

Singapore Management University

## Institutional Knowledge at Singapore Management University

---

Research Collection School Of Computing and Information Systems

School of Computing and Information Systems

---

9-2022

### Secure hierarchical deterministic wallet supporting stealth address

Xin YIN

*Shanghai Jiao Tong University*

Zhen LIU

Guomin YANG

*Singapore Management University, gmyang@smu.edu.sg*

Guoxing CHEN

Haojin ZHU

Follow this and additional works at: [https://ink.library.smu.edu.sg/sis\\_research](https://ink.library.smu.edu.sg/sis_research)



Part of the [Databases and Information Systems Commons](#), [Finance and Financial Management Commons](#), and the [Information Security Commons](#)

---

#### Citation







YIN, Xin; LIU, Zhen; YANG, Guomin; CHEN, Guoxing; and ZHU, Haojin. Secure hierarchical deterministic wallet supporting stealth address. (2022). *Computer Security ESORICS 2022: 27th European Symposium on Research, Copenhagen, Denmark, September 26-30: Proceedings*. 13554, 89-109.

Available at: [https://ink.library.smu.edu.sg/sis\\_research/7476](https://ink.library.smu.edu.sg/sis_research/7476)

This Conference Proceeding Article is brought to you for free and open access by the School of Computing and Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Computing and Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email [cherylds@smu.edu.sg](mailto:cherylds@smu.edu.sg).



# Secure Hierarchical Deterministic Wallet Supporting Stealth Address

Xin Yin<sup>1</sup> , Zhen Liu<sup>1,3,4</sup>  , Guomin Yang<sup>2</sup> , Guoxing Chen<sup>1</sup> ,  
and Haojin Zhu<sup>1</sup> 

<sup>1</sup> Shanghai Jiao Tong University, Shanghai 200240, China  
{yinxin, liuzhen, guoxingchen, zhu-hj}@sjtu.edu.cn

<sup>2</sup> Singapore Management University, Singapore 178902, Singapore  
gmyang@smu.edu.sg

<sup>3</sup> State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China

<sup>4</sup> Shanghai Qizhi Institute, Shanghai 200232, China

**Abstract.** Over the past decade, cryptocurrency has been undergoing a rapid development. Digital wallet, as the tool to store and manage the cryptographic keys, is the primary entrance for the public to access cryptocurrency assets. Hierarchical Deterministic Wallet (HDW), proposed in Bitcoin Improvement Proposal 32 (BIP32), has attracted much attention and been widely used in the community, due to its virtues such as easy backup/recovery, convenient cold-address management, and supporting trust-less audits and applications in hierarchical organizations. While HDW allows the wallet owner to generate and manage his keys conveniently, Stealth Address (SA) allows a payer to generate fresh address (i.e., public key) for the receiver without any interaction, so that users can achieve “one coin each address” in a very convenient manner, which is widely regarded as a simple but effective way to protect user privacy. Consequently, SA has also attracted much attention and been widely used in the community. However, as so far, there is not a *secure* wallet algorithm that provides the virtues of both HDW and SA. Actually, even for standalone HDW, to the best of our knowledge, there is no strict definition of syntax and models that captures the functionality and security (i.e., safety of coins and privacy of users) requirements that practical scenarios in cryptocurrency impose on wallet. As a result, the existing wallet algorithms either have (potential) security flaws or lack crucial functionality features.

In this work, after investigating HDW and SA comprehensively and deeply, we formally define the syntax and security models of Hierarchical Deterministic Wallet supporting Stealth Address (HDWSA), capturing the functionality and security (including safety and privacy) requirements imposed by the practice in cryptocurrency, which include all the versatile functionalities that lead to the popularity of HDW and SA as well as all the security guarantees that underlie these functionalities. We propose a concrete HDWSA construction and prove its security in the random oracle model. We implement our scheme and the experimental results show that the efficiency is suitable for typical cryptocurrency settings.

**Keywords:** Signature scheme · Hierarchical deterministic wallet · Stealth address · Blockchain · Cryptocurrency

## 1 Introduction

Since the invention of Bitcoin in 2008, cryptocurrency has been undergoing a tremendous development and been attracting much attention in the community. Digital Signature [14,16] is employed in cryptocurrencies to enable users to own and spend their coins. More specifically, each coin is assigned to a public key (which is also referred to as coin-address), implying that the coin belongs to the owner of the public key. When a user wants to spend the coin on a public key  $pk$ , he needs to generate a transaction  $tx$  and a signature  $\sigma$  such that  $(tx, \sigma)$  is a valid (message, signature) pair with respect to  $pk$ , authenticating the spending of the coin by this transaction. In such a mechanism, the secret key is the only thing that a user uses to own and spend his coins. Naturally, key management plays a crucial role in cryptocurrencies and it needs to work like a “wallet” for the coins, providing some particular features reflecting the functionalities of currency, such as making the transfers among users convenient and/or preserving the users’ privacy. Actually, digital wallet is indispensable for any cryptocurrency system. A secure, convenient, and versatile wallet is desired.

**Hierarchical Deterministic Wallet and Its Merits.** Hierarchical Deterministic Wallet (HDW), proposed in BIP32 [21], has been accepted as a standard in the Bitcoin community. Roughly speaking, HDW is characterized by three functionality features: *deterministic generation property*, *master public key property*, and *hierarchy property*. As the name implies, the *deterministic generation property* means that all keys in a wallet are *deterministically* generated from a “seed” directly or indirectly, so that when necessary (e.g., the crash of the device hosting the wallet) the wallet owner can recover all the keys from the seed. The *master public key property* means that a wallet owner can generate derived public keys from the wallet’s master public key and use the derived public keys as coin-addresses to receive coins, without needing any secrets involved, and subsequently, the wallet owner can generate the corresponding derived secret keys to serve as the secret signing keys. The *hierarchy property* means that the derived key pairs could serve as the master key pairs to generate further derived keys. With these three functionality features, HDW provides very appealing virtues which lead to its popularity in the community. Readers are referred to [7,10,12,21] for the details of these use cases.

**“One Coin Each Cold-address” for Enhanced Coin Safety and User Privacy.** In cryptocurrencies, before the corresponding secret key appears in vulnerable online devices (referred to as “hot storage”, e.g., computers or smart phones that are connected to Internet), a public key (i.e., coin-address) is referred to as a “cold-address”. Once the corresponding secret key is exposed in any hot storage, it is not cold any more and thus becomes a “hot address”. The cold/hot-address mechanism (or referred to as cold/hot wallet mechanism) is used to

reduce the exposure chance of secret keys and achieve better safety of the coins. As shown in Fig. 1, comparing with traditional wallet (where the public/secret key pairs are generated via a standard key generation algorithm), HDW’s master public key property enables the wallet owner to generate cold-addresses much more conveniently. Namely, the owner stores the master public key in a hot storage, and when needed, he can generate derived public keys from the master public key without needing any secrets. Note that these derived public keys keep to be cold-addresses until the owner generates the corresponding derived secret keys and uses them in a hot storage to spend the coins. In addition, as considered in [1, 6–9, 13, 21], the convenient cold-address generation implied by master public key property is also used to enhance the privacy of the wallet owner, say achieving *transaction unlinkability*. In particular, “one coin each address” mechanism is a simple but effective way to achieve transaction unlinkability. However, for traditional wallets, this will result in a huge cost on generating and managing a large number of (public key, secret key) pairs, especially when cold-address mechanism is considered simultaneously. In contrast, for wallets with master public key property, generating derived public keys from master public key is very simple and convenient and the generated derived public keys are inherently cold, so they achieve “one coin each cold-address”<sup>1</sup> efficiently.

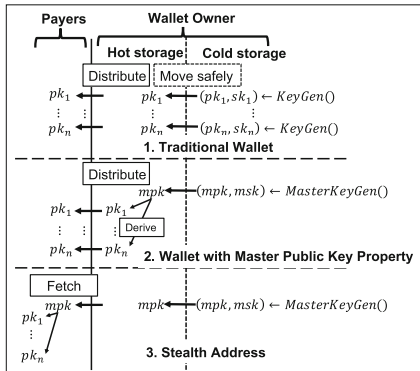


Fig. 1. Cold address generation/distribution.

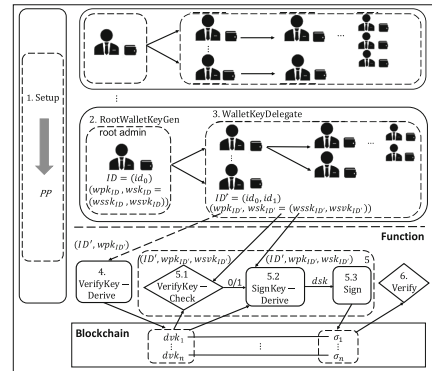


Fig. 2. System model.

**Stealth Address for Efficient, Non-interactive and Privacy-Preserving Payment.** While HDW focuses on the key derivation and management from the point of view of a wallet owner managing his keys, *stealth address* (SA) [5, 15, 17, 18] is another popular key derivation and management mechanism in the community, but from the perspective of privacy-preservation when transferring

<sup>1</sup> Note that we change the term from “one coin each address” to “one coin each cold-address”, to explicitly emphasize that it is not only for the privacy-preservation but also for the safety of coins.

coins among users. More specifically, with SA mechanism, each user publishes his master public key, and for each payment, the payer could generate a fresh derived public key for the payee from the payee’s master public key, without needing any interaction with the payee. On the payee’s side, when he wants to spend a coin on a derived public key belonging to him, he can generate the corresponding derived secret/signing key by himself, without needing any interaction with the payer or anyone else. While the derived public keys serve as the coin-addresses, the master public key never appears in any transaction or blockchain of the cryptocurrency, and no one (except the payer and payee) could link a derived public key to the corresponding master public key, so that master public key is “stealth” from the public. In summary, by SA mechanism, each coin-address is fresh and unique by default (unless the payer uses the same random data for each of his payments to the same payee<sup>2</sup>), so that there is no such issue as “address reuse” by design. In other words, *SA is an inherent mechanism for “one coin each cold-address”*.

**Hierarchical Deterministic Wallet Supporting Stealth Address for Improved Versatility and Security.** Noting the virtues of HDW and SA, as well as the facts that both are related to key derivation/management and both take convenient cold-address generation as an important virtue, it is natural to consider *Hierarchical Deterministic Wallet supporting Stealth Address* (HDWSA), which will be a more versatile wallet providing virtues of HDW and SA *simultaneously*, and consequently will empower more applications in cryptocurrency. However, designing a secure HDWSA is quite challenging, rather than a trivial combination of two existing mechanisms. To the best of our knowledge, as shown in Table 1, such a wallet has not been proposed yet. Existing wallet algorithms either suffer from (potential) security flaws or lack crucial functionality features.

We now explain that HDWSA is well motivated by realistic scenarios, but existing schemes fail to achieve both security and full functionality. As shown in [1, 6, 7, 10, 21], master public key property enables convenient cold-address generation, thus supports “one coin each cold-address”, and helps achieve transaction unlinkability. Note that master public key property means that the wallet owner can *generate cold-addresses from only information stored in hot storage* (which we refer to as “*cold-address generation material*” below), without needing any sensitive secrets. Considering this property of HDW more comprehensively and deeply, we can find two issues: (1) to use these cold-addresses to receive coins, the wallet owner has to somehow distribute these cold-addresses to the corresponding payers, and (2) as storing the cold-address generation material in hot storage is the fundamental setting that enables the master public key property, the cold-address generation material is very likely to be leaked due to its continuous exposure in hot storage. These issues would cause not only inconvenience but also privacy concerns, since for the existing HDW algorithms, if the cold-address generation material is leaked, the privacy is compromised completely (e.g., [6, 10, 21]) or partially (e.g., [1, 7]). In contrast, as shown in Fig. 1, SA

<sup>2</sup> Note that the payee can detect such malicious behaviors easily.

**Table 1.** Comparison with existing works

Scheme <sup>a</sup>	Support master public key property?	Support hierarchy property?	Support stealth address?	Privilege escalation attack resistant?	Privacy Preserving? <sup>b</sup>	Formally modeled and proved?
[21]	✓	✓	✗	✗	✗	✗
Hardened [21]	✗	✓	✗	✓	— <sup>c</sup>	✗
[10]	✓	✓	✗	Partially <sup>d</sup>	✗	Partially <sup>d</sup>
[8, 9]	✓	✓	✗	✓	✗	✗
[13]	✓	Partially <sup>e</sup>	✗	✓	✗	✓
[1, 7]	✓	✗	✗	— <sup>f</sup>	Partially <sup>g</sup>	✓
[6]	✓	✓	✗	Partially <sup>h</sup>	✗ <sup>h</sup>	✓
[15, 17]	— <sup>i</sup>	— <sup>i</sup>	✓	✗	✓	✗
[11, 12]	✓	✗	✓	✓	✓	✓
This work	✓	✓	✓	✓	✓	✓

<sup>a</sup> All schemes in this table support deterministic generation property.

<sup>b</sup> Here “Privacy-Preserving” means “Privacy-Preserving when cold-address generation material (in hot storage) is compromised”.

<sup>c</sup> The Hardened BIP32 in [21] does not support convenient cold-address generation, since it loses the master public key property.

<sup>d</sup> The scheme in [10] considers only the resistance to complete key-recovery against only severely restricted adversary.

<sup>e</sup> The definition in [13] requires the hierarchy organization to be predefined in the setup of the wallet.

<sup>f</sup> The security models in [1, 7] do not capture the privilege escalation attack.

<sup>g</sup> The schemes in [1, 7] consider only forward-unlinkability, for the generated keys prior to a hot wallet breach.

<sup>h</sup> The HDW in [6] still suffers from the same security flaws as the initial HDW and the Hardened HDW in [21].

<sup>i</sup> The schemes in [15, 17] focus on stealth address, without considering the wallet properties.

mechanism does not suffer from these concerns at all. Nevertheless, SA mechanism cannot provide the hierarchy property, which is crucial to its applications in large companies and institutions, most of which are hierarchical organizations. A secure and fully-fledged HDWSA will address the shortcomings and provide all the features of HDW and SA, thus support more applications in practice.

**Security Shortfalls of Prior Wallet Schemes.** The aforementioned functionality and privacy features have led to the popularity of HDW and SA, but security is always the primary concern on these cryptographic mechanisms. Actually, the initial HDW algorithm in BIP32 [21] suffers from a fatal security flaw, namely, as pointed out in [21] and [4], once an attacker obtains a derived secret key and the master public key somehow, he could figure out the master secret key and compromise the wallet completely and steal all the related coins. Note that this is a very realistic attack (also referred to as *privilege escalation attack* [8, 9]) in various application scenarios. Since then, a series of works [8–13, 21] have attempted to address this problem. However, as shown in Table 1, the Hardened BIP32 in [21] loses the master public key property. The HDW wallet in [10] considers only the resistance to complete key-recovery against only severely restricted adversaries (which cannot query the signing oracle) rather than the standard unforgeability of signature. The wallet in [8, 9] lacks formal security analysis and actually still suffers from a similar flaw. The wallet in [13] requires the hierarchical organization to be preset in the setup phase of the wallet and

does not support transaction unlinkability<sup>3</sup>. The algorithms in [11,12] do not support the hierarchy property.

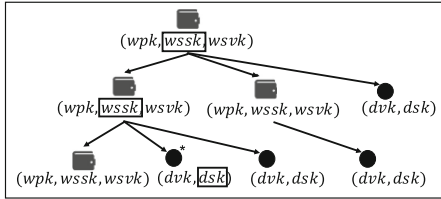
Meanwhile, it is widely accepted that *provable security*, i.e., *formal security analysis under formal definition of syntax and security models*, will provide solid confidence on the security of a cryptographic scheme. Several existing works [1,6,7,10–13] aimed to provide formal definitions and security analysis for HDW and/or SA. However, as shown in Table 1, while the formal definitions and provably secure constructions for SA have been proposed [11,12], HDW still lacks a formal definition (of syntax and security models) that captures the functionality and security requirements in practice. The latest work in providing provable security for HDW is due to Das et al. [6], which focuses on the formal analysis of BIP32 system [21]. Although the formal definition of syntax and models in [6] captures the deterministic generation property, master public key property, and hierarchy property of BIP32 HDW wallet in [21], it inherits the flaws of BIP32 wallet in [21], namely, (1) the compromising of any non-hardened node’s secret key may lead to the compromising of all nodes in the hierarchical wallet, and (2) the hardened nodes escape from this flaw but lose the master public key property.

## 1.1 Our Contribution

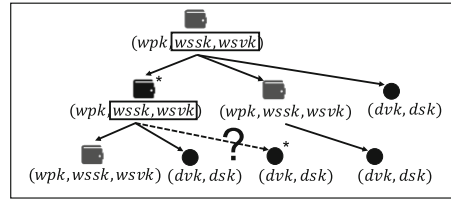
In this work, we propose a novel Hierarchical Deterministic Wallet supporting Stealth Address (HDWSA) scheme. In particular, we first formalize the syntax and the security models for HDWSA, capturing the functionality and security (including safety of coins and privacy of users) requirements that the cryptocurrency practice imposes on wallet. Then we propose a HDWSA construction and prove its security (i.e., unforgeability for safety and unlinkability for privacy) in the random oracle model. We implement our scheme and the experimental results show that it is practical for typical cryptocurrency settings. The full-fledged functionality, provable security, and practical efficiency of our HDWSA scheme will empower its applications in practice.

**System Model.** On the functionality, as the first attempt to formally define a primitive that provides the functionalities of HDW and SA simultaneously, we would like to present the system and illustrate how HDWSA works in cryptocurrency. In particular, as shown in Fig. 2, there are two layers in HDWSA, namely, the layer supporting the management of hierarchical deterministic wallets for hierarchical organizations, and the layer supporting stealth address for the wallet of each entity in an organization. In other words, from the point of view of wallet management, HDWSA is a hierarchical deterministic wallet with the hierarchy property and the deterministic generation property, and from the point of view of transactions, HDWSA provides *enhanced* master public key property

<sup>3</sup> [13] discussed how to support dynamic hierarchy but does not give formal model or proof, and discussed a method to achieve transaction unlinkability, but will lose the master public key property.



**Fig. 3.** Safety of Coins. For a target coin (e.g., the starred one), as long as the derived signing key of the coin and the wallet secret spend keys of its owner and its owner’s ancestor entities (i.e., the boxed ones) are safe, the coin is safe, even if all other keys (i.e., the non-boxed ones) in the organization are compromised.



**Fig. 4.** Privacy of Users. For a target wallet (e.g., the starred one), as long as the wallet secret key and its ancestor entities’ wallet secret keys (i.e., the boxed ones) are safe, no attacker can tell whether a target coin (e.g., the starred one) belongs to the target wallet, even if the attacker compromises all other keys (i.e., the non-boxed ones) in the organization, including the derived signing key of the target coin.

so that the wallet owners can enjoy the virtues of stealth address. More specifically, to capture the essence of hierarchical organizations (as shown later in Sect. 2.1), each entity is identified by a unique identifier  $ID = (id_0, id_1, \dots, id_t)$  with  $t \geq 0$ , and a HDWSA scheme consists of eight polynomial-time algorithms (Setup, RootWalletKeyGen, WalletKeyDelegate, VerifyKeyDerive, VerifyKeyCheck, SignKeyDerive, Sign, Verify), which, from the functionality and data-flow view points, work as follows:

- (1) The Setup() algorithm is run to generate the system parameter PP.
- (2) For any organization, the root administrator of the organization can set a unique identifier (e.g., the name of the organization)  $ID = (id_0)$  and run the RootWalletKeyGen() algorithm, generating the root wallet key pair  $(\text{wpk}_{ID}, \text{wsk}_{ID})$  of the organization.
- (3) With a wallet key pair  $(\text{wpk}_{ID}, \text{wsk}_{ID})$ , the wallet owner (i.e., the entity with identifier  $ID = (id_0, \dots, id_t)$  with  $t \geq 0$ ) can run the WalletKeyDelegate() algorithm to generate wallet key pair  $(\text{wpk}_{ID'}, \text{wsk}_{ID'})$  for its any direct subordinate with identifier  $ID' = (id_0, id_1, \dots, id_t, id_{t+1})$ , where  $id_{t+1} \in \{0, 1\}^*$  identifies a unique direct subordinate of the entity  $ID$ .
- (4) For each entity, its identifier and wallet public key will serve as the cold-address generation material. In particular, given an identifier  $ID$  and corresponding wallet public key  $\text{wpk}_{ID}$ , anyone (e.g., the payer of a transaction) can run the VerifyKeyDerive() algorithm to generate a fresh derived verification key  $\text{dvk}$ , which will be used as a coin-address for the wallet key owner (i.e., the entity with identifier  $ID$ ). Note that VerifyKeyDerive() does not need secret keys and is a randomized algorithm, namely, each time it outputs a fresh (different) derived verification key (even on input the same  $(ID,$



$wpk_{ID}$ )), so that “one coin each cold-address” is achieved in a natural and very convenient manner.

- (5) From the view of a wallet owner, say with identifier  $ID$ , the wallet secret key  $wsk_{ID}$  is divided into two parts: a *wallet secret spend key*  $wssk_{ID}$  and a *wallet secret view key*  $wsvk_{ID}$ . For any coin on the blockchain, a wallet owner can use his wallet secret view key to run the `VerifyKeyCheck()` algorithm to check whether the coin’s address  $dvk$  belongs to him, and for a derived verification key belonging to him, say  $dvk$ , the owner can use his wallet secret spend key to run `SignKeyDerive()` to generate the signing key  $dsk$  corresponding to  $dvk$ . Moreover, with the  $dsk$ , the wallet owner can run the `Sign()` algorithm to authenticate a transaction, spending the coin on  $dvk$ . Note that  $wssk_{ID}$  is more sensitive and high-value than  $wsvk_{ID}$  while  $wsvk_{ID}$  is used more frequently than  $wssk_{ID}$ , such a separation enhances the security from the point of view of practice since it greatly reduces the exposure chance of the high-value  $wssk_{ID}$ . In addition, such a separation enables our HDWSA to support the promising applications such as trust-less audits, by allowing the wallet owner to provide the wallet secret view key to the auditor while keeping the wallet secret spend key secret.<sup>4</sup>
- (6) For any coin on the blockchain, suppose the coin-address is  $dvk$ , anyone can run the `Verify()` algorithm on inputs  $dvk$  and a (transaction, signature) pair  $(tx, \sigma)$ , checking the validity of the signature, without needing (or more precisely, being able to learn) any information about the wallet/coin owner.

**HDWSA Features, Security and Efficiency.** Based on the above system model, it is easy to see that our HDWSA scheme achieves *the deterministic generation property, the (enhanced) master public key property, and the hierarchy property of HDW, and simultaneously provides the virtues of stealth address, namely the convenient fresh cold-address generation and privacy-preserving features*. Here we would like to clarify that the master public key property of our HDWSA exactly captures the essence of this property imposed by the practical applications, in the sense that *the derived verification keys (i.e., coin-addresses) are generated by using only **public** information (say user’s identifier and wallet public key) which are publicly posted in hot-storage without incurring any security concerns*. Note that this is the original motivation of the master public key property and we indeed achieved, we do not pursue the “master public key property” for wallet key generation (i.e., wallet key delegation). Actually, from the point of view of practice, it is natural for an entity to use its wallet public key and secret key to generate wallet key pairs for its direct subordinates in a safe environment (i.e., cold storage), and then each entity can publish its wallet public key and enjoy the advantage of master public key property for derived verification key (i.e., coin-address) generation.

On the security, *our HDWSA scheme achieves full resistance to privilege escalation attack*. Namely, as shown in Fig. 3 and Fig. 4, the compromising of a derived signing key will not affect the security of any other derived signing

<sup>4</sup> This separation is borrowed from the stealth address mechanisms in [12, 17].

key or wallet secret key, and the compromising of a wallet secret key will not affect the security of any derived signing key or wallet secret key except those of the compromised wallet and its direct/indirect subordinates. *The security of our HDWSA scheme is proved in the random oracle mode, based on the standard Computational Diffie-Hellman Assumption in bilinear map groups.*

On the efficiency, from the experimental results shown in Table 2 and Table 3 (in Sect. 4), we can see that the efficiency of our HDWSA scheme is lower than that of ECDSA, but is still practical for typical cryptocurrency settings. Given the versatile functionalities and provable security provided by our HDWSA scheme, such costs are reasonable and acceptable.

With the above versatile functionalities and the strong (i.e., provable) security that underlies these functionalities, our HDWSA scheme could solidly (without any security concern) support the promising use cases that have led to the popularity of HDW and SA, such as low-maintenance wallets with easy backup and recovery, convenient fresh cold-address generation, trust-less audits, treasurer allocating funds to departments in hierarchical organizations, and privacy-preservation, and so on.

## 1.2 Related Work

Table 1 gives a comprehensive comparison between our work and the existing related works, and below we would like to give further details on the comparison with the state-of-the-art HDW and SA.

When compared with the state-of-the-art HDW [6], besides providing the virtues of SA, our HDWSA can be regarded as a more *secure* HDW than the HDW in [6] and can support more promising applications. More specifically, the HDW in [6] consists of two types of nodes, say non-hardened nodes and hardened nodes, where the hardened nodes are leaf nodes of the hierarchy. If any of the non-hardened nodes is compromised, then the privilege escalation attack will work and all nodes (including the root node and all hardened nodes) will be compromised completely. The compromising of hardened nodes will not affect the security of other nodes, but the cost is that the public key generation of a hardened node requires its parent node's secret key, i.e., losing the master public key property. In addition, on the privacy, if the cold-address generation material (i.e., the public key and the chain code) of any non-hardened node is leaked, then the privacy of all its descendent non-hardened nodes is compromised. As a result, due to its vulnerability to the privilege escalation attack, the HDW in [6] cannot support the use case of treasurer allocating funds to departments (which is supposed to be the main reason that leads to HDW's popularity in hierarchical organizations), and due to the existing of hardened nodes, the HDW in [6] cannot support the use case of trust-less audits. In contrast, when working as a HDW, our HDWSA does not have these concerns at all and can support all the promising use cases that lead to the popularity of HDW in the community. It is worth mentioning that the advantage of the HDW in [6] over our HDWSA is its compatibility with ECDSA, as well as the resulting better efficiency. Actually, the above advantages and disadvantages are not surprising, since while [6] focuses

on formalizing the BIP32 HDW system and its security, *our work focuses on (1) establishing the functionality and security requirements of HDW (with SA) that underlie the use cases leading to its popularity in the community, and (2) proposing a concrete construction with provable security and practical efficiency.* Finally, we would like to point out that while the HDW in [6] only works under the assumption that the non-hardened nodes are trusted (i.e., would not attempt to compromise other nodes' secret keys) and could protect their secret keys from being compromised, our HDWSA scheme does not need such assumptions and can work in much more harsh environments.

When compared with the state-of-the-art SA [11,12], the advantage of our HDWSA is the hierarchy property, which will enable our HDWSA scheme to be applied in the hierarchical organizations (i.e., large companies and institutions) and support the use cases such as treasurer allocating funds to departments. While this work is the first one to formalize the definition and security models of HDWSA, the construction in this work seems to follow the approach of [12]. We would like to point out that this is not trivial, since supporting hierarchy property makes the formal definition, the construction, and the formal security proof pretty challenging. The effort from provably secure deterministic wallet [1,7] to provably secure *hierarchical* deterministic wallet [6] could serve as an evidence of such challenges.

### 1.3 Outline

We formalize the syntax and security models for HDWSA in Sect. 2. Then we present our HDWSA construction with its security analysis in Sect. 3. Finally we describe an implementation of our HDWSA construction in Sect. 4.

## 2 Definitions of HDWSA

In this section, we first clarify the notations of hierarchical wallet, then we formalize the syntax and security models of HDWSA.

### 2.1 Notations of Hierarchy

In this work, we use the typical hierarchical identifiers to capture the features of hierarchical organizations/wallets. In particular,

- All wallet owners are regarded as entities in hierarchical organizations.<sup>5</sup>
- Each entity in the system has a unique identifier  $ID$  in the form of  $ID = (id_0, \dots, id_t)$  with  $t \geq 0$  and  $id_i \in \{0, 1\}^*$  ( $i = 0, \dots, t$ ).

---

<sup>5</sup> Actually, an individual user can also be regarded as a special organization, for example, a user may manage his wallets in a hierarchy manner.

- For any identifier  $ID = (id_0, \dots, id_t)$  with  $t \geq 0$ , we define  $ID_{|i} := (id_0, \dots, id_i)$  for  $i = 0, \dots, t$ , and we have that (1)  $ID_{|t}$  is just  $ID$ , (2)  $ID_{|(t-1)}$  is the identifier of  $ID$ 's parent (i.e., direct supervisor) entity, (3)  $ID_{|i}$  ( $i = 0, \dots, t-1$ ) are the identifiers of  $ID$ 's ancestor entities, and (4)  $ID_{|0}$  is the identifier of the root entity (root administrator) of the organization that  $ID$  belongs to.
- For an identifier  $ID = (id_0, \dots, id_t)$  with  $t \geq 0$ , we say that the entity lies in the Level- $t$  of an organization with identifier  $ID_{|0} = (id_0)$ . Note that for any Level-0 identifier  $ID$ , it is the identifier of the root entity of some organization, and does not have parent entity.

From now on, we will denote a hierarchical organization by the identifier of its root entity, say a Level-0 identifier, e.g., “organization  $ID_0$ ”, and for an entity in some organization, we will use its identifier to denote the entity or its wallet, e.g., “ $ID$ 's wallet public key”.

## 2.2 Algorithm Definition

A HDWSA scheme consists of a tuple of algorithms (Setup, RootWalletKeyGen, WalletKeyDelegate, VerifyKeyDerive, VerifyKeyCheck, SignKeyDerive, Sign, Verify) as below:

- Setup( $\lambda$ )  $\rightarrow$  PP. On input a security parameter  $\lambda$ , the algorithm runs in polynomial time in  $\lambda$ , and outputs system public parameter PP.

*The system public parameter PP consists of the common parameters used by all entities (e.g., wallet owners, users, etc.) in the system, including the underlying groups, hash functions, and some specific rules such as the hierarchical identifier rules in Sect. 2.1, etc. Below, PP is assumed to be an implicit input to all the remaining algorithms.*

- RootWalletKeyGen( $ID$ )  $\rightarrow$  ( $wpk_{ID}, wsk_{ID}$ ). This is a randomized algorithm. On input a Level-0 identifier  $ID$ , the algorithm outputs a root (wallet public key, wallet secret key) pair ( $wpk_{ID}, wsk_{ID}$ ) for  $ID$ , where  $wsk_{ID} := (wssk_{ID}, wsvk_{ID})$  consists of a *wallet secret spend key*  $wssk_{ID}$  and a *wallet secret view key*  $wsvk_{ID}$ .

*The root administrator of each organization can run this algorithm to generate the root wallet key pair for the organization.*

- WalletKeyDelegate( $ID, wpk_{ID_{|(t-1)}}, wsk_{ID_{|(t-1)}}$ )  $\rightarrow$  ( $wpk_{ID}, wsk_{ID}$ ). This is a *deterministic* algorithm. On input an entity's identifier  $ID = (id_0, \dots, id_t)$  with  $t \geq 1$  and its parent entity's (wallet public key, wallet secret key) pair, say ( $wpk_{ID_{|(t-1)}}, wsk_{ID_{|(t-1)}}$ ), the algorithm outputs a (wallet public key, wallet secret key) pair ( $wpk_{ID}, wsk_{ID}$ ) for  $ID$ , with  $wsk_{ID} := (wssk_{ID}, wsvk_{ID})$  consisting of wallet secret spend key  $wssk_{ID}$  and wallet secret view key  $wsvk_{ID}$ .

Each entity can run this algorithm to generate wallet key pairs for its direct subordinates.

- $\text{VerifyKeyDerive}(ID, \text{wpk}_{ID}) \rightarrow \text{dvk}$ . This is a randomized algorithm. On input an entity's identifier  $ID = (id_0, \dots, id_t)$  with  $t \geq 0$  and its wallet public key  $\text{wpk}_{ID}$ , the algorithm outputs a derived verification key  $\text{dvk}$  of the entity.

Anyone can run this algorithm to generate a fresh public/verification key for an entity at  $\text{Level} \geq 0$ .

- $\text{VerifyKeyCheck}(\text{dvk}, ID, \text{wpk}_{ID}, \text{wsvk}_{ID}) \rightarrow 1/0$ . This is a deterministic algorithm. On input a derived verification key  $\text{dvk}$ , an entity's identifier  $ID = (id_0, id_1, \dots, id_t)$  with  $t \geq 0$ , and the entity's wallet public key  $\text{wpk}_{ID}$  and wallet secret view key  $\text{wsvk}_{ID}$ , the algorithm outputs a bit  $b \in \{0, 1\}$ , with  $b = 1$  meaning that  $\text{dvk}$  belongs to the entity (i.e., is a valid derived verification key generated for the entity), and  $b = 0$  otherwise.

Each entity can use this algorithm to check whether a verification key belongs to him. Note that only the wallet secret view key is needed here, rather than the whole wallet secret key.

- $\text{SignKeyDerive}(\text{dvk}, ID, \text{wpk}_{ID}, \text{wsk}_{ID}) \rightarrow \text{dsk}$  or  $\perp$ . On input a derived verification key  $\text{dvk}$ , an entity's identifier  $ID = (id_0, \dots, id_t)$  with  $t \geq 0$ , and the entity's (wallet public key, wallet secret key) pair  $(\text{wpk}_{ID}, \text{wsk}_{ID})$ , the algorithm outputs a derived signing key  $\text{dsk}$ , or  $\perp$  implying that  $\text{dvk}$  is not a valid verification key derived from  $(ID, \text{wpk}_{ID})$ .
- $\text{Sign}(m, \text{dvk}, \text{dsk}) \rightarrow \sigma$ . On input a message  $m$  in message space  $\mathcal{M}$  and a derived (verification key, signing key) pair  $(\text{dvk}, \text{dsk})$ , the algorithm outputs a signature  $\sigma$ .
- $\text{Verify}(m, \sigma, \text{dvk}) \rightarrow 1/0$ . This is a deterministic algorithm. On input a (message, signature) pair  $(m, \sigma)$  and a derived verification key  $\text{dvk}$ , the algorithm outputs a bit  $b \in \{0, 1\}$ , with  $b = 1$  meaning the validness of signature and  $b = 0$  otherwise.

**Correctness.** A HDWSA scheme must satisfy the correctness property: for any  $ID = (id_0, \dots, id_t)$  with  $t \geq 0$ , any  $0 \leq j \leq t$ , and any message  $m \in \mathcal{M}$ , suppose

$$\begin{aligned} \text{PP} &\leftarrow \text{Setup}(\lambda), (\text{wpk}_{ID_{|0}}, \text{wsk}_{ID_{|0}}) \leftarrow \text{RootWalletKeyGen}(ID_{|0}), \\ (\text{wpk}_{ID_{|i}}, \text{wsk}_{ID_{|i}}) &\leftarrow \text{WalletKeyDelegate}(ID_{|i}, \text{wpk}_{ID_{|(i-1)}}, \text{wsk}_{ID_{|(i-1)}}) \\ &\quad \text{for } i = 1, \dots, j, \\ \text{dvk} &\leftarrow \text{VerifyKeyDerive}(ID_{|j}, \text{wpk}_{ID_{|j}}), \\ \text{dsk} &\leftarrow \text{SignKeyDerive}(\text{dvk}, ID_{|j}, \text{wpk}_{ID_{|j}}, \text{wsk}_{ID_{|j}}), \end{aligned}$$

it holds that  $\text{VerifyKeyCheck}(\text{dvk}, ID_{|j}, \text{wpk}_{ID_{|j}}, \text{wsvk}_{ID_{|j}}) = 1$  and  $\text{Verify}(m, \text{Sign}(m, \text{dvk}, \text{dsk}), \text{dvk}) = 1$ .

*Remark:* The deterministic algorithm `WalletKeyDelegate()` enables the deterministic generation property and hierarchy property, and the randomized algorithm `VerifyKeyDerive()` enables the enhanced master public key property where the identifier and wallet public key of the target wallet owner serve as the cold-address generation material. Also, the algorithms `VerifyKeyDerive()`, `VerifyKeyCheck()`, and `SignKeyDerive()` together enable the features of SA.

In addition, as the hierarchical identifiers are the foundation on which the hierarchy features are built, in the above syntax, each entity, as well as its wallet, is uniquely identified by its (hierarchical) identifier. Here we would like to point out that, the binding of an entity's wallet public key and its identifier could be achieved using the standard approach of digital certificates. While the details of certificate mechanism are out of the scope of this work, here we would like to point out that, in our HDWSA, publishing the binding relation of wallet public key and its owner's (real) identifier will not cause any problem of privacy. Instead, this is an advantage of HDWSA over pure HDW and traditional wallet (in Bitcoin). In particular, in pure HDW and traditional wallet, the privacy is achieved by *artificially* hiding the real identity of a coin-address' owner, whereas in HDWSA, each entity can enjoy the convenience of (wallet) public key distribution while keeping its privacy protected, as the wallet public key is stealth in the blockchain. *For simplicity, below we will assume that each entity's wallet public key and its identifier are integrated, i.e., each wallet public key is identified by corresponding identifier.*

### 2.3 Security Models

In this section, we define the security models for HDWSA, capturing the requirements on the safety (of coins) and privacy (of users) by unforgeability and (wallet) unlinkability, respectively.

In particular, **unforgeability** is defined by the following game  $\text{Game}_{\text{EUF}}$ , which captures that, as shown in Fig. 3, for a target derived verification key  $\text{dvk}$  belonging to a target entity/wallet in some organization, as long as the corresponding derived signing key  $\text{dsk}$  is safe and the wallet secret spend keys of the target entity and its ancestor entities are safe, no attacker can forge a valid signature with respect to  $\text{dvk}$ , even if the attacker compromises all other wallet secret keys and derived signing keys in the organization.

**Definition 1.** *A HDWSA scheme is existentially unforgeable under an adaptive chosen-message attack (or just existentially unforgeable), if for all probabilistic polynomial time (PPT) adversaries  $\mathcal{A}$ , the success probability of  $\mathcal{A}$  in the following game  $\text{Game}_{\text{EUF}}$  is negligible.*

■ **Setup.**  $\text{PP} \leftarrow \text{Setup}(\lambda)$  is run and  $\text{PP}$  is given to  $\mathcal{A}$ .

An empty set  $L_{wk} = \emptyset$  is initialized, each element of which will be an (identifier, wallet public key, wallet secret key) tuple  $(ID, \text{wpk}_{ID}, \text{wsk}_{ID})$ .

An empty set  $L_{dvk} = \emptyset$  is initialized, each element of which will be a (derived verification key, identifier) pair  $(\text{dvk}, ID)$ .

Note that the two sets are defined just for the simplicity of description, and  $\mathcal{A}$  knows the  $(ID, \text{wpk}_{ID})$  pairs in  $L_{wk}$  and  $(\text{dvk}, ID)$  pairs in  $L_{dvk}$ .

$\mathcal{A}$  submits a Level-0 identifier, say  $ID_0^*$ , to trigger the setup of the target organization.  $(\text{wpk}_{ID_0^*}, \text{wsk}_{ID_0^*}) \leftarrow \text{RootWalletKeyGen}(ID_0^*)$  is run,  $\text{wpk}_{ID_0^*}$  is given to  $\mathcal{A}$ , and  $(ID_0^*, \text{wpk}_{ID_0^*}, \text{wsk}_{ID_0^*})$  is added into  $L_{wk}$ .

This captures that the adversary may manipulate the organization's identifier.

■ **Probing Phase.**  $\mathcal{A}$  can adaptively query the following oracles:

- **Wallet Key Delegate Oracle**  $\text{OWKeyDelegate}(\cdot)$ :  
On input an identifier  $ID = (id_0, \dots, id_t)$  with  $\underline{t \geq 1}$  such that  $ID_{|(t-1)} \in L_{wk}$ ,<sup>6</sup> this oracle runs  $(\text{wpk}_{ID}, \text{wsk}_{ID}) \leftarrow \text{WalletKeyDelegate}(ID, \text{wpk}_{ID_{|(t-1)}}, \text{wsk}_{ID_{|(t-1)}})$ , returns  $\text{wpk}_{ID}$  to  $\mathcal{A}$ , and sets  $L_{wk} = L_{wk} \cup (ID, \text{wpk}_{ID}, \text{wsk}_{ID})$ ,<sup>7</sup> where  $(\text{wpk}_{ID_{|(t-1)}}, \text{wsk}_{ID_{|(t-1)}})$  is the wallet key pair for  $ID_{|(t-1)}$ .  
*This captures that  $\mathcal{A}$  can trigger the wallet key delegation for any identifier  $ID$  of its choice, as long as  $ID_{|(t-1)} \in L_{wk}$ , i.e.,  $ID$ 's parent entity's wallet key pair has been generated (resp. delegated) previously due to  $\mathcal{A}$ 's trigger in the Setup phase (resp.  $\mathcal{A}$ 's query on  $\text{OWKeyDelegate}(\cdot)$ ). Note that the requirement  $ID_{|(t-1)} \in L_{wk}$  is natural, since  $ID_0^*$  is the target organization and  $\mathcal{A}$  will attack some derived verification key belonging to some entity in the organization  $ID_0^*$ .*
- **Wallet Secret Key Corruption Oracle**  $\text{OWskCorrupt}(\cdot)$ :  
On input an entity's identifier  $ID = (id_0, \dots, id_t)$  with  $\underline{t \geq 0}$ <sup>8</sup> such that  $ID \in L_{wk}$ , this oracle returns the wallet secret key  $\text{wsk}_{ID}$  of  $ID$  to  $\mathcal{A}$ .  
*This captures that  $\mathcal{A}$  can obtain the wallet secret keys for the existing wallets of its choice.*
- **Wallet Secret View Key Corruption Oracle**  $\text{OWsvkCorrupt}(\cdot)$ :  
On input an entity's identifier  $ID = (id_0, \dots, id_t)$  with  $\underline{t \geq 0}$  such that  $ID \in L_{wk}$ , this oracle returns the wallet secret view key  $\text{wsvk}_{ID}$  of  $ID$  to  $\mathcal{A}$ .  
*This captures that  $\mathcal{A}$  can obtain the wallet secret view keys for the existing wallets of its choice.*
- **Verification Key Adding Oracle**  $\text{ODVKAdd}(\cdot, \cdot)$ :  
On input a derived verification key  $\text{dvk}$  and an identifier  $ID = (id_0, \dots, id_t)$  with  $\underline{t \geq 0}$  such that  $ID \in L_{wk}$ , this oracle returns  $b \leftarrow \text{VerifyKeyCheck}(\text{dvk}, ID, \text{wpk}_{ID}, \text{wsvk}_{ID})$  to  $\mathcal{A}$ , where  $\text{wpk}_{ID}$  and  $\text{wsvk}_{ID}$  are  $ID$ 's wallet public key and wallet secret view key respectively. And if  $b = 1$ , this oracle sets  $L_{dvk} = L_{dvk} \cup (\text{dvk}, ID)$ .  
*This captures that  $\mathcal{A}$  can probe whether the derived verification keys generated by it are accepted by the owners of the target wallets.*

<sup>6</sup> Note that we are abusing the concept of ' $\in$ '. In particular, if there exists a tuple  $(ID, \text{wpk}_{ID}, \text{wsk}_{ID}) \in L_{wk}$  for some  $(\text{wpk}_{ID}, \text{wsk}_{ID})$  pair, we say that  $ID \in L_{wk}$ .

<sup>7</sup> Note that  $\text{WalletKeyDelegate}(\cdot, \cdot, \cdot)$  is a deterministic algorithm, so that querying  $\text{OWKeyDelegate}(\cdot)$  on the same identifier will obtain the same response.

<sup>8</sup> Note that actually the adversary  $\mathcal{A}$  here should not make such a query with  $t = 0$  (as required by the success conditions defined in later **Output Phase**). In later definition of unlinkability, the adversary may query this oracle on  $ID$  with  $t = 0$ .

- **Signing Key Corruption Oracle ODSKCorrupt( $\cdot$ ):**  
On input a derived verification key  $\text{dvk}$  such that there is a corresponding pair  $(\text{dvk}, ID)$  in  $L_{\text{dvk}}$ , this oracle returns  $\text{dsk} \leftarrow \text{SignKeyDerive}(\text{dvk}, ID, \text{wpk}_{ID}, \text{wsk}_{ID})$  to  $\mathcal{A}$ , where  $(\text{wpk}_{ID}, \text{wsk}_{ID})$  is the wallet key pair of  $ID$ . *This captures that  $\mathcal{A}$  can obtain the derived signing keys for the existing derived verification keys of the target wallets, of its choice.*
  - **Signing Oracle OSign( $\cdot, \cdot$ ):**  
On input a message  $m \in \mathcal{M}$  and a derived verification key  $\text{dvk} \in L_{\text{dvk}}$ <sup>9</sup>, this oracle returns  $\sigma \leftarrow \text{Sign}(m, \text{dvk}, \text{dsk})$  to  $\mathcal{A}$ , where  $\text{dsk}$  is a signing key for  $\text{dvk}$ . *This captures that  $\mathcal{A}$  can obtain the signatures for messages and derived verification keys of its choice.*
- **Output Phase.**  $\mathcal{A}$  outputs a message  $m^* \in \mathcal{M}$ , a signature  $\sigma^*$ , and a derived verification key  $\text{dvk}^* \in L_{\text{dvk}}$ .

Let  $(\text{dvk}^*, ID^*) \in L_{\text{dvk}}$ , and suppose the target wallet identifier  $ID^*$  be a Level- $t^*$  identifier, say  $ID^* = (id_0^*, \dots, id_{t^*}^*)$ .  $\mathcal{A}$  succeeds in the game if  $\text{Verify}(m^*, \sigma^*, \text{dvk}^*) = 1$  under the restrictions that (1)  $\mathcal{A}$  did not query  $\text{OWskCorrupt}(\cdot)$  on  $ID_{|i}^*$  for any  $i$  such that  $0 \leq i \leq t^*$ , (2)  $\mathcal{A}$  did not query  $\text{ODSKCorrupt}()$  on  $\text{dvk}^*$ , and (3)  $\mathcal{A}$  did not query  $\text{OSign}()$  on  $(m^*, \text{dvk}^*)$ .

*Remark:* It is worth mentioning that  $\mathcal{A}$  is allowed to query the wallet secret view keys for  $ID_{|i}^*$  ( $i = 0, \dots, t^*$ ) by  $\text{OWsvkCorrupt}(\cdot)$ . Note this will guarantee the safety of coins when the trust-less audits functionality is employed.

As the above unforgeability model captures the attackers' ability exactly and completely, below we also present a weaker unforgeability model, where the adversary is required to commit its target wallet's identifier (i.e.,  $ID^*$ , which the target derived verification key  $\text{dvk}^*$  belongs to) in ahead.

**Definition 2.** *A HDWSA scheme is existentially unforgeable under an adaptive chosen-message attack with selective wallet (or just selective wallet existentially unforgeable), if for all PPT adversaries  $\mathcal{A}$ , the success probability of  $\mathcal{A}$  in the following game  $\text{Game}_{\text{swEUF}}$  is negligible.*

The game  $\text{Game}_{\text{swEUF}}$  is identical to the above game  $\text{Game}_{\text{EUF}}$ , except that the adversary commits its target wallet in the **Setup** phase. More specifically, just before the start of **Probing Phase**, the adversary  $\mathcal{A}$  commits the identifier of its target wallet, say,  $ID^* = (id_0^*, \dots, id_{t^*}^*)$  such that  $t^* \geq 0$  and  $ID_{|0}^* = ID_0^*$ , committing that the target derived verification key  $\text{dvk}^*$  in the **Output Phase** will be one belonging to  $ID^*$ .

The **wallet unlinkability** is defined by the following game  $\text{Game}_{\text{wUNL}}$ , which captures that, the adversary is unable to identify the wallet, out of two wallets, from which a target derived verification key was generated from, whatever the two wallets belong to the same organization or different organizations. Note that such an "indistinguishability" model captures the intuition in Fig. 4 well.

<sup>9</sup> Note that we are abusing the concepts of ' $\in$ '. In particular, if there exists a pair  $(\text{dvk}, ID) \in L_{\text{dvk}}$  for some  $ID$ , we say that  $\text{dvk} \in L_{\text{dvk}}$ .



**Definition 3.** A HDWSA scheme is wallet unlinkable, if for all PPT adversaries  $\mathcal{A}$ , the advantages of  $\mathcal{A}$  in the following game  $\text{Game}_{\text{WUNL}}$ , denoted by  $\text{Adv}_{\mathcal{A}}^{\text{WUNL}}$ , is negligible.

■ **Setup.**  $\text{PP} \leftarrow \text{Setup}(\lambda)$  is run and  $\text{PP}$  is given to  $\mathcal{A}$ .

As in the **Setup** phase of  $\text{Game}_{\text{EUF}}$ ,  $L_{wk} = \emptyset$  and  $L_{dvk} = \emptyset$  are initialized.

$\mathcal{A}$  submits two different Level-0 identifiers, say  $ID_0^{*(0)}$  and  $ID_0^{*(1)}$ .

For  $k = 0, 1$ :  $(\text{wpk}_{ID_0^{*(k)}}, \text{wsk}_{ID_0^{*(k)}}) \leftarrow \text{RootWalletKeyGen}(ID_0^{*(k)})$  is run,  $\text{wpk}_{ID_0^{*(k)}}$  is given to  $\mathcal{A}$ , and  $(ID_0^{*(k)}, \text{wpk}_{ID_0^{*(k)}}, \text{wsk}_{ID_0^{*(k)}})$  is added into  $L_{wk}$ .

*This captures that the adversary may manipulate the organizations' identifiers.*

■ **Probing Phase 1.** Same as the **Probing Phase** of  $\text{Game}_{\text{EUF}}$ .

■ **Challenge.**  $\mathcal{A}$  submits two different challenge wallets' identifiers  $ID^{(0)} = (id_0^{(0)}, \dots, id_{t^{(0)}}^{(0)})$  and  $ID^{(1)} = (id_0^{(1)}, \dots, id_{t^{(1)}}^{(1)})$ , such that  $t^{(0)} \geq 0, t^{(1)} \geq 0$ , and  $ID^{(0)}, ID^{(1)} \in L_{wk}$  (implying  $ID_{|0}^{(0)}, ID_{|0}^{(1)} \in \{ID_0^{*(0)}, ID_0^{*(1)}\}$ ).

A random bit  $c \in \{0, 1\}$  is chosen,  $\text{dvk}^* \leftarrow \text{VerifyKeyDerive}(ID^{(c)}, \text{wpk}_{ID^{(c)}})$  is given to  $\mathcal{A}$ . And  $(\text{dvk}^*, ID^{(c)})$  is added into  $L_{dvk}$ .

■ **Probing Phase 2.** Same as **Probing Phase 1**.

■ **Guess.**  $\mathcal{A}$  outputs a bit  $c' \in \{0, 1\}$  as its guess to  $c$ .

$\mathcal{A}$  succeeds in the game if  $c' = c$  under the restrictions that (1)  $\mathcal{A}$  did not query  $\text{OWskCorrupt}(\cdot)$  or  $\text{OWsvkCorrupt}(\cdot)$  oracle on any  $ID \in \{ID_{|i}^{(0)} \mid 0 \leq i \leq t^{(0)}\} \cup \{ID_{|i}^{(1)} \mid 0 \leq i \leq t^{(1)}\}$ , and (2)  $\mathcal{A}$  did not query oracle  $\text{ODVAdd}(\cdot, \cdot)$  on  $(\text{dvk}^*, ID^{(0)})$  or  $(\text{dvk}^*, ID^{(1)})$ . The advantage of  $\mathcal{A}$  is  $\text{Adv}_{\mathcal{A}}^{\text{WUNL}} = |\Pr[c' = c] - \frac{1}{2}|$ .

*Remark:* Note that the adversary is allowed to choose the challenge identifiers  $ID^{(0)}$  and  $ID^{(1)}$  of its choice completely, namely, they could be from same or different organizations, from same or different levels, or one could be an ancestor of another. Note that the adversary is allowed to query the  $\text{ODskCorrupt}()$  and  $\text{OSign}()$  oracles on the challenge derived verification key  $\text{dvk}^*$ , and this captures that neither the signature nor the derived signing key leaks the privacy of the owner of  $\text{dvk}^*$ . It is also worth noticing that, while the adversary in  $\text{Game}_{\text{EUF}}$  should not query  $\text{OWskCorrupt}(\cdot)$  on an identifier  $ID = (id_0, \dots, id_t)$  with  $t = 0$  such that  $ID \in L_{wk}$  (since it means corrupting the root wallet secret key of the target organization  $ID_0^*$ ), the adversary in  $\text{Game}_{\text{WUNL}}$  may query  $\text{OWskCorrupt}(\cdot)$  and/or  $\text{OWsvkCorrupt}(\cdot)$  on an identifier  $ID = (id_0, \dots, id_t)$  with  $t = 0$  such that  $ID \in L_{wk}$ , depending on its challenge wallet identifier pair  $(ID^{(0)}, ID^{(1)})$ . In particular, if  $ID_{|0}^{(0)} = ID_{|0}^{(1)} = ID_0^{*(k)}$ , then the adversary is allowed to query  $\text{OWskCorrupt}(\cdot)$  and/or  $\text{OWsvkCorrupt}(\cdot)$  on  $ID_0^{*(1-k)}$ .

As the above unlinkability model captures the attackers' ability exactly and completely, below we also present a weaker unlinkability model, where the adversary is required to commit its challenge wallets in ahead.

**Definition 4.** A HDWSA scheme is selective wallet unlinkable, if for all PPT adversaries  $\mathcal{A}$ , the advantage of  $\mathcal{A}$  in the following games  $\text{Game}_{\text{swWUNL}}$ , denoted by  $\text{Adv}_{\mathcal{A}}^{\text{swWUNL}}$ , is negligible.

The game  $\text{Game}_{\text{swWUNL}}$  is identical to the above game  $\text{Game}_{\text{WUNL}}$ , except that the adversary commits the two challenge wallet identifiers in the **Setup** phase. More specifically, just before the start of **Probing Phase 1**, the adversary  $\mathcal{A}$  commits the two challenge wallet identifiers, namely,  $ID^{(0)} = (id_0^{(0)}, \dots, id_{t^{(0)}}^{(0)})$  and  $ID^{(1)} = (id_0^{(1)}, \dots, id_{t^{(1)}}^{(1)})$  such that  $t^{(0)} \geq 0, t^{(1)} \geq 0$ , and  $ID_{|0}^{(0)}, ID_{|0}^{(1)} \in \{ID_0^{*(0)}, ID_0^{*(1)}\}$ .

### 3 Our Construction

In this section, we first review some preliminaries, including the bilinear groups and CDH assumption. Then we propose our HDWSA construction.

#### 3.1 Preliminaries

**Bilinear Map Groups [3].** Let  $\lambda$  be a security parameter and  $p$  be a  $\lambda$ -bit prime number. Let  $\mathbb{G}_1$  be an additive cyclic group of order  $p$ ,  $\mathbb{G}_2$  be a multiplicative cyclic group of order  $p$ , and  $P$  be a generator of  $\mathbb{G}_1$ .  $(\mathbb{G}_1, \mathbb{G}_2)$  are bilinear groups if there exists a bilinear map  $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$  satisfying the following properties:

1. Bilinearity:  $\forall (S, T) \in \mathbb{G}_1 \times \mathbb{G}_1, \forall a, b \in \mathbb{Z}, \hat{e}(aS, bT) = \hat{e}(S, T)^{ab}$ .
2. Non-degeneracy:  $\hat{e}(P, P) \neq 1$ .
3. Computable:  $\forall (S, T) \in \mathbb{G}_1 \times \mathbb{G}_1, \hat{e}(S, T)$  is efficiently computable.

**Definition 5 (Computational Diffie-Hellman (CDH) Assumption [20]).** The CDH problem in bilinear groups  $(p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e})$  is defined as follows: given  $(P, aP, bP) \in \mathbb{G}_1^3$  as input, output an element  $C \in \mathbb{G}_1$  such that  $C = abP$ . An algorithm  $\mathcal{A}$  has advantage  $\epsilon$  in solving CDH problem in  $(p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e})$  if  $\Pr[\mathcal{A}(P, aP, bP) = abP] \geq \epsilon$ , where the probability is over the random choice of  $a, b \in \mathbb{Z}_p$  and the random bits consumed by  $\mathcal{A}$ .

We say that the  $(t, \epsilon)$ -CDH assumption holds in  $(p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e})$  if no  $t$ -time algorithm has advantage at least  $\epsilon$  in solving the CDH problem in  $(p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e})$ .

#### 3.2 Construction

- **Setup**( $\lambda$ )  $\rightarrow$  PP. On input a security parameter  $\lambda$ , the algorithm chooses bilinear groups  $(p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e})$  and cryptographic hash functions  $H_0 : \mathcal{S}_{ID} \rightarrow \mathbb{G}_1^*$ ,  $H_1 : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{Z}_p^*$ ,  $H_2 : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{Z}_p^*$ ,  $H_3 : \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_1^*$ , and  $H_4 : (\mathbb{G}_1 \times \mathbb{G}_2) \times \mathcal{M} \times \mathbb{G}_2 \rightarrow \mathbb{Z}_p^*$ , where  $\mathbb{G}_1^* = \mathbb{G}_1 \setminus \{0\}$ ,  $\mathcal{M} = \{0, 1\}^*$ , and  $\mathcal{S}_{ID} := \{ID = (id_0, id_1, \dots, id_t) \mid t \geq 0, id_i \in \{0, 1\}^* \forall 0 \leq i \leq t\}$ . The algorithm outputs public parameter PP =  $((p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e}), H_0, H_1, H_2, H_3, H_4)$ , where the message space is  $\mathcal{M}$  and the identifier space is  $\mathcal{S}_{ID}$ .

PP is assumed to be an implicit input to all the algorithms below.

- $\text{RootWalletKeyGen}(ID) \rightarrow (\text{wpk}_{ID}, \text{wsk}_{ID})$ . On input a Level-0 identifier  $ID$ , the algorithm chooses uniformly random  $\alpha_{ID}, \beta_{ID} \in \mathbb{Z}_p^*$ , then outputs a (root) waller key pair  $(\text{wpk}_{ID}, \text{wsk}_{ID})$ :

$$\begin{aligned}\text{wpk}_{ID} &:= (A_{ID}, B_{ID}) = (\alpha_{ID}P, \beta_{ID}P) \in \mathbb{G}_1 \times \mathbb{G}_1, \\ \text{wsk}_{ID} &:= (\text{wssk}_{ID}, \text{wsvk}_{ID}) = (\alpha_{ID}, \beta_{ID}) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*.\end{aligned}$$

- $\text{WalletKeyDelegate}(ID, \text{wpk}_{ID_{|(t-1)}}, \text{wsk}_{ID_{|(t-1)}}) \rightarrow (\text{wpk}_{ID}, \text{wsk}_{ID})$ . On input an entity's identifier  $ID = (id_0, \dots, id_t)$  with  $t \geq 1$  and its parent entity's (wallet public key, wallet secret key) pair, say  $\text{wpk}_{ID_{|(t-1)}} \in \mathbb{G}_1 \times \mathbb{G}_1$  and  $\text{wsk}_{ID_{|(t-1)}} = (\alpha_{ID_{|(t-1)}}, \beta_{ID_{|(t-1)}}) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ , the algorithm proceeds as below:
  1. Compute  $Q_{ID} = H_0(ID) \in \mathbb{G}_1^*$ ,
  2. Compute  $\alpha_{ID} = H_1(Q_{ID}, \alpha_{ID_{|(t-1)}}Q_{ID})$  and  $\beta_{ID} = H_2(Q_{ID}, \beta_{ID_{|(t-1)}}Q_{ID})$ ,
  3. Output wallet key pair  $(\text{wpk}_{ID}, \text{wsk}_{ID})$  for  $ID$  as

$$\begin{aligned}\text{wpk}_{ID} &:= (A_{ID}, B_{ID}) = (\alpha_{ID}P, \beta_{ID}P) \in \mathbb{G}_1 \times \mathbb{G}_1, \\ \text{wsk}_{ID} &:= (\text{wssk}_{ID}, \text{wsvk}_{ID}) = (\alpha_{ID}, \beta_{ID}) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*.\end{aligned}$$

- $\text{VerifyKeyDerive}(ID, \text{wpk}_{ID}) \rightarrow \text{dvk}$ . On input an identifier  $ID = (id_0, \dots, id_t)$  with  $t \geq 0$  and its wallet public key  $\text{wpk}_{ID} = (A_{ID}, B_{ID})$ , the algorithm chooses a uniformly random  $r \in \mathbb{Z}_p^*$  and outputs a derived verification key  $\text{dvk} := (Q_r, Q_{vk})$  with  $Q_r = rP \in \mathbb{G}_1$ ,  $Q_{vk} = \hat{e}(H_3(B_{ID}, rP, rB_{ID}), -A_{ID}) \in \mathbb{G}_2$ .
- $\text{VerifyKeyCheck}(\text{dvk}, ID, \text{wpk}_{ID}, \text{wsvk}_{ID}) \rightarrow 1/0$ . On input a derived verification key  $\text{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ , an entity's identifier  $ID = (id_0, \dots, id_t)$  with  $t \geq 0$ , and the entity's wallet public key  $\text{wpk}_{ID} = (A_{ID}, B_{ID}) \in \mathbb{G}_1 \times \mathbb{G}_1$  and wallet secret view key  $\text{wsvk}_{ID} = \beta_{ID} \in \mathbb{Z}_p^*$ , the algorithm checks whether  $Q_{vk} \stackrel{?}{=} \hat{e}(H_3(B_{ID}, Q_r, \beta_{ID}Q_r), -A_{ID})$  holds. If it does not hold, the algorithm outputs 0, otherwise outputs 1.
- $\text{SignKeyDerive}(\text{dvk}, ID, \text{wpk}_{ID}, \text{wsk}_{ID}) \rightarrow \text{dsk}$  or  $\perp$ . On input a derived verification key  $\text{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ , an entity's identifier  $ID = (id_0, \dots, id_t)$  with  $t \geq 0$ , and the entity's (wallet public key, wallet secret key) pair, say  $\text{wpk}_{ID} = (A_{ID}, B_{ID}) \in \mathbb{G}_1 \times \mathbb{G}_1$  and  $\text{wsk}_{ID} = (\alpha_{ID}, \beta_{ID}) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ , the algorithm checks whether  $Q_{vk} \stackrel{?}{=} \hat{e}(H_3(B_{ID}, Q_r, \beta_{ID}Q_r), -A_{ID})$  holds. If it does not hold, the algorithm outputs  $\perp$ , otherwise outputs a derived signing key  $\text{dsk}$

$$\text{dsk} = \alpha_{ID}H_3(B_{ID}, Q_r, \beta_{ID}Q_r) \in \mathbb{G}_1.$$

- $\text{Sign}(m, \text{dvk}, \text{dsk}) \rightarrow \sigma$ . On input a message  $m$  in message space  $\mathcal{M}$ , a derived verification key  $\text{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ , and a derived signing key  $\text{dsk} \in \mathbb{G}_1$ , the algorithm proceeds as below:
  1. Choose a uniformly random  $x \in \mathbb{Z}_p^*$ , then compute  $X = \hat{e}(xP, P) \in \mathbb{G}_2$ .
  2. Compute  $h = H_4(\text{dvk}, m, X) \in \mathbb{Z}_p^*$  and  $Q_\sigma = h \cdot \text{dsk} + xP \in \mathbb{G}_1$ .
  3. Output  $\sigma = (h, Q_\sigma) \in \mathbb{Z}_p^* \times \mathbb{G}_1$  as the signature for  $m$ .

**Table 2.** Sizes of signature and keys of an implementation of our HDWSA scheme

Derived verification key	Signature	Wallet public key	Wallet secret key	Derived signing key
193 Bytes	97 Bytes	130 Bytes	64 Bytes	65 Bytes

**Table 3.** Computation time of an implementation of our HDWSA scheme

Setup	RootWalletKeyGen	WalletKeyDelegate	VerifyKeyDerive	VerifyKeyCheck	SignKeyDerive	Sign	Verify
7.769 ms	0.920 ms	4.111 ms	3.352 ms	3.122 ms	3.322 ms	1.505 ms	0.665 ms

- $\text{Verify}(m, \sigma, \text{dvk}) \rightarrow 1/0$ . On input a (message, signature) pair  $(m, \sigma)$  with  $\sigma = (h, Q_\sigma) \in \mathbb{Z}_p^* \times \mathbb{G}_1$  and a derived verification key  $\text{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ , the algorithm checks whether  $h \stackrel{?}{=} H_4(\text{dvk}, m, \hat{e}(Q_\sigma, P) \cdot (Q_{vk})^h)$  holds. If it holds, the algorithm outputs 1, otherwise outputs 0.

**Correctness.** The correctness of the construction can be easily verified. Due to space limitation, we defer the details to the full version [22].

### 3.3 Security Analysis

Due to space limitation, we only give the security conclusion here and readers are referred to the full version [22, Appendix B] for the proof details.

**Theorem 1.** *The HDWSA scheme is selective wallet existentially unforgeable under the CDH assumption in the random oracle model.*

**Theorem 2.** *The HDWSA scheme is selective wallet unlinkable under the CDH assumption in the random oracle model.*

**Unforgeability and Unlinkability in the Adaptive Model.** Following Boneh et al.’s approach for achieving full security of Hierarchical Identity Based Encryption (HIBE) construction [2], our HDWSA construction can be proven unforgeable in the adaptive model, at the cost of a reduction loss factor of  $\frac{1}{h+1} \frac{1}{q_{H_0}^h}$ , where  $q_{H_0}$  is the number of hash oracle queries to  $H_0$  and  $h$  is the maximum level of identifiers/entities in an organization. Note that for hierarchical systems, it is natural to set such a parameter  $h$ , and in a practical organization hierarchy,  $h$  would be a small integer. We give more details of the reduction in the full version [22, Sect. 4.3]. The proof for unlinkability in the adaptive model is similar.

## 4 Implementation

We implemented our HDWSA scheme in Golang and the source codes are available at <https://github.com/cryptoscheme/hdwsa>. Our implementation uses the Pairing-Based Cryptography Library [19], and uses a type A pairing on elliptic curve  $y^2 = x^3 + x$  over  $F_q$  for a 512-bit  $q$  and group with 160-bit prime order, implying 80-bit security. Our implementation uses SHA-256 to implement the hash functions.

Table 2 and Table 3 show the experimental results of our implementation on a usual computation environment, namely, a desktop with Intel(R) Core(TM) i7 10700 CPU @2.90GHz., 16 GB memory, and operating system Ubuntu 20.04 LTS. We clarify that our implementation was simply experimental and did not optimize further. We clarify further that we use point compression in computing the size of element in group  $\mathbb{G}_1$ .

**Acknowledgements.** This work was supported by the National Natural Science Foundation of China (No. 62072305, 62132013), and Shanghai Key Laboratory of Privacy-Preserving Computation.

## References

1. Alkadri, N.A., et al.: Deterministic wallets in a quantum world. In: CCS 2020, pp. 1017–1031 (2020)
2. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005). [https://doi.org/10.1007/11426639\\_26](https://doi.org/10.1007/11426639_26)
3. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-44647-8\\_13](https://doi.org/10.1007/3-540-44647-8_13)
4. Buterin, V.: Deterministic wallets, their advantages and their understated flaws. Bitcoin Magazine (2013)
5. ByteCoin: Untraceable transactions which can contain a secure message are inevitable (2011). <https://bitcointalk.org/index.php?topic=5965.0>
6. Das, P., Erwig, A., Faust, S., Loss, J., Riahi, S.: The exact security of BIP32 wallets. In: CCS 2021, pp. 1020–1042 (2021)
7. Das, P., Faust, S., Loss, J.: A formal treatment of deterministic wallets. In: CCS 2019, pp. 651–668 (2019)
8. Fan, C., Tseng, Y., Su, H., Hsu, R., Kikuchi, H.: Secure hierarchical bitcoin wallet scheme against privilege escalation attacks. In: IEEE Conference on Dependable and Secure Computing, DSC 2018, pp. 1–8 (2018)
9. Fan, C., Tseng, Y., Su, H., Hsu, R., Kikuchi, H.: Secure hierarchical bitcoin wallet scheme against privilege escalation attacks. Int. J. Inf. Sec. **19**(3), 245–255 (2020)
10. Gutoski, G., Stebila, D.: Hierarchical deterministic bitcoin wallets that tolerate key leakage. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 497–504. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-47854-7\\_31](https://doi.org/10.1007/978-3-662-47854-7_31)
11. Liu, W., Liu, Z., Nguyen, K., Yang, G., Yu, Yu.: A lattice-based key-insulated and privacy-preserving signature scheme with publicly derived public key. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12309, pp. 357–377. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-59013-0\\_18](https://doi.org/10.1007/978-3-030-59013-0_18)

12. Liu, Z., Yang, G., Wong, D.S., Nguyen, K., Wang, H.: Key-insulated and privacy-preserving signature scheme with publicly derived public key. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 215–230. IEEE (2019)
13. Di Luzio, A., Francati, D., Ateniese, G.: Arcula: a secure hierarchical deterministic wallet for multi-asset blockchains. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 2020. LNCS, vol. 12579, pp. 323–343. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-65411-5\\_16](https://doi.org/10.1007/978-3-030-65411-5_16)
14. NIST: FIPS pub 186-4. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>. Accessed 10 Jan 2021
15. Noether, S., Mackenzie, A.: Ring confidential transactions. *Ledger* **1**, 1–18 (2016)
16. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **21**(2), 120–126 (1978)
17. van Saberhagen, N.: Cryptonote v 2.0 (2013). <https://cryptonote.org/whitepaper.pdf>
18. Todd, P.: Stealth addresses. Post on Bitcoin development mailing list (2014). <https://www.mail-archive.com/bitcoindevelopment@lists.sourceforge.net/msg03613.html>
19. Unger, N.: The PBC go wrapper, December 2018. <https://github.com/Nik-U/pbc>
20. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005). [https://doi.org/10.1007/11426639\\_7](https://doi.org/10.1007/11426639_7)
21. Wuille, P.: BIP32: hierarchical deterministic wallets (2012). <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>
22. Yin, X., Liu, Z., Yang, G., Chen, G., Zhu, H.: Secure Hierarchical Deterministic Wallet Supporting Stealth Address. *Cryptology ePrint Archive*, Paper 2022/627 (2022). <https://eprint.iacr.org/2022/627>