sigma_0, m_0) = 1$.

Algorithm
$\mathsf{MREC}[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}, m)$
00 $\mathsf{pm} \leftarrow (\mathsf{pk}, m)$
01 $\sigma \leftarrow \mathsf{EC}[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}, \mathsf{pm})$
02 Return σ

Algorithm
$\mathsf{MREC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}, \sigma, m)$
03 $\mathsf{pm} \leftarrow (\mathsf{pk}, m)$
04 Return
$\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}, \sigma', \mathsf{pm})$

Algorithm
$\mathsf{MREC}[\mathsf{H}_1].\mathsf{RandSK}(\mathsf{sk}; \rho)$
00 $\mathsf{sk}' \leftarrow \mathsf{sk} \cdot \rho \bmod p$
01 Return sk'

Algorithm
$\mathsf{MREC}[\mathsf{H}_1].\mathsf{RandPK}(\mathsf{pk}; \rho)$
02 $\mathsf{pk}' \leftarrow \mathsf{pk} \cdot \rho$
03 Return pk'

Figure 7: Salt-free and key-prefixed version of the ECDSA signature scheme with perfectly rerandomizable keys $\mathsf{MREC}[\mathsf{H}_1] := (\mathsf{MREC}[\mathsf{H}_1].\mathsf{Gen} = \mathsf{EC}[\mathsf{H}_1].\mathsf{Gen}, \mathsf{MREC}[\mathsf{H}_1].\mathsf{Sign}, \mathsf{MREC}[\mathsf{H}_1].\mathsf{Verify}, \mathsf{MREC}[\mathsf{H}_1].\mathsf{RandSK}, \mathsf{MREC}[\mathsf{H}_1].\mathsf{RandPK})$ from the ECDSA signature scheme $\mathsf{EC}[\mathsf{H}_1]$. $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ denotes a hash function.

$\mathsf{Trf}[\mathsf{H}_0, \mathsf{H}_1]_{\mathsf{EC}}(m_0, m_1, \sigma_1, \omega, X_0, X_1)$
00 $z_0 \leftarrow \mathsf{H}_0(m_0)$
01 $z_1 \leftarrow \mathsf{H}_1(m_1)$
02 If $(\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\sigma_1, m_1, X_1) = 0) \vee \left(\omega \neq \frac{z_1}{z_0} \vee X_1 \neq X_0 \cdot \omega\right)$:
03 Return $\perp$
04 $(r, s_1) \leftarrow \sigma_1$
05 $s_0 \leftarrow \frac{s_1}{\omega} \bmod p$
06 $\sigma_0 \leftarrow (r, s_0)$
07 Return σ_0

Figure 8: Figure shows the $\mathsf{Trf}_{\mathsf{ECDSA}}$ algorithm for hash functions $\mathsf{H}_0, \mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$.

Theorem C.4 *Let $\mathsf{H}_0 \colon \{0,1\}^* \to \mathbb{Z}_p$, $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ be a hash function modeled as a random oracle. Let $\mathcal{A}$ be an algorithm that plays in the game* $\textbf{uf-cma-hrk}_{\mathsf{MREC}[\mathsf{H}_1]}$*, then there exists an algorithm $\mathcal{C}$ running in roughly the same time as $\mathcal{A}$ such that*

$$\mathsf{Adv}^{\mathcal{C}}_{\textbf{uf-cma}_{\mathsf{EC}[\mathsf{H}_0]}} \geq \mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk}_{\mathsf{MREC}[\mathsf{H}_1]}} - \frac{3q^2}{p}$$

Proof. Consider an adversary $\mathcal{A}$ playing in Game $\textbf{uf-cma-hrk}_{\mathsf{MREC}[\mathsf{H}_1]}$. As such $\mathcal{A}$ is granted access to the oracles $\mathtt{Rand}, \mathtt{RSign}$, and the random oracle $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$. In the following, we use that $2^\kappa \leq p$. We prove the statement via a sequence of games. Each game $\boldsymbol{G}_{i(i>0)}$ is presented in Figure 10 via the description of the oracles that are modified with respect to the previous game $\boldsymbol{G}_{i-1}$. The exact differences of game $\boldsymbol{G}_i$ to game $\boldsymbol{G}_{i-1}$ are highlighted in the form of boxed pseudocode. Moreover, we denote by $E_{i-1,i}$ a difference event, where the indices of the event correspond to games $\boldsymbol{G}_{i-1}, \boldsymbol{G}_i$ that are affected by the event.

GAME $\boldsymbol{G}_0$: This game is equivalent to the original $\textbf{uf-cma-hrk}_{\mathsf{MREC}[\mathsf{H}_1]}$ game. In particular, a key pair $(\mathsf{sk}, \mathsf{pk})$ is sampled as $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{MREC}[\mathsf{H}_1].\mathsf{Gen}(\mathsf{par})$. The adversary $\mathcal{A}$ is given pk as the challenge public key and oracle access to $\mathtt{Rand}$, $\mathtt{RSign}$ and random oracle H_1. $\mathcal{A}$ can query $\mathtt{Rand}$ to receive a randomness ρ and make a follow-up query to $\mathtt{RSign}$ to receive a signature on message m with respect to the rerandomized key $\mathsf{pk}' \leftarrow \mathsf{pk} \cdot \rho$. In particular, $\mathcal{A}$ is allowed to query $\mathtt{RSign}$ on every input pair (m, ρ) at most once. Additionally, $\mathcal{A}$ can make direct queries to the random oracle H_1. The game internally maintains a random oracle H_0 in a straightforward manner, by storing a list H_0 of query/response pairs. Eventually, in order to win the game, $\mathcal{A}$ has to come up with a valid forgery σ^* on a new message m^* with respect to a randomness ρ^*. Since $\boldsymbol{G_0}$ proceeds as $\textbf{uf-cma-hrk}_{\mathsf{MREC}[\mathsf{H}_1]}$ we have that

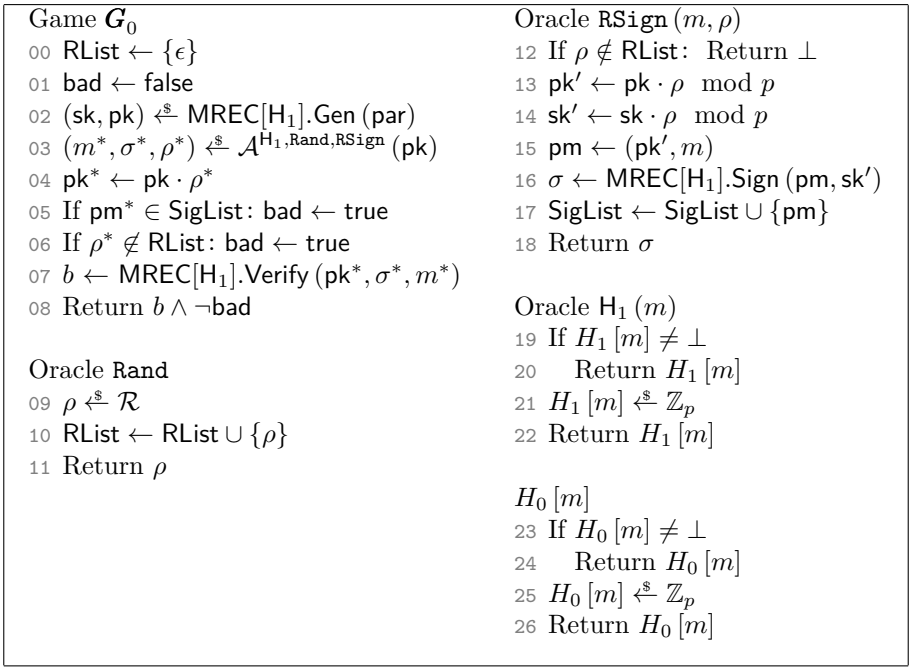

Figure 9: Game $\boldsymbol{G}_0 = \textbf{uf-cma-hrk}_{\mathsf{MREC}[\mathsf{H}_0]}$ with adversary $\mathcal{C}$.

$\Pr[\boldsymbol{G_0} = 1] = \Pr[\textbf{uf-cma-hrk}_{\mathsf{MREC}[\mathsf{H}_1]} = 1] = \mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk}_{\mathsf{MREC}[\mathsf{H}_1]}}$.

GAME $\boldsymbol{G}_1$: In $\boldsymbol{G}_1$, the way that random oracle queries to H_1 from $\mathcal{A}$ are answered, is internally modified as follows. To answer queries to H_1, $\boldsymbol{G}_1$ internally keeps two lists H_1 and H_1' which it programs throughout its interaction with $\mathcal{A}$. Depending on whether a queried message m contains as part of its prefix a public key pk', it programs $H_1[m]$ and $H_1'[m]$ in two different possible ways. Note that pk' is the result of rerandomizing pk as $\mathsf{pk}' = \mathsf{pk} \cdot \rho$, where $\rho \leftarrow \mathtt{Rand}(\rho \in \mathsf{RList})$ is a previous answer to an oracle query Rand. We now analyze the three types of queries to H_1 that can occur.

- $\mathsf{H}_1[m] \neq \perp$: In this case, $\boldsymbol{G}_1$ returns $H_1[m]$.
- $\mathsf{H}_1[m] = \perp$ and m is of the form $m = (\mathsf{pk}', \cdot)$, s.t. $\mathsf{pk}' = \mathsf{pk} \cdot \rho$ for some $\rho \in \mathsf{RList}$: In this case, $\boldsymbol{G}_1$ computes $h \leftarrow \mathsf{H}_0(ctr)$, where $ctr \xleftarrow{\$} \{0,1\}^\kappa$. Consequently, $\boldsymbol{G}_1$ sets $H_1[m] \leftarrow \rho \cdot h \mod p$ and $H_1'[m] \leftarrow ctr$. It returns $H_1[m]$.
- Otherwise, $\boldsymbol{G}_1$ samples $h \xleftarrow{\$} \mathbb{Z}_p$ and sets the values $H_1[m] \leftarrow h$, $H_1'[m] \leftarrow \epsilon$. It then returns $H_1[m]$.

It is easy to see that all answers for queries to H_1 that $\boldsymbol{G}_1$ returns are uniformly distributed from $\mathcal{A}$'s perspective. This follows from the uniformity of output h computed via random oracle H_0. Therefore, $\boldsymbol{G}_1$ behaves exactly as $\boldsymbol{G}_0$.

GAME $\boldsymbol{G}_2$: In $\boldsymbol{G}_2$, the way in which queries to Rand are answered, is internally modified as follows. When $\mathcal{A}$ asks a query of the form Rand, the game aborts if there exists a message of the form $m = (\mathsf{pk}', \cdot)$ for which $H_1'[m]$ evaluates to ϵ and where pk' is the (rerandomized) key that corresponds to the return value ρ of Rand, i.e., $\mathsf{pk}' = \mathsf{pk} \cdot \rho$. The following claim bounds the probability of such an abort scenario.

Claim C.5 Let $E_{1,2}$ denote the event that $\boldsymbol{G}_2$ aborts during a Rand query, for which $H_1'[m]$ evaluates to ϵ, where $m = (\mathsf{pk}', \cdot)$. Then $\Pr[E_{1,2}] \leq \frac{q^2}{p}$.

Proof. During any particular call to the oracle Rand, this event can only occur if $\mathcal{A}$ has already made a query of the form $\mathsf{H}_1(m)$, where $m = (\mathsf{pk}', \cdot)$ (prior to the oracle Rand returning the value ρ for this query). Since RList contains at most q values at any point during the game, any of them coincide with the (uniformly chosen) value ρ with probability at most $\frac{q}{p}$. Since keys are uniquely rerandomizable, a query of the form $\mathsf{H}_1(m)$ thus also has probability at most $\frac{q}{p}$ of having been made prior to this particular call to Rand. Since there at most q queries to Rand, it follows that $\Pr[E_{1,2}] \leq \frac{q^2}{p}$. ∎

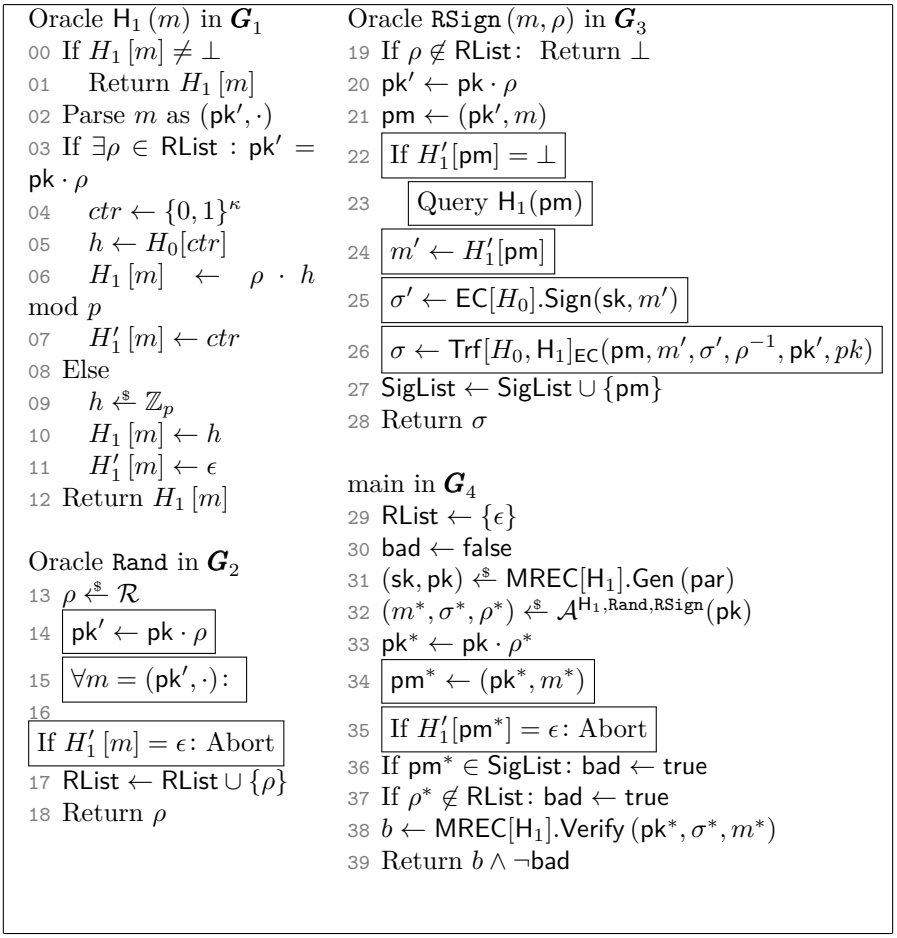

Figure 10: Games $\boldsymbol{G}_1$-$\boldsymbol{G}_4$

Since the games $\boldsymbol{G}_1$, $\boldsymbol{G}_2$ are equivalent unless the event $\Pr[E_{1,2}]$ occurs, $\Pr[\boldsymbol{G}_0 = 1] \leq \Pr[\boldsymbol{G}_1] + \frac{q^2}{p}$

GAME $\boldsymbol{G}_3$: In $\boldsymbol{G}_3$, the way that signing queries from $\mathcal{A}$ are answered, is again internally modified as follows. When $\mathcal{A}$ makes a query of the form $\mathtt{RSign}(m, \rho)$, $\boldsymbol{G}_3$ first checks whether $\rho \in \mathsf{RList}$ and if not, returns $\perp$. Otherwise, it computes $\mathsf{pk}' \leftarrow \mathsf{pk} \cdot \rho$, and sets $\mathsf{pm} \leftarrow (\mathsf{pk}', m)$. If $H_1[\hat{m}] = \perp$, it internally queries H_1 on input message pm. This means it queries $h \leftarrow \mathsf{H}(ctr)$, where $ctr \xleftarrow{\$} \{0,1\}^\kappa$. $\boldsymbol{G}_3$ internally sets $H_1[\mathsf{pm}] \leftarrow \rho \cdot h \mod p$ and stores $H_1'[\mathsf{pm}] \leftarrow ctr$. After making the query to H_1, $\boldsymbol{G}_3$ fetches $m' \leftarrow H_1'[\mathsf{pm}]$, where m' was set to ctr during H_1 query. Since sk is known to the game, it can now compute the signature σ' as $\sigma' \xleftarrow{\$} \mathsf{EC}[H_0].\mathsf{Sign}(\mathsf{sk}, m')$. Finally, it computes and returns the signature σ as $\sigma \leftarrow \mathsf{Trf}[H_0, \mathsf{H}_1]_{\mathsf{EC}}(\mathsf{pm}, m', \sigma', \rho^{-1}, \mathsf{pk}', \mathsf{pk})$, where $\mathsf{pk} = \mathsf{pk}' \cdot \rho^{-1}$.

Claim C.6 $\Pr[\boldsymbol{G}_2 = 1] = \Pr[\boldsymbol{G}_3 = 1]$

Proof. We argue that in both games, the answers to signing queries are identically distributed. To this end, we analyze how $\boldsymbol{G}_3$ replies to a query of the form $\mathtt{RSign}(m, \rho)$. $\boldsymbol{G}_3$ derives signature σ on input (m, ρ) as $\sigma \leftarrow \mathsf{Trf}[H_0, \mathsf{H}_1]_{\mathsf{EC}}(\mathsf{pm}, m', \sigma', \rho^{-1}, \mathsf{pk}', \mathsf{pk})$, where $m' = H_1'[\mathsf{pm}], \mathsf{pk} = \mathsf{pk}' \cdot \rho^{-1}, \mathsf{EC}[H_0].\mathsf{Verify}(\mathsf{pk}, \sigma', m') = 1$, and $\frac{H_0[m']}{H_1[\mathsf{pm}]} = \frac{h'}{H_1[\mathsf{pm}]} = \frac{h'}{\rho \cdot h'} = \rho^{-1} \mod p$. It follows from Lemma C.3 that σ constitutes a correct signature on message pm and under public key pk' relative to $\mathsf{EC}[H_0].\mathsf{Verify}$. It follows immediately that the signature σ constitutes a valid signature relative to $\mathsf{MREC}[\mathsf{H}_1].\mathsf{Verify}$. This concludes the proof. ∎

GAME $\boldsymbol{G}_4$: $\boldsymbol{G}_4$ behaves identically to $\boldsymbol{G}_3$ except for the following modification in the main procedure: Upon receiving a forgery of the form (m^*, σ^*, ρ^*) from $\mathcal{A}$, it sets $\mathsf{pm}^* \leftarrow (\mathsf{pk}^*, m^*)$ and aborts if $H_1'[\mathsf{pm}^*] = \epsilon$.

Claim C.7 Let $E_{3,4}$ be the event that $\boldsymbol{G}_4$ aborts if $H_1'[\mathsf{pm}^*] = \epsilon$, where $\mathsf{pm}^* = (\mathsf{pk}^*, m^*)$. Then $\Pr[E_{3,4}] \leq \frac{q^2}{p}$.

Proof. The only way this event can happen, is if $\mathcal{A}$ manages to make a query of the form $\mathsf{H}_1(\mathsf{pm}^*)$ before querying $\texttt{Rand}$ to obtain the corresponding value of ρ^*. The proof of this claim follows in a similar way as the corresponding proof in claim C.5. ∎

Since the games $\boldsymbol{G}_3$, $\boldsymbol{G}_4$ are equivalent unless event $E_{3,4}$ occurs, $\Pr[\boldsymbol{G}_3 = 1] \leq \Pr[\boldsymbol{G}_4 = 1] + \frac{q^2}{p}$

Reduction to UF-CMA security. We describe an algorithm $\mathcal{C}$ that plays in the $\textbf{uf-cma}_{\mathsf{EC}[\mathsf{H}_0]}$ game and simulates game $\boldsymbol{G}_4$ to $\mathcal{A}$. Instead of sampling its own key pair as is done in $\boldsymbol{G}_4$, $\mathcal{C}$ obtains as input a public key $\mathsf{pk}_{\mathcal{C}}$ from the $\textbf{uf-cma}_{\mathsf{EC}[\mathsf{H}_0]}$ game and is given access to the signing oracle $\texttt{Sign}$ to obtain signatures under $\mathsf{pk}_{\mathcal{C}}$ under messages of its choice. Furthermore, $\mathcal{C}$ has access to the random oracle H_0 by which it replaces the list H_0. $\mathcal{C}$ runs $\mathcal{A}$ on input $\mathsf{pk}_{\mathcal{C}}$.

Simulation of Randomness Queries. Queries to $\texttt{Rand}$ from $\mathcal{A}$ do not require knowledge of the secret key corresponding to $\mathsf{pk}_{\mathcal{C}}$ and hence are straight forward to simulate.

Simulation of Random Oracle Queries. $\mathcal{C}$'s simulation of random oracle queries coincides with the above programming strategy that is already internally present in $\boldsymbol{G}_4$.

Simulation of Signing Queries. Recall that in $\boldsymbol{G}_4$, queries of the form $\texttt{RSign}\,(m, \rho)$ internally prompt the computation of signature $\sigma' = \mathsf{EC}[\mathsf{H}_0].\mathsf{Sign}(\mathsf{sk}_{\mathcal{C}}, m')$, where $m' \leftarrow ctr$. Since $\mathcal{C}$ does not know $sk_{\mathcal{C}}$, it needs to compute σ' via a call to its signing oracle, i.e., as $\sigma' \leftarrow \texttt{Sign}(m')$. Other than that $\mathcal{C}$ simulates such a query exactly as internally done for $\boldsymbol{G}_4$.

Extracting the forgery. When the tuple (m^*, σ^*, ρ^*) is returned as an answer from $\mathcal{A}$, $\mathcal{C}$ checks whether it constitutes a valid forgery, and aborts otherwise (note that in this case, $\boldsymbol{G}_4$ would return 0, so $\mathcal{C}$ can safely abort). In case $\mathcal{C}$ does not abort, it computes $\mathsf{pk}^* = \mathsf{pk}_{\mathcal{C}} \cdot \rho^*$, where pk^* is the public key under which $\mathcal{A}$'s forgery is valid. $\mathcal{C}$ computes $\mathsf{pm}^* \leftarrow (\mathsf{pk}^*, m^*)$ and if $H_1'[\mathsf{pm}^*] = \epsilon$, it aborts. Otherwise, $\mathcal{C}$ fetches $m' \leftarrow H_1'[\mathsf{pm}^*]$ and computes

$$\sigma' \leftarrow \mathsf{Trf}[\mathsf{H}_1, \mathsf{H}_0]_{\mathsf{ECDSA}}\,(m', \mathsf{pm}^*, \sigma^*, \rho^*, \mathsf{pk}_{\mathcal{C}}, \mathsf{pk}^*)\,.$$

Since $H_1[\mathsf{pm}^*] = \mathsf{H}_0(H_1'[\mathsf{pm}^*]) \cdot \rho^* = \mathsf{H}(m') \cdot \rho^*$, we have that $\frac{H_1[\mathsf{pm}^*]}{\mathsf{H}_0(m')} = \frac{\mathsf{H}_0(m') \cdot \rho^*}{\mathsf{H}_0(m')} = \rho^*$. Together with $\mathsf{pk}^* = \mathsf{pk}_{\mathcal{C}} \cdot \rho^*$ and $\mathsf{MREC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}^*, \sigma^*, \mathsf{pm}^*) = 1$, Lemma C.3 implies that

$$\mathsf{EC}[\mathsf{H}_0].\mathsf{Verify}\,(\mathsf{pk}_{\mathcal{C}}, \sigma', m') = 1.$$

Claim C.8 (m', σ') constitutes a valid forgery in $\textbf{uf-cma}_{\mathsf{EC}[\mathsf{H}_0]}$ with probability $1 - q^2/p$.

Proof. We have to show that the query $\texttt{Sign}(m')$ was not made by $\mathcal{C}$ during its simulation and hence (m', σ') is a valid forgery in $\textbf{uf-cma}_{\mathsf{EC}[\mathsf{H}_0]}$. Note that $\mathcal{A}$ has not made a query of the form $\texttt{RSign}\,(m^*, \rho^*)$ throughout the simulation. If it had, (m^*, σ^*, ρ^*) would not constitute a valid forgery in $\boldsymbol{G}_4$ and the simulation would have aborted at this point. This implies that $\mathcal{C}$ never had to simulate a query $\texttt{RSign}(m^*, \rho^*)$ to $\mathcal{A}$ which entailed a H_1 query on message $\mathsf{pm}^* \leftarrow (\mathsf{pk}^*, m^*)$. Hence, m' associated with query $\mathsf{H}_1(\mathsf{pm}^*)$ was not queried by $\mathcal{C}$ to the oracle $\texttt{Sign}$ in any query of the form $\texttt{RSign}\,(m, \rho)$ with $m \neq m^*$ unless there exist (any) two values m_1, m_2 s.t. $H_1'[m_1] = H_1'[m_2] \neq \perp$. It is easy to see that this happens with probability at most q^2/p during $\mathcal{C}$'s simulation, since all values that $\mathcal{C}$ queries to the oracle $\texttt{Sign}$ are sampled independently and uniformly at random from $\{0,1\}^{\kappa}$. ∎

From claims C.5-C.7, we have $\Pr[\boldsymbol{G}_0 = 1] \leq \Pr[\boldsymbol{G}_4] + \frac{3q^2}{p}$ Since $\mathcal{C}$ provides a perfect simulation of $\boldsymbol{G}_4$ to $\mathcal{A}$ up to an error of q^2/p, as shown in the previous claim, we obtain

$$\mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk},\mathsf{MREC}[\mathsf{H}_1]} \leq \mathsf{Adv}^{\mathcal{A}}_{\boldsymbol{G}_4} + \frac{3q^2}{p} \leq \mathsf{Adv}^{\mathcal{C}}_{\textbf{uf-cma},\mathsf{EC}[\mathsf{H}_0]} + \frac{3q^2}{p},$$

which implies the theorem.

∎

Theorem 5.3 can be combined with Theorem C.4 in a similar manner as Theorem C.2.