Secure Hierarchical Deterministic Wallet Supporting Stealth Address

Xin Yin$^{1[0000-0001-8761-183X]}$, Zhen Liu$^{1[0000-0001-9268-702X]}$, Guomin Yang$^{2[0000-0002-4949-7738]}$, Guoxing Chen$^{1[0000-0001-8107-5909]}$, and Haojin Zhu$^{1[0000-0001-5079-4556]}$

[1] Shanghai Jiao Tong University, China
{yinxin,liuzhen,guoxingchen,zhu-hj}@sjtu.edu.cn
[2] University of Wollongong, Australia
gyang@uow.edu.au

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, 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. As the name "cryptocurrency" implies, cryptography plays a crucial role in cryptocurrencies. Particularly, Digital Signature [21, 19] 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 σ such that (tx, σ) is a valid (message, signature) pair with respect to the public key 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. Different from the key management in pure cryptography systems, the key management in cryptocurrency 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 (Bitcoin Improvement Proposal 32) [28], has been accepted as a standard in the Bitcoin community. It is so popular that almost each cryptocurrency has implemented or is planing to implement a HDW. 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, which are used to sign and authenticate the transactions of spending the coins on the derived public 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. In particular, the deterministic generation property allows the low-maintenance wallets with easy backup and recovery, the master public key property supports convenient cold-address generation and enables the use case of trust-less audits, and the hierarchy property enables the use case of treasurer allocating funds to departments and gives HDW great potential to be applied in hierarchical organizations (e.g., large companies and institutions). Readers are referred to [28, 15, 9, 17] for the details of these use cases.

"One Coin Each Cold-address" via HDW 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. Namely, it recommends to store only a small amount of cryptocurrency coins on hot addresses while transfer large amounts of coins to cold-addresses. 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 (e.g., a server connected to Internet), 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 [28, 10, 11, 18, 9, 1, 8], 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 (with standard key generation mechanism), 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. Consequently, many users would hesitate to adopt this mechanism and still use one coin-address to handle multiple or even many coins (referred to as address-reuse). This will definitely weaken the users' privacy and even may weaken the safety of their coins. 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 (since the corresponding secret keys will be generated only when they are needed for signing transactions). Therefore, a wallet owner will be very willing to generate a fresh derived public key (i.e., a new cold-address) for each payment (i.e., each coin), achieving "one coin each cold-address" [3] efficiently.

[3] 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. More specifically, *"one coin each cold-address"* mechanism means that, each address is used to host only one coin (i.e., no address-reuse), and before

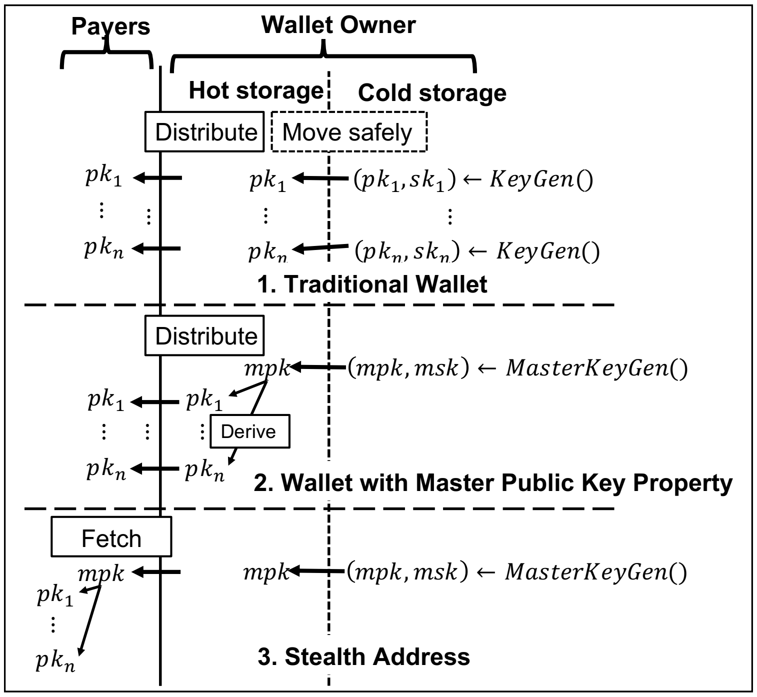
Payers
Wallet Owner
Hot storage
Cold storage
Distribute
Move safely
pk_1
pk_n
$(pk_1, sk_1) \leftarrow KeyGen()$
$(pk_n, sk_n) \leftarrow KeyGen()$
1. Traditional Wallet
Distribute
mpk
$(mpk, msk) \leftarrow MasterKeyGen()$
Derive
2. Wallet with Master Public Key Property
Fetch
mpk
$(mpk, msk) \leftarrow MasterKeyGen()$
3. Stealth Address

Fig. 1. Cold Address Generation and Distribution.

Stealth Address for Efficient, Non-interactive and Privacy-preserving Payment. While HDW focuses on the key generation/derivation and management from the view point of a wallet owner managing his wallet (i.e., keys), *stealth address* (SA), introduced by [6, 23, 24], is another popular key-derivation mechanism in the community, which is also related to key generation and management, but from the perspective of privacy-preservation when transferring 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 and corresponding coin/signature/transaction 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 [4]), 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"*. As a convenient way to protect user privacy, say achieving transaction unlinkability, SA has attracted much attention and been widely used in the community, for example in Monero [20, 13], which is one of the most popular privacy-centric cryptocurrencies.

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 generation/derivation and management and both take convenient cold-address generation as an important virtue,

the coin is spent, the address keeps to be cold, i.e., the address becomes hot only after the coin is spent. As a result, ideally, even if the secret key for a coin-address is compromised (note that this may happen only when the secret key is exposed in hot storage to spend the coin), it is useless to the attackers, since the only coin on the address has been spent.

[4] Note that the payee can detect such malicious behaviors easily.

Table 1. Comparison with Existing Works

Scheme [1]	Support Master Public Key Property?	Support Hierarchy Property?	Support Stealth Address?	Secure again privilege escalation attack?	Privacy-Preserving when cold-address generation material (in hot storage) compromised?	Formally Modeled & Proved?
[28]	✓	✓	✗	✗	✗	✗
Hardened [28]	✗	✓	✗	✓	— [2]	✗
[15]	✓	✓	✗	Partially [3]	✗	Partially [3]
[10, 11]	✓	✓	✗	✓	✗	✗
[18]	✓	Partially [4]	✗	✓	✗	✓
[9, 1] (CCS'19, '20)	✓	✗	✗	— [5]	Partially [6]	✓
[8] (CCS'21)	✓	✓	✗	Partially [7]	✗ [7]	✓
[23, 20]	— [8]	— [8]	✓	✗	✓	✗
[17, 16]	✓	✗	✓	✓	✓	✓
This work	✓	✓	✓	✓	✓	✓

[1] All schemes in this table support deterministic generation property.
[2] The Hardened BIP32 in [28] does not support convenient cold-address generation, since it loses the master public key property.
[3] The scheme in [15] considers only the resistance to complete key-recovery against only severely restricted adversary.
[4] The definition in [18] requires the hierarchy organization to be predefined in the setup of the wallet.
[5] The security models in [9, 1] do not capture the attack of derived secret key compromising (i.e., the privilege escalation attack).
[6] The schemes in [9, 1] consider only forward-unlinkability, for the generated keys prior to a hot wallet breach.
[7] The HDW in [8] still suffers from the same security flaws as the initial HDW and the Hardened HDW in [28].
[8] The schemes in [23, 20] focus on stealth address, without considering the wallet properties.

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 Hierarchical Deterministic Wallet supporting Stealth Address is well motivated by realistic scenarios, and how existing schemes fail to achieve both security and full functionality. In particular, as shown in [28, 15, 9, 1, 8], master public key property enables convenient cold-address generation, thus supports "one coin each cold-address", and finally helps achieve transaction unlinkability, which is one of the most important features that the users in practice are interested in. 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. Note that the first issue may cause concerns not only on convenience but also on privacy, since the relation between the cold-addresses and the wallet owner may be leaked somehow during the distribution. As for the second issue, for the existing typical HDW algorithms [28, 15, 9, 1, 8], if the cold-address generation material is leaked, the privacy (i.e., the transaction unlinkability) is compromised completely (e.g., [28, 15, 8]) or partially (e.g., [9, 1]). In contrast, as shown in Fig. 1, SA mechanism does not suffer from these concerns at all, namely, the owner of a master public key only needs to publish his master public key, then any one (including the payers and the wallet owner himself) can generate fresh cold-addresses from the master public key, *without needing any interaction or distribution*, and as the master public key serves as the only *published* cold-address generation material, there are no concerns on the leakage of cold-address generation material. *Thus, when compared with HDW, HDWSA will provide deterministic generation property and hierarchy property as HDW does, and will provide* ***enhanced*** *master public key property, which enables much better convenience and privacy; and when compared with SA, HDWSA will provide hierarchy property, which is crucial to its applications in large companies and institutions, most of which are hierarchical organizations. These features will enable HDWSA to support more applications in practice than standalone HDW and SA.*

Security Shortfalls of Prior Wallet Schemes. As the above mentioned functionality features lead to the popularity of HDW and SA respectively, security is always the primary concern on these cryptographic mechanisms. Actually, the initial HDW algorithm in BIP32 [28] suffers from a fatal security flaw, namely, as pointed out in [28] and [5], 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* [10, 11]), for example, in practice, in the use case of trust-less audits, the master public key will be given to an auditor, and in the use case of treasurer allocating funds to departments, derived secret keys will be distributed to the department managers, so that the auditor and some department manager may collude to launch such an attack. Also, from the cryptographic design or fault tolerant perspective, this issue is unsatisfactory, since as pointed out by Liu et al.[17], in essence it means that, when a minor fault happens (say, one derived key is compromised somehow), the damage is not limited to the leaked derived key only. Instead, it renders the master key and all derived keys compromised. Since then, a series of works [28, 15, 10, 11, 17, 16, 18] have attempted to address this problem. However, as shown in Table 1, the Hardened BIP32 in [28] loses the master public key property (i.e., the parent secret key is required when generating a derived public key). The HDW wallet in [15] 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 [10, 11] lacks formal security analysis and actually still suffers from a similar flaw. The wallet in [18] requires the hierarchical organization to be preset in the setup phase of the wallet and does not support transaction unlinkability [5]. The algorithms in [17, 16] do not support the hierarchy property.

On the other side, based on the wide acceptance that *provable security*, i.e., *formal security analysis under formal definition of syntax and security models*, will provide solid confidence on the security, existing works [15, 17, 9, 16, 18, 1, 8] focus on providing 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 [17, 16], HDW still lacks a formal definition (of syntax and security models) that captures the functionality and security requirements in practice. In particular, the scheme in [15] considers only a model for resistance to complete key-recovery against only severely restricted adversary; the schemes in [9, 1] consider only a weak unforgeability model (without considering the compromising of derived secret keys, i.e., the realistic privilege escalation attack considered in [28, 15, 10, 11, 17, 16, 18]) and a weak unlinkability model (referred to as forward unlinkability, meaning that only the generated keys prior to a hot wallet breach cannot be linked to the master public key), and does not consider the hierarchy property; the definition in [18] requires the hierarchical organization to be preset in the setup phase of the wallet and does not support privacy-preservation (namely, once a coin is spent, the corresponding signature will be linked to the wallet); the schemes in [17, 16] do not support the hierarchy property. The latest work in providing provable security for HDW is due to Das et al. [8], which focuses on the formal analysis of BIP32 system [28]. Although the formal definition of syntax and models in [8] captures the deterministic generation property, master public key property, and hierarchy property of BIP32 HDW wallet in [28], it inherits the flaws of BIP32 wallet in [28], 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.

[5] [18] 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.

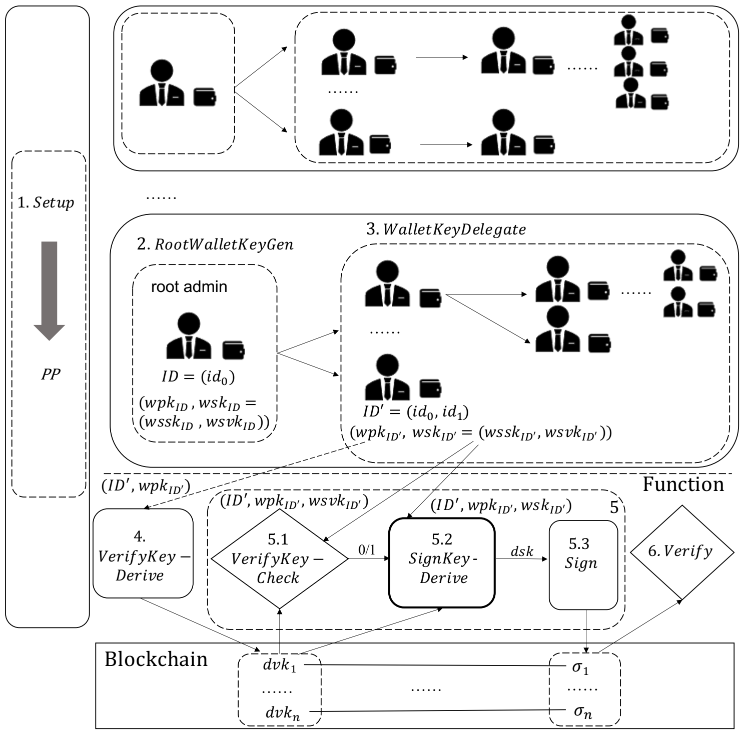

Fig. 2. System model.

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

- (1) The $\mathsf{Setup}()$ algorithm is run to generate the system parameters PP.

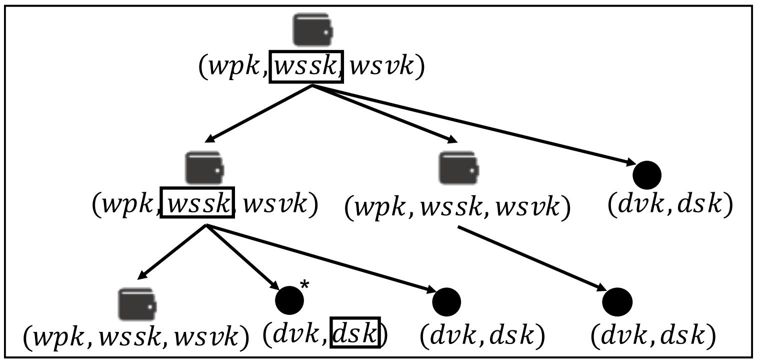

Fig. 3. Safety of Coins. For a target coin (e.g., the starred one), as long as the derived signing key of the coin and the wallet secret spend keys of its owner and its owner's ancestor entities (i.e., the boxed ones) are safe, the coin is safe, even if all other keys (i.e., the non-boxed ones) in the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

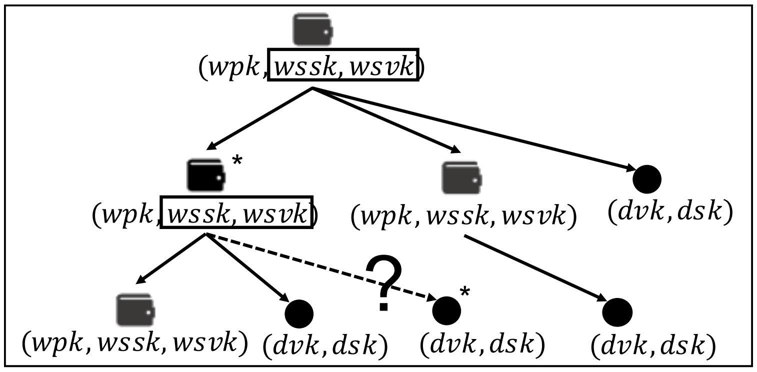

Fig. 4. Privacy of Users. For a target wallet (e.g., the starred one), as long as the wallet secret key and its ancestor entities' wallet secret keys (i.e., the boxed ones) are safe, no attacker can tell whether a target coin (e.g., the starred one) belongs to the target wallet, even if the attacker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

• (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 $\mathsf{RootWalletKeyGen}()$ algorithm, generating the root wallet key pair $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$ of the organization.

• (3) With the root wallet key pair $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$ with $ID = (id_0)$, the root administrator can run the $\mathsf{WalletKeyDelegate}()$ algorithm to generate a wallet key pair $(\mathsf{wpk}_{ID'}, \mathsf{wsk}_{ID'})$ for its direct subordinate with identifier $ID' = (id_0, id_1)$, where $id_1 \in \{0,1\}^*$ identifies a direct subordinate of ID. Furthermore, more generally, with a wallet key pair $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$, the wallet owner (i.e., the entity with identifier $ID = (id_0, \ldots, id_t)$) can run the $\mathsf{WalletKeyDelegate}()$ algorithm to generate wallet key pair $(\mathsf{wpk}_{ID'}, \mathsf{wsk}_{ID'})$ for its any direct subordinate with identifier $ID' = (id_0, id_1, \ldots, id_t, id_{t+1})$, 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 wpk_{ID}, anyone (e.g., the payer of a transaction) can run the $\mathsf{VerifyKeyDerive}()$ algorithm to generate a fresh derived verification key dvk, which will be used as a coin-address for the wallet key owner (i.e., the entity with identifier ID). Note that $\mathsf{VerifyKeyDerive}()$ 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 an entity 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 (together with his identifier and wallet public key) to run the $\mathsf{VerifyKeyCheck}()$ algorithm to check whether the coin's address dvk belongs to him (i.e., was generated from his (ID, wpk_{ID})), and for a derived verification key belonging to him, say dvk, the owner can use his wallet secret spend key (together with his identifier, wallet public key, and wallet secret view key) to run the $\mathsf{SignKeyDerive}()$ algorithm to generate the signing key dsk corresponding to dvk. Moreover, with the dsk, the wallet owner can run the $\mathsf{Sign}()$ 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.[6]

• (6) For any coin on the blockchain, suppose the coin-address is dvk, anyone can run the $\mathsf{Verify}()$ algorithm on inputs dvk and a (transaction, signature) pair (tx, σ), checking the validity of the signature (i.e. whether the transaction is authenticated to spend the coin). In other words, from the view of the public, they can check the transaction's validity using only the information on the blockchain, 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., <u>coin-addresses</u>) 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, the compromising of a derived signing key will not affect the security of any other derived signing 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. In other words, from the coin safety point of view, for a target derived verification key dvk that belongs to entity $ID = (id_0, \ldots, id_t)$ with $t \geq 0$, as long as the corresponding derived signing key dsk is safe and the wallet secret spend keys of the entity (say wssk_{ID}) and its ancestor entities [7] (say $\mathsf{wssk}_{ID_{|i}}$ for $i = 0, \ldots, t-1$) are safe, the coin on dvk is safe, i.e., no attacker can spend the coin on dvk, even if the attacker obtains all other wallet secret keys and derived signing keys, as well as wallet secret view keys $\mathsf{wsvk}_{ID_{|i}} (i = 0, \ldots, t)$. And from the point of view of the privacy of users, as shown in Fig. 4, for a target wallet owner with identifier ID, as long as its wallet secret key and its ancestors' wallet secret keys are safe, given a target derived verification key dvk, no one (except the creator of dvk) can tell whether dvk belongs to ID. *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. 5), we can see that the efficiency of our HDWSA scheme is lower than that of ECDSA, but is still practical for typical

[6] This separation is borrowed from the stealth address mechanisms in [23, 17].

[7] For an entity with identifier $ID = (id_0, \ldots, id_t)$ with $t \geq 1$, we say the entities with identifiers $ID_{|i} := (id_0, \ldots, id_i)$ $(i = 0, \ldots, t-1)$ are its ancestor entities.

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, as shown in Appendix A.

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 [8], besides providing the virtues of SA, our HDWSA can be regarded as a more *secure* HDW than the HDW in [8] and can support more promising applications. More specifically, the HDW in [8] 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 [8] 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 [8] 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 [8] 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 [8] 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 [8] 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 [17, 16], 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 Hierarchical Deterministic Wallet supporting Stealth Address, the construction in this work seems to follow the approach of [17]. 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 [9, 1] to provably secure *hierarchical* deterministic wallet [8] could serve as an evidence of such challenges.

In addition, as our construction employs the concept of hierarchical identifier to support the hierarchy property and is indeed a signature scheme that makes use of the techniques of identity-based cryptography to resist the privilege escalation attack, it is natural to consider whether such a construction could be obtained directly from a Hierarchical Identity-Based Signature (HIBS) scheme [12, 7, 22]. We would like to point out that, similar to that pointed out by Liu et al. [17], neither the definition nor the existing constructions of HIBS consider protecting the signer's privacy, whereas achieving privacy-preservation is one of the most challenging issues in the construction and proof of HDWSA. Consequently, it is not trivial to convert a HIBS scheme to a HDWSA scheme.

1.3 Outline

We formalize the syntax and security models for HDWSA in Sect. 2. Then we propose a HDWSA construction in Sect. 3 and present the proof sketches in Sect. 4. Finally we describe an implementation of our HDWSA construction in Sect. 5 and conclude the paper in Sect. 6. In Appendix. A we show that HDWSA supports the promising use cases, and the security proof is given in Appendix. B.

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.[8]

• Each entity in the system has a unique identifier ID in the form of $ID = (id_0, \ldots, id_t)$ with $t \geq 0$ and $id_i \in \{0,1\}^*(i = 0, \ldots, t)$.

• For any identifier $ID = (id_0, \ldots, id_t)$ with $t \geq 0$, we define $ID_{|i} := (id_0, \ldots, id_i)$ for $i = 0, \ldots, t$, 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) $ID_{|i}(i = 0, \ldots, t-1)$ are the identifiers of ID' 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 $ID = (id_0, \ldots, id_t)$ 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 Hierarchical Deterministic Wallet supporting Stealth Address (HDWSA) scheme consists of eight polynomial-time algorithms (Setup, $\mathsf{RootWalletKeyGen}$, $\mathsf{WalletKeyDelegate}$, $\mathsf{VerifyKeyDerive}$, $\mathsf{VerifyKeyCheck}$, $\mathsf{SignKeyDerive}$, Sign, Verify) as below:

• $\mathsf{Setup}(\lambda) \to \mathsf{PP}$. On input a security parameter λ, the algorithm runs in polynomial time in λ, 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.*

• $\mathsf{RootWalletKeyGen}(ID) \to (\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$. This is a randomized algorithm. On input a Level-0 identifier ID, the algorithm outputs a root (wallet public key, wallet secret key) pair $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$ for ID, where

[8] Actually, an individual user can also be regarded as a special organization, for example, a user may manage his wallets in a hierarchy manner.

$\mathsf{wsk}_{ID} := (\mathsf{wssk}_{ID}, \mathsf{wsvk}_{ID})$ consists of two parts, say *wallet secret spend key* wssk_{ID} and *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.

• $\mathsf{WalletKeyDelegate}(ID, \mathsf{wpk}_{ID_{|(t-1)}}, \mathsf{wsk}_{ID_{|(t-1)}}) \to (\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$. This is a *deterministic* algorithm. On input an entity's identifier $ID = (id_0, \ldots, id_t)$ with $\underline{t \geq 1}$ and its parent entity's (wallet public key, wallet secret key) pair, say $(\mathsf{wpk}_{ID_{|(t-1)}}, \mathsf{wsk}_{ID_{|(t-1)}})$, the algorithm outputs a (wallet public key, wallet secret key) pair $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$ for ID, with $\mathsf{wsk}_{ID} := (\mathsf{wssk}_{ID}, \mathsf{wsvk}_{ID})$ consisting of wallet secret spend key wssk_{ID} and wallet secret view key wsvk_{ID}.

Each entity in the system, including the root entities of the organizations and any entity in any organization, can run this algorithm to generate wallet key pairs for its direct subordinates.

• $\mathsf{VerifyKeyDerive}(ID, \mathsf{wpk}_{ID}) \to \mathsf{dvk}$. This is a randomized algorithm. On input an entity's identifier $ID = (id_0, \ldots, id_t)$ with $\underline{t \geq 0}$ and its wallet public key wpk_{ID}, the algorithm outputs a derived verification key dvk belonging to the entity.

Anyone can run this algorithm to generate a fresh public/verification key for an entity at Level ≥ 0.

• $\mathsf{VerifyKeyCheck}(\mathsf{dvk}, ID, \mathsf{wpk}_{ID}, \mathsf{wsvk}_{ID}) \to 1/0$. This is a deterministic algorithm. On input a derived verification key dvk, an entity's identifier $ID = (id_0, id_1, \ldots, id_t)$ with $\underline{t \geq 0}$, and the entity's wallet public key wpk_{ID} and wallet secret view key wsvk_{ID}, the algorithm outputs a bit $b \in \{0,1\}$, with $b = 1$ meaning that 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 (i.e., was derived from his identifier and wallet public key). Note that only the wallet secret view key is needed here, rather than the whole wallet secret key.

• $\mathsf{SignKeyDerive}(\mathsf{dvk}, ID, \mathsf{wpk}_{ID}, \mathsf{wsk}_{ID}) \to \mathsf{dsk}$ or $\perp$. On input a derived verification key dvk, an entity's identifier $ID = (id_0, \ldots, id_t)$ with $\underline{t \geq 0}$, and the entity's (wallet public key, wallet secret key) pair $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$, the algorithm outputs a derived signing key dsk, or $\perp$ implying that dvk is not a valid verification key derived from (ID, wpk_{ID}).

• $\mathsf{Sign}(m, \mathsf{dvk}, \mathsf{dsk}) \to \sigma$. On input a message m in message space $\mathcal{M}$ and a derived (verification key, signing key) pair $(\mathsf{dvk}, \mathsf{dsk})$, the algorithm outputs a signature σ.

• $\mathsf{Verify}(m, \sigma, \mathsf{dvk}) \to 1/0$. This is a deterministic algorithm. On input a (message, signature) pair (m, σ) and a derived verification key dvk, the algorithm outputs a bit $b \in \{0,1\}$, with $b = 1$ meaning the validness of signature and $b = 0$ otherwise.

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

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

it holds that

$$\begin{aligned}
&\mathsf{VerifyKeyCheck}(\mathsf{dvk}, ID_{|j}, \mathsf{wpk}_{ID_{|j}}, \mathsf{wsvk}_{ID_{|j}}) = 1 \text{ and}\\
&\mathsf{Verify}(m, \mathsf{Sign}(m, \mathsf{dvk}, \mathsf{dsk}), \mathsf{dvk}) = 1.
\end{aligned}$$

Remark: The deterministic algorithm $\mathsf{WalletKeyDelegate}()$ enables the deterministic generation property and hierarchy property, and the randomized $\mathsf{VerifyKeyDerive}()$ algorithm 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 $\mathsf{VerifyKeyDerive}()$, $\mathsf{VerifyKeyCheck}()$, and $\mathsf{SignKeyDerive}()$ together enable the features of SA.

In addition, as the hierarchical identifiers are the foundation on which the hierarchy features are built on, 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. More specifically, for each hierarchical organization, starting from the root administrator, when an entity generates wallet key pair for a direct subordinate, it also issues a corresponding certificate to bind the subordinate's identifier and wallet pubic key. Thus, from the view of a user who wants to send coins to an entity, he can check the certificate chain and make sure he is using the right wallet public key. Note that each payer only needs to verify the target payee's certificate one time, rather than for each transfer. 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, i.e., no one could link the derived verification keys or the corresponding signatures to the corresponding wallet public key and identifier. *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 $\mathsf{Game}_{\mathsf{EUF}}$, which captures that, as shown in Fig. 3, for a target derived verification key dvk belonging to a target entity/wallet in some organization, as long as the corresponding derived signing key 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 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** $\mathsf{Game}_{\mathsf{EUF}}$ is negligible.*

■ **Setup.** $\mathsf{PP} \leftarrow \mathsf{Setup}(\lambda)$ is run and 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 $(ID, \mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$.
An empty set $L_{dvk} = \emptyset$ is initialized, each element of which will be a (derived verification key, identifier) pair (dvk, ID).
Note that the two sets are defined just for the simplicity of description, and $\mathcal{A}$ knows the (ID, wpk_{ID}) pairs in L_{wk} and (dvk, ID) pairs in L_{dvk}.

$\mathcal{A}$ submits a Level-0 identifier, say ID_0^*, to trigger the setup of the target organization.
$(\mathsf{wpk}_{ID_0^*}, \mathsf{wsk}_{ID_0^*}) \leftarrow \mathsf{RootWalletKeyGen}(ID_0^*)$ is run, $\mathsf{wpk}_{ID_0^*}$ is given to $\mathcal{A}$, and $(ID_0^*, \mathsf{wpk}_{ID_0^*}, \mathsf{wsk}_{ID_0^*})$ is added into L_{wk}.
This captures that the adversary may somehow manipulate the target organization's identifier. Note that $\mathsf{RootWalletKeyGen}()$ *does not take any secret information as input and the adversary can also run this algorithm on input any identifier of its choice, this captures that even if the adversary also runs* $\mathsf{RootWalletKeyGen}()$ *on ID_0^* again, it could not break the unforgeability defined by this game.*

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

• Wallet Key Delegate Oracle $\mathsf{OWKeyDelegate}(\cdot)$:
On input an entity's identifier $ID = (id_0, \ldots, id_t)$ with $\underline{t \geq 1}$ such that $ID_{|(t-1)} \in L_{wk}$,[9] this oracle runs $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID}) \leftarrow \mathsf{WalletKeyDelegate}(\ ID, \mathsf{wpk}_{ID_{|(t-1)}}, \mathsf{wsk}_{ID_{|(t-1)}})$, returns wpk_{ID} to $\mathcal{A}$, and sets $L_{wk} = L_{wk} \cup (ID, \mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$,[10] where $(\mathsf{wpk}_{ID_{|(t-1)}}, \mathsf{wsk}_{ID_{|(t-1)}})$ is the (wallet public key, wallet secret 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 $\mathsf{OWKeyDelegate}(\cdot)$). Note that the requirement $ID_{|(t-1)} \in L_{wk}$ is natural although it makes $\mathcal{A}$ have to query $\mathsf{OWKeyDelegate}(\cdot)$ level-by-level starting from ID_0^*'s direct subordinates, 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 $\mathsf{OWskCorrupt}(\cdot)$:
On input an entity's identifier $ID = (id_0, \ldots, id_t)$ with $\underline{t \geq 0}$ [11] such that $ID \in L_{wk}$, this oracle returns the wallet secret key wsk_{ID} of ID to $\mathcal{A}$.
This captures that $\mathcal{A}$ can obtain the wallet secret keys for the existing wallet public keys of its choice.

• Wallet Secret View Key Corruption Oracle $\mathsf{OWsvkCorrupt}(\cdot)$:
On input an entity's identifier $ID = (id_0, \ldots, id_t)$ with $\underline{t \geq 0}$ such that $ID \in L_{wk}$, this oracle returns the wallet secret view key wsvk_{ID} of ID to $\mathcal{A}$.
This captures that $\mathcal{A}$ can obtain the wallet secret view keys for the existing wallet public keys of its choice.

• Verification Key Adding Oracle $\mathsf{ODVKAdd}(\cdot, \cdot)$:
On input a derived verification key dvk and an entity's identifier $ID = (id_0, \ldots, id_t)$ with $\underline{t \geq 0}$ such that $ID \in L_{wk}$, this oracle returns $b \leftarrow \mathsf{VerifyKeyCheck}(\mathsf{dvk}, ID, \mathsf{wpk}_{ID}, \mathsf{wsvk}_{ID})$ to $\mathcal{A}$, where wpk_{ID} and wsvk_{ID} are ID's wallet public key and wallet secret view key respectively. And if $b = 1$, this oracle sets $L_{dvk} = L_{dvk} \cup (\mathsf{dvk}, ID)$.
This captures that $\mathcal{A}$ can probe whether the derived verification keys generated by it are accepted by the owners of the target wallets.

• Signing Key Corruption Oracle $\mathsf{ODSKCorrupt}(\cdot)$:
On input a derived verification key dvk such that there is a corresponding pair (dvk, ID) in L_{dvk}, this oracle returns
$\mathsf{dsk} \leftarrow \mathsf{SignKeyDerive}(\mathsf{dvk}, ID, \mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$ to $\mathcal{A}$, where $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$ 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 $\mathsf{OSign}(\cdot, \cdot)$:
On input a message $m \in \mathcal{M}$ and a derived verification key $\mathsf{dvk} \in L_{dvk}$[12], this oracle returns $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{dvk}, \mathsf{dsk})$ to $\mathcal{A}$, where dsk is a signing key corresponding to 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 σ^*, and a derived verification key dvk^* such that $\mathsf{dvk}^* \in L_{dvk}$.

Let $(\mathsf{dvk}^*, ID^*) \in L_{dvk}$ be the pair corresponding to dvk^*, and suppose the <u>target wallet</u> identifier ID^* be a Level-t^* identifier, say $ID^* = (id_0^*, \ldots, id_{t^*}^*)$. $\mathcal{A}$ succeeds in the game if
$\mathsf{Verify}(m^*, \sigma^*, \mathsf{dvk}^*) = 1$ under the restrictions that (1) $\mathcal{A}$ did not query $\mathsf{OWskCorrupt}(\cdot)$ on $ID_{|i}^*$ for any i

[9] Note that we are abusing the concept of '$\in$'. In particular, if there exists a tuple $(ID, \mathsf{wpk}_{ID}, \mathsf{wsk}_{ID}) \in L_{wk}$ for some $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$ pair, we say that $ID \in L_{wk}$.

[10] Note that $\mathsf{WalletKeyDelegate}(\cdot, \cdot, \cdot)$ is a deterministic algorithm, so that querying $\mathsf{OWKeyDelegate}(\cdot)$ on the same identifier will obtain the same response.

[11] Note that actually $\mathcal{A}$ should not make such a query with $t = 0$ (as required by the success conditions defined in later **Output Phase**), since that means $\mathcal{A}$ is querying the root wallet secret key of the target organization and will win the game trivially.

[12] Note that we are abusing the concepts of '$\in$'. In particular, if there exists a pair $(\mathsf{dvk}, ID) \in L_{dvk}$ for some ID, we say that $\mathsf{dvk} \in L_{dvk}$.

such that $0 \le i \le t^*$, (2) $\mathcal{A}$ did not query $\mathsf{ODSKCorrupt}()$ on dvk^*, and (3) $\mathcal{A}$ did not query $\mathsf{OSign}()$ on (m^*, dvk^*).

Remark: It is worth mentioning that $\mathcal{A}$ is allowed to query the wallet secret view keys for $ID^*_{|i}$ $(i = 0, \ldots, t^*)$ by $\mathsf{OWsvkCorrupt}(\cdot)$. This is important since it 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 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** $\mathsf{Game}_{\mathsf{swEUF}}$ is negligible.*

The game $\mathsf{Game}_{\mathsf{swEUF}}$ is identical to the above game $\mathsf{Game}_{\mathsf{EUF}}$, 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 the its target wallet, say, $ID^* = (id^*_0, id^*_1, \ldots, id^*_{t^*})$ with $t^* \ge 0$ and $ID^*_{|0} = ID^*_0$ (i.e., the identifier of an entity in the target organization ID^*_0), committing that the target derived verification key dvk^* in the **Ouput Phase** will be one belonging to ID^*.

The **wallet unlinkability** is defined by the following game $\mathsf{Game}_{\mathsf{WUNL}}$, 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.

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

■ **Setup.** $\mathsf{PP} \leftarrow \mathsf{Setup}(\lambda)$ is run and PP is given to $\mathcal{A}$.
As in the **Setup** phase of $\mathsf{Game}_{\mathsf{EUF}}$, two empty sets $L_{wk} = \emptyset$ and $L_{dvk} = \emptyset$ are initialized.
$\mathcal{A}$ submits two different Level-0 identifiers, say $ID^{*(0)}_0$ and $ID^{*(1)}_0$.
For $k = 0, 1$: $(\mathsf{wpk}_{ID^{*(k)}_0}, \mathsf{wsk}_{ID^{*(k)}_0}) \leftarrow \mathsf{RootWalletKeyGen}(ID^{*(k)}_0)$ is run, $\mathsf{wpk}_{ID^{*(k)}_0}$ is given to $\mathcal{A}$, and $(ID^{*(k)}_0, \mathsf{wpk}_{ID^{*(k)}_0}, \mathsf{wsk}_{ID^{*(k)}_0})$ is added into L_{wk}.
This captures that the adversary may somehow manipulate the target organizations' identifiers.

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

■ **Challenge.** $\mathcal{A}$ submits two different challenge wallets' identifiers $ID^{(0)} = (id^{(0)}_0, \ldots, id^{(0)}_{t^{(0)}})$ and $ID^{(1)} = (id^{(1)}_0, \ldots, id^{(1)}_{t^{(1)}})$, such that $t^{(0)} \ge 0, t^{(1)} \ge 0$, and $ID^{(0)}, ID^{(1)} \in L_{wk}$ (implying $ID^{(0)}_{|0}, ID^{(1)}_{|0} \in \{ID^{*(0)}_0, ID^{*(1)}_0\}$).
A random bit $c \in \{0, 1\}$ is chosen, $\mathsf{dvk}^* \leftarrow \mathsf{VerifyKeyDerive}(ID^{(c)}, \mathsf{wpk}_{ID^{(c)}})$ is given to $\mathcal{A}$.
$(\mathsf{dvk}^*, ID^{(c)})$ 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 $\mathsf{OWskCorrupt}(\cdot)$ or $\mathsf{OWsvkCorrupt}(\cdot)$ oracle on any $ID \in \{ID^{(0)}_{|i} \mid 0 \le i \le t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 0 \le i \le t^{(1)}\}$, and (2) $\mathcal{A}$ did not query oracle $\mathsf{ODVKAdd}(\cdot, \cdot)$ on $(\mathsf{dvk}^*, ID^{(0)})$ or $(\mathsf{dvk}^*, ID^{(1)})$.
The advantage of $\mathcal{A}$ is $Adv^{\mathsf{WUNL}}_{\mathcal{A}} = |\Pr[c' = c] - \frac{1}{2}|$.
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 $\mathsf{ODskCorrupt}()$ and $\mathsf{OSign}()$

oracles on the challenge derived verification key dvk^*, and this captures that neither the signature or the derived signing key leaks the privacy of the owner of dvk^*.

It is worth noticing that, while the adversary in $\mathsf{Game}_{\mathsf{EUF}}$ should not query $\mathsf{OWskCorrupt}(\cdot)$ on an identifier $ID = (id_0, \ldots, id_t)$ 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 $\mathsf{Game}_{\mathsf{WUNL}}$ may query $\mathsf{OWskCorrupt}(\cdot)$ and/or $\mathsf{OWsvkCorrupt}(\cdot)$ on an identifier $ID = (id_0, \ldots, id_t)$ with $t = 0$ such that $ID \in L_{wk}$, depending on its challenge wallet identifier pair $(ID^{(0)}, ID^{(1)})$. In particular, if $ID^{(0)}_{|0} = ID^{(1)}_{|0} = ID_0^{*(k)}$, then the adversary is allowed to query $\mathsf{OWskCorrupt}(\cdot)$ and/or $\mathsf{OWsvkCorrupt}(\cdot)$ 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 $\mathsf{Game}_{\mathsf{swWUNL}}$, denoted by $Adv_{\mathcal{A}}^{\mathsf{swWUNL}}$, is negligible.*

The game $\mathsf{Game}_{\mathsf{swWUNL}}$ is identical to the above game $\mathsf{Game}_{\mathsf{WUNL}}$, 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, $ID^{(0)} = (id_0^{(0)}, \ldots, id_{t^{(0)}}^{(0)})$ and $ID^{(1)} = (id_0^{(1)}, \ldots, id_{t^{(1)}}^{(1)})$ such that $t^{(0)} \geq 0, t^{(1)} \geq 0$, and $ID^{(0)}_{|0}, ID^{(1)}_{|1} \in \{ID_0^{*(0)}, ID_0^{*(1)}\}$.

3 Our Construction

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

3.1 Preliminaries

Bilinear Map Groups [4] Let λ be a security parameter and p be a λ-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 map groups if there exists a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ 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 [26]). *The CDH problem in bilinear map 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 ϵ 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, ϵ)-CDH assumption holds in $(p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e})$ if no t-time algorithm has advantage at least ϵ in solving the CDH problem in $(p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e})$.

3.2 Construction

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

$$\mathsf{PP} = ((p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e}), H_0, H_1, H_2, H_3, H_4),$$

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.

• $\mathsf{RootWalletKeyGen}(ID) \rightarrow (\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$. 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 $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$ as :

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

• $\mathsf{WalletKeyDelegate}(ID, \mathsf{wpk}_{ID_{|(t-1)}}, \mathsf{wsk}_{ID_{|(t-1)}}) \rightarrow (\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$. On input an entity's identifier $ID = (id_0, \ldots, id_t)$ with $\underline{t \geq 1}$ and its parent entity's (wallet public key, wallet secret key) pair, say $\mathsf{wpk}_{ID_{|(t-1)}} \in \mathbb{G}_1 \times \mathbb{G}_1$ and $\mathsf{wsk}_{ID_{|(t-1)}} = (\alpha_{ID_{|(t-1)}}, \beta_{ID_{|(t-1)}}) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, the algorithm proceeds as below:

1. Compute $Q_{ID} = H_0(ID) \in \mathbb{G}_1^*$,
2. Compute $\alpha_{ID} = H_1(Q_{ID}, \alpha_{ID_{|(t-1)}} Q_{ID}) \in \mathbb{Z}_p^*$,
3. Compute $\beta_{ID} = H_2(Q_{ID}, \beta_{ID_{|(t-1)}} Q_{ID}) \in \mathbb{Z}_p^*$,
4. Output wallet key pair $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID})$ for ID as

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

• $\mathsf{VerifyKeyDerive}(ID, \mathsf{wpk}_{ID}) \rightarrow \mathsf{dvk}$. On input an entity's identifier $ID = (id_0, \ldots, id_t)$ with $\underline{t \geq 0}$ and the entity's wallet public key $\mathsf{wpk}_{ID} = (A_{ID}, B_{ID}) \in \mathbb{G}_1 \times \mathbb{G}_1$, the algorithm proceeds as below:

1. Choose a uniformly random $r \in \mathbb{Z}_p^*$,
2. Output a derived verification key $\mathsf{dvk} := (Q_r, Q_{vk})$ with

$$\begin{aligned}Q_r &= rP \in \mathbb{G}_1,\\ Q_{vk} &= \hat{e}(H_3(B_{ID}, rP, rB_{ID}), -A_{ID}) \in \mathbb{G}_2.\end{aligned}$$

• $\mathsf{VerifyKeyCheck}(\mathsf{dvk}, ID, \mathsf{wpk}_{ID}, \mathsf{wsvk}_{ID}) \rightarrow 1/0$. On input a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$, an entity's identifier $ID = (id_0, \ldots, id_t)$ with $\underline{t \geq 0}$, and the entity's wallet public key $\mathsf{wpk}_{ID} = (A_{ID}, B_{ID}) \in \mathbb{G}_1 \times \mathbb{G}_1$ and wallet secret view key $\mathsf{wsvk}_{ID} = \beta_{ID} \in \mathbb{Z}_p^*$, the algorithm checks whether $Q_{vk} \overset{?}{=} \hat{e}(H_3(B_{ID}, Q_r, \beta_{ID} Q_r), -A_{ID})$ holds. If it holds, the algorithm outputs 1, otherwise outputs 0.

• $\mathsf{SignKeyDerive}(\mathsf{dvk}, ID, \mathsf{wpk}_{ID}, \mathsf{wsk}_{ID}) \rightarrow \mathsf{dsk}$ or $\perp$. On input a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$, an entity's identifier $ID = (id_0, \ldots,\ id_t)$ with $\underline{t \geq 0}$, and the entity's (wallet public key, wallet secret key) pair, say $\mathsf{wpk}_{ID} = (A_{ID}, B_{ID}) \in \mathbb{G}_1 \times \mathbb{G}_1$ and $\mathsf{wsk}_{ID} = (\alpha_{ID}, \beta_{ID}) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, the algorithm checks whether
$Q_{vk} \overset{?}{=} \hat{e}(H_3(B_{ID}, Q_r,\ \beta_{ID} Q_r), -A_{ID})$ holds. If it holds, the algorithm outputs a derived signing key dsk as

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

otherwise, outputs $\perp$.

• $\mathsf{Sign}(m, \mathsf{dvk}, \mathsf{dsk}) \rightarrow \sigma$. On input a message m in message space $\mathcal{M}$, a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$, and a derived signing key $\mathsf{dsk} \in \mathbb{G}_1$, 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 $h = H_4(\mathsf{dvk}, m, X) \in \mathbb{Z}_p^*$.
3. Compute $Q_\sigma = h \cdot \mathsf{dsk} + xP \in \mathbb{G}_1$.
4. Output $\sigma = (h, Q_\sigma) \in \mathbb{Z}_p^* \times \mathbb{G}_1$ as the signature for m.

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

Correctness. The correctness is satisfied, since

$$\begin{aligned}
&\beta_{ID} Q_r = \beta_{ID}(rP) = r(\beta_{ID} P) = rB_{ID},\\
&\hat{e}(Q_\sigma, P) \cdot (Q_{vk})^h = \hat{e}(h \cdot \mathsf{dsk} + xP, P) \cdot \hat{e}(H_3(B_{ID}, rP, rB_{ID}), -A_{ID})^h\\
&= \hat{e}(h \cdot \alpha_{ID} H_3(B_{ID}, Q_r, \beta_{ID} Q_r), P) \cdot \hat{e}(xP, P)\\
&\quad \cdot \hat{e}(H_3(B_{ID}, rP, rB_{ID}), -\alpha_{ID} P)^h\\
&= \hat{e}(xP, P) = X.
\end{aligned}$$

4 Security Proofs

In this section, we prove our HDWSA construction is selective wallet existentially unforgeable (with respect to Definition 2) and is selective wallet unlinkable (with respect to Definition 4). In addition, we also show that at the cost of a reduction loss factor, our construction can be proven unforgeable and unlinkable in the adaptive model.

4.1 Proof of Selective Unforgeability

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

Proof (**Proof Sketch).** We give a sketch of the proof below and give the proof details in Appendix B.1.

In the proof we will show that, if there exists a PPT adversary $\mathcal{A}$ that can win $\mathsf{Game}_{\mathsf{swEUF}}$ for our HDWSA construction with non-negligible probability, then we can construct a PPT algorithm $\mathcal{B}$ that can solve the CDH problem with non-negligible probability.

$\mathcal{B}$ is given bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e})$ and a tuple $(P, A = aP, B = bP) \in \mathbb{G}_1^3$ for unknown $a, b \in \mathbb{Z}_p^*$, and the target of $\mathcal{B}$ is to compute an element $C \in \mathbb{G}_1$ such that $C = abP$.

Let $\mathcal{A}$'s target wallet identifier be $ID^* = (id_0^*, \ldots, id_{t^*}^*)$. To simulate the game $\mathsf{Game}_{\mathsf{swEUF}}$ to $\mathcal{A}$, $\mathcal{B}$ programs random oracles for hash functions H_0, H_1, H_3 and H_4 and sets wallet keys for $ID^*_{|i}$ $(i = 0, \ldots, t^*)$ as below:

• For $i = 0, \ldots, t^*$: $\mathcal{B}$ sets $(A_{ID^*_{|i}} = \alpha'_{ID^*_{|i}} A,\ B_{ID^*_{|i}} = \beta_{ID^*_{|i}} P)$, where $\alpha'_{ID^*_{|i}}$ and $\beta_{ID^*_{|i}}$ are chosen uniformly at random from $\mathbb{Z}_p^*$. Note that this means that $\mathcal{B}$ knows $\mathsf{wsvk}_{ID^*_{|i}} = \beta_{ID^*_{|i}}$ but does not know the value of $\mathsf{wssk}_{ID^*_{|i}}$ (i.e., $\alpha'_{ID^*_{|i}} a$).

• $\mathcal{B}$ programs H_0 so that $H_0(ID) = Q_{ID} = b_{ID} B$ for $ID \in \{ID^*_{|i} \mid 1 \le i \le t^*\}$ and $H_0(ID) = Q_{ID} = b_{ID} P$ for $ID \notin \{ID^*_{|i} \mid 1 \le i \le t^*\}$, where b_{ID} is chosen uniformly at random from $\mathbb{Z}_p^*$.

• Note that with the above two settings, H_1 is implicitly programmed so that $H_1(Q_{ID^*_{|i}}, (\alpha'_{ID^*_{|(i-1)}} a) Q_{ID^*_{|i}}) = \alpha'_{ID^*_{|i}} a$ (for $i = 1 \ldots, t^*$). Note that $\mathcal{B}$ does not need to worry that it cannot output hash value $\alpha'_{ID^*_{|i}} a$ for such a query, since if $\mathcal{A}$ made such a query, the second input, say $(\alpha'_{ID^*_{|(i-1)}} a) Q_{ID^*_{|i}}$ actually provides a solution for CDH problem (note that $(\alpha'_{ID^*_{|(i-1)}} a) Q_{ID^*_{|i}} = (\alpha'_{ID^*_{|(i-1)}} b_{ID^*_{|i}}) abP$). For any other queries to H_1, $\mathcal{B}$ chooses uniformly random value from $\mathbb{Z}_p^*$ as the output hash value.

• With the above settings, and by calling H_2 as a normal hash function, $\mathcal{B}$ can generate wallet key pairs $\mathsf{wpk}_{ID} = (A_{ID} = \alpha_{ID} P, B_{ID} = \beta_{ID} P)$ for any $ID \notin \{ID^*_{|i} \mid 0 \le i \le t^*\}$, where α_{ID} and β_{ID} are outputs of H_1 and H_2 respectively. This means that, for any $ID \notin \{ID^*_{|i} \mid 0 \le i \le t^*\}$, $\mathcal{B}$ can simulate the $\mathsf{OWKDelegate}()$, $\mathsf{OWskCorrupt}()$, and $\mathsf{OWsvkCorrupt}()$ to $\mathcal{A}$. On the other side, for any $ID \in \{ID^*_{|i} \mid 0 \le$

$i \leq t^*\}$, $\mathcal{B}$ can simulate $\mathsf{OWKDelegate}()$ by returning the corresponding $\mathsf{wpk}_{ID^*_{|i}}$, and can simulate the $\mathsf{OWsvkCorrupt}()$ by returning corresponding $\beta_{ID^*_{|i}}$. Note that $\mathcal{A}$ does not query $\mathsf{OWskCorrupt}()$ on any $ID \in \{ID^*_{|i} \mid 0 \leq i \leq t^*\}$.

• Note that $\mathcal{A}$'s target derived verification key dvk^* must be one valid with respect to ID^* (i.e., $\mathsf{ODVKAdd}(\mathsf{dvk}^*, ID^*)$ returns 1), which must be based on a H_3-query on input $(B_{ID^*}, input_2, input_3)$ for some $input_2, input_3 \in \mathbb{G}_1$ such that $\hat{e}(B_{ID^*}, input_2) = \hat{e}(P, input_3)$. $\mathcal{B}$ chooses a uniformly random $j^* \in \{1, \ldots, q_{H_3} + q_{vka}\}$ as its guess that (1) the j^*-th query to H_3, say on inputs $(input^*_1, input^*_2, input^*_3)$, will satisfy $input^*_1 = B_{ID^*}$ AND $\hat{e}(input^*_1, input^*_2) = \hat{e}(P, input^*_3)$, and (2) the target derived verification key dvk^* will be based on the j^*-th query to H_3.[13] Then $\mathcal{B}$ programs H_3 such that, for j-th H_3-query where $j \neq j^*$, $H_3(input_1, input_2, input_3) = V_j = \eta_j P$, and for j^*-th H_3-query, $H_3(input^*_1, input^*_2, input^*_3) = V_j = \eta_j B$, where $\eta_j \in \mathbb{Z}^*_p$ is chosen uniformly at random. Note that the probability that $\mathcal{A}$ guess the right j^* is at least $1/(q_{H_3} + q_{vka})$.

• With the above settings, $\mathcal{B}$ can simulate the $\mathsf{ODVKAdd}()$ to $\mathcal{A}$, since it knows the wallet secret view key β_{ID} for any ID. Further, $\mathcal{B}$ can simulate the $\mathsf{ODSKCorrupt}()$ to $\mathcal{A}$. In particular, consider a queried dvk belonging to ID: if $ID \notin \{ID^*_{|i} \mid 0 \leq i \leq t^*\}$, $\mathcal{B}$ can compute $\mathsf{dsk} = \alpha_{ID}(\eta_j P)$; if $ID \in \{ID^*_{|i} \mid 0 \leq i \leq t^*\}$ and $\mathsf{dvk} \neq \mathsf{dvk}^*$, $\mathcal{B}$ can compute $\mathsf{dsk} = \alpha'_{ID^*_{|i}} \eta_j A$ (note that $\alpha_{ID^*_{|i}} = \alpha'_{ID^*_{|i}} a$). Note that $\mathcal{A}$ does not query $\mathsf{ODSKCorrupt}()$ on dvk^*.

• With the above settings, $\mathcal{B}$ can simulate the $\mathsf{OSign}()$ to $\mathcal{A}$. In particular, for any $\mathsf{dvk} \neq \mathsf{dvk}^*$, $\mathcal{B}$ just runs the signing algorithm, since it knows the corresponding dsk as shown above. For dvk^*, $\mathcal{B}$ does not know the value of $\mathsf{dsk}^* = (\alpha'_{ID^*} a) \cdot (\eta_{j^*} B)$, but it can set $X = \hat{e}(x_1 A, B) \cdot \hat{e}(x_2 P, P)$, implicitly setting $x = x_1 ab + x_2$, and program H_4 to make $h = H_4(\mathsf{dvk}^*, m, X) = -\frac{x_1}{\alpha'_{ID^*} \eta_{j^*}}$, so that $Q_\sigma = x_2 P = h \cdot \mathsf{dsk}^* + xP$.

• Finally, by programming H_4 and applying rewinding and forking lemma [2], $\mathcal{B}$ can extract a solution for the CDH problem.

4.2 Proof of Selective Unlinkability

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

*Proof (***Proof Sketch).** We give a sketch of the proof below and give the proof details in Appendix B.2.

In the proof we will show that, if there exists a PPT adversary $\mathcal{A}$ that can win $\mathsf{Game}_{\mathsf{swWUNL}}$ for our HDWSA construction with non-negligible advantage, then we can construct a PPT algorithm $\mathcal{B}$ that can solve the CDH problem with non-negligible probability.

$\mathcal{B}$ is given bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e})$ and a tuple $(P, A = aP, B = bP) \in \mathbb{G}_1^3$ for unknown $a, b \in \mathbb{Z}^*_p$, and the target of $\mathcal{B}$ is to compute an element $C \in \mathbb{G}_1$ such that $C = abP$.

Let $\mathcal{A}$'s two challenge wallet identifiers be $ID^{(k)} = (id^{(k)}_0, \ldots, id^{(k)}_{t^{(k)}})$ for $k = 0, 1$. Let $S_{chID} := \{ID^{(0)}_{|i} \mid 0 \leq i \leq t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 0 \leq i \leq t^{(1)}\}$. To simulate the game $\mathsf{Game}_{\mathsf{swWUNL}}$ to $\mathcal{A}$, $\mathcal{B}$ programs random oracles for hash functions H_0, H_2, and H_3 and sets wallet keys for $ID \in S_{chID}$ as below:

• For $ID \in S_{chID}$: suppose $ID = ID^{(k)}_{|i}$, $\mathcal{B}$ sets $(A_{ID^{(k)}_{|i}} = \alpha_{ID^{(k)}_{|i}} P$, $B_{ID^{(k)}_{|i}} = \beta'_{ID^{(k)}_{|i}} B)$, where $\alpha_{ID^{(k)}_{|i}}$ and $\beta'_{ID^{(k)}_{|i}}$ are chosen uniformly at random from $\mathbb{Z}^*_p$. Note that this means $\mathcal{B}$ knows $\mathsf{wssk}_{ID^{(k)}_{|i}} = \alpha_{ID^{(k)}_{|i}}$ but does not know the value of $\mathsf{wsvk}_{ID^{(k)}_{|i}}$ (i.e., $\beta'_{ID^{(k)}_{|i}} b$).

• $\mathcal{B}$ programs H_0 so that $H_0(ID) = Q_{ID} = a_{ID} A$ for $ID \in \{ID^{(0)}_{|i} \mid 1 \leq i \leq t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 1 \leq i \leq t^{(1)}\}$ and $H_0(ID) = Q_{ID} = a_{ID} P$ for $ID \notin \{ID^{(0)}_{|i} \mid 1 \leq i \leq t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 1 \leq i \leq t^{(1)}\}$, where a_{ID} is chosen uniformly at random from $\mathbb{Z}^*_p$.

[13] It is assumed that the adversary makes q_{H_3} queries to random oracle H_3 and q_{vka} queries to the verification key adding oracle $\mathsf{ODVKAdd}()$.

• Note that with above two settings, H_2 is implicitly programmed so that $H_2(Q_{ID^{(k)}_{|i}}, (\beta'_{ID^{(k)}_{|(i-1)}} b)Q_{ID^{(k)}_{|i}}) = \beta'_{ID^{(k)}_{|i}} b$ for $ID^{(k)}_{|i} \in \{ID^{(0)}_{|i} \mid 1 \le i \le t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 1 \le i \le t^{(1)}\}$. Note that $\mathcal{B}$ does not need to worry that it cannot output hash value $\beta'_{ID^{(k)}_{|i}} b$ for such a query, since if $\mathcal{A}$ made such a query **(we denote this event by $\mathbf{E_2}$)**, the second input, say $(\beta'_{ID^{(k)}_{|(i-1)}} b)Q_{ID^{(k)}_{|i}}$ actually provides a solution for CDH problem (note that $(\beta'_{ID^{(k)}_{|(i-1)}} b)Q_{ID^{(k)}_{|i}} = (\beta'_{ID^{(k)}_{|(i-1)}} a_{ID^{(k)}_{|i}})abP$). For any other queries to H_2, $\mathcal{B}$ chooses uniformly random value from $\mathbb{Z}_p^*$ as the output hash value.

• With the above settings, and by calling H_1 as a normal hash function, $\mathcal{B}$ can generate wallet key pairs $\mathsf{wpk}_{ID} = (A_{ID} = \alpha_{ID}P, B_{ID} = \beta_{ID}P)$ for any $ID \notin S_{chID}$, where α_{ID} and β_{ID} are outputs of H_1 and H_2 respectively. This means that, for any $ID \notin S_{chID}$, $\mathcal{B}$ can simulate the $\mathsf{OWKDelegate}()$, $\mathsf{OWskCorrupt}()$, and $\mathsf{OWsvkCorrupt}()$ to $\mathcal{A}$. On the other side, for any $ID \in S_{chID}$, $\mathcal{B}$ can simulate $\mathsf{OWKDelegate}()$ by returning the corresponding $\mathsf{wpk}_{ID^{(k)}_{|i}}$. Note that $\mathcal{A}$ does not query $\mathsf{OWskCorrupt}()$ or $\mathsf{OWsvkCorrupt}()$ on any $ID \in S_{chID}$.

• For the H_3-queries and $\mathsf{ODVKAdd}()$-queries, $\mathcal{B}$ behaves in different manners before and after the **Challenge Phase**. In particular,

1. In **Probing Phase 1**: $\mathcal{B}$ programs H_3 in a standard manner, namely, for each query on new/different inputs, $\mathcal{B}$ chooses a uniformly random $V \in \mathbb{G}_1$ and returns V. Consequently, for any $\mathsf{ODVKAdd}()$-query, say on $(\mathsf{dvk} = (Q_r, Q_{vk}), ID)$, $\mathcal{B}$ just runs the normal $\mathsf{VerifyKeyCheck}()$, namely, identifies the H_3-tuple $(B_{ID}, Q_r, input_3, V) \in L_{H_3}$ and checks whether $\hat{e}(B_{ID}, Q_r) = \hat{e}(P, input_3)$ and $Q_{vk} \overset{?}{=} \hat{e}(V, -A_{ID})$ hold.
2. In **Challenge Phase**, $\mathcal{B}$ chooses a random $c \in \{0,1\}$, and sets $\mathsf{dvk}^* = (Q_r^*, Q_{vk}^*)$ with $Q_r^* = A, Q_{vk}^* = \hat{e}(V^*, -A_{ID^{(c)}})$ where $V^* \in \mathbb{G}_1^*$ is chosen uniformly at random. Note that $\mathcal{B}$ is implicitly programming H_3 such that $H_3(B_{ID^{(c)}}, A, \beta'_{ID^{(c)}}aB) = V^*$, although it does not know the value of the third input $\beta'_{ID^{(c)}}aB$. Note that $\mathcal{B}$ does not need to worry that $\mathcal{A}$ has made such a query to H_3, since this is the first time the group element A appears in $\mathcal{A}$'s view.
3. In **Probing Phase 2**: When H_3 is queried on a tuple $(B_{ID^{(k)}}, A, input_3)$ for some $k \in \{0,1\}$ and $input_3 \in \mathbb{G}_1$, $\mathcal{B}$ checks whether $\hat{e}(input_3, P) = \hat{e}(B_{ID^{(k)}}, A)$ holds **(this event is denoted by $\mathbf{E}_3$)**. If $\mathbf{E}_3$ happens, $\mathcal{B}$ outputs $\frac{1}{\beta'_{ID^{(k)}}} input_3$ as its solution for the CDH problem and aborts the game, otherwise, $\mathcal{B}$ acts as the same way as in that of **Probing Phase 1**. When $\mathsf{ODVKAdd}()$ is queried on a pair $(\mathsf{dvk} = (Q_r, Q_{vk}), ID)$, if $ID \notin \{ID^{(0)}, ID^{(1)}\}$ OR $Q_r \neq Q_r^*$, $\mathcal{B}$ just runs the normal $\mathsf{VerifyKeyCheck}()$, otherwise (i.e, $ID \in \{ID^{(0)}, ID^{(1)}\}$ AND $Q_r = Q_r^*$ AND $Q_{vk} \neq Q_{vk}^*$), $\mathcal{B}$ directly returns 0 to $\mathcal{A}$ (the probability that $\mathcal{B}$'s simulation is incorrect is bounded by $q_{vka}/(p-1)$, the analysis is referred to the proof details in Appendix B.2).

• With the above settings, $\mathcal{B}$ can simulate $\mathsf{ODSKCorrupt}()$ to $\mathcal{A}$, since it knows α_{ID} for any ID and programs the H_3 oracle. Particularly, when $\mathcal{A}$ makes $\mathsf{ODSKCorrupt}()$ on dvk^*, $\mathcal{B}$ returns $\mathsf{dsk}^* = \alpha_{ID^{(c)}} \cdot V^*$. Furthermore, $\mathcal{B}$ can simulate the $\mathsf{OSign}()$ to $\mathcal{A}$, since it knows the derived signing key for any derived verification key, including dvk^*.

• Finally, if neither $\mathbf{E}_2$ nor $\mathbf{E}_3$ happens, and $\mathcal{B}$ does not simulate $\mathsf{ODVKAdd}()$ wrongly (which happens only with probability $q_{vka}/(p-1)$), the game simulated by $\mathcal{B}$ is identical to the real game. Meanwhile, in the simulated game, V^* is uniformly chosen and unknown to $\mathcal{A}$, so no information of c is leaked through dvk^* and dsk^* and the advantage of $\mathcal{A}$ is 0 over a random guess. Therefore, under the assumption that $\mathcal{A}$ has a non-negligible advantage $\epsilon_{\mathcal{A}}$ in $\mathsf{Game}_{\mathsf{swWUNL}}$, we conclude that $\Pr[\mathbf{E}_2 \vee \mathbf{E}_3] \ge \epsilon_{\mathcal{A}} - \frac{q_{vka}}{p-1}$ in the simulation, and $\mathcal{B}$ can solve the CDH problem with the same probability.

4.3 Unforgeability and Unlinkability in the Adaptive Model

Following Boneh et al.'s approach for achieving full security of Hierarchical Identity Based Encryption (HIBE) construction [3], 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

Table 2. Sizes of Signature and Keys of an Implementation

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. Time of Key Generation/Derivation, Signing and Verification of an Implementation of Our HDWSA Scheme

Setup	RootWalletKeyGen	WalletKeyDelegate	VerifyKeyDerive	VerifyKeyCheck	SignKeyDerive	Sign	Verify
7.831 ms	1.603 ms	4.893 ms	3.920 ms	3.125 ms	3.316 ms	2.188 ms	0.667 ms

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.

The approach to proving adaptive unforgeability is to guess the target wallet before interacting with the adversary as follows. In the Setup phase, the challenger simulates the root wallet key for the root identifier $ID_0^* = (id_0^*)$ as in the current proof for selective unforgeability. Then, before the start of the Probing Phase, the challenger chooses a random integer $t^* \in [0, h]$ as its guess that the target wallet will be at Level-t^*. Then the challenger chooses random $j_1^* \in [1, q_{H_0}], j_2^* \in [1, q_{H_0}], \ldots, j_{t^*}^* \in [1, q_{H_0}]$, as its guess that ID_0^*'s j_1^* -th subordinate, say $ID_1^* = (id_0^*, id_1^*)$, will be the Level-1 ancestor of the target wallet, ID_1^*'s j_2^* -th subordinate, say $ID_2^* = (id_0^*, id_1^*, id_2^*)$, will be the Level-2 ancestor of the target wallet, ..., and $ID_{t^*-1}^*$'s $j_{t^*}^*$ -th subordinate, say $ID_{t^*}^* = (id_0^*, id_1^*, id_2^*, \ldots, id_{t^*}^*)$, will be the target wallet. Note that for the case of $t^* = 0$, the challenger guesses $ID_0^* = (id_0^*)$ as the target wallet and does not need to choose the random $j_1^*, \ldots, j_{t^*}^*$.

To deal with the above random guess, each identifier should be assigned an index in the hash table for H_0. In the Probing Phase, when the adversary queries H_0 on input an identifier ID at Level-k of organization ID_0^*, say $ID = (id_0^*, id_1, \ldots, id_k)$, for each $i \in [1, k]$, if $ID_i := (id_0^*, id_1, \ldots, id_i)$ has not been queried before, the challenger simulates a query to H_0 on input ID_i to add ID_i into the hash table for H_0.

In such a way, the challenger can program H_0 as in the current proof for selective unforgeability, even when the target wallet identifier is selected adaptively, rather than committed in advance as in the selective model. If the adversary queries $\mathsf{OWskCorrupt}()$ on any of the guessed identifiers (i.e., the guessed target wallet or its ancestors), the challenger aborts (i.e., the challenger's guess is wrong), otherwise, the proof is the same as the current proof for selective unforgeability.

Note that the challenger can guess the right target with probability at least $\frac{1}{h+1}(\frac{1}{q_{H_0}})^{t^*} \geq \frac{1}{h+1}\frac{1}{q_{H_0}^h}$.

The proof for unlinkability in the adaptive model is similar.

It is worth mentioning that, removing such a reduction loss factor in the adaptive unforgeability and unlinkability proof could be pretty challenging. As the situation is similar to that of HIBE's development, the techniques for adaptive HIBE, e.g., the dual system encryption [27], could be potential tools to address this problem. We leave this as our future work.

5 Implementation

We implemented our HDWSA scheme in Go [14] and the code is available at https://github.com/cryptoscheme/hdwsa. Our implementation uses the Pairing-Based Cryptography Library [25], 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 $\mathbb{G}_1$.

6 Conclusion

In this work, we formally define the syntax and security models of Hierarchical Deterministic Wallet supporting Stealth Address, which captures all the versatile functionalities that lead to the popularity of HDW and SA as well as all the security guarantees (including the safety and privacy) 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 time and space consumption is suitable for cryptocurrency settings.

References

1. Alkadri, N.A., Das, P., Erwig, A., Faust, S., Krämer, J., Riahi, S., Struck, P.: Deterministic wallets in a quantum world. In: CCS '20. pp. 1017–1031 (2020). https://doi.org/10.1145/3372297.3423361
2. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) CCS 2006. pp. 390–399. ACM (2006). https://doi.org/10.1145/1180405.1180453
3. Boneh, D., Boyen, X., Goh, E.: Hierarchical identity based encryption with constant size ciphertext. In: EUROCRYPT 2005. pp. 440–456 (2005). https://doi.org/10.1007/11426639“˙26, https://doi.org/10.1007/11426639_26
4. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. In: CRYPTO 2001. pp. 213–229 (2001). https://doi.org/10.1007/3-540-44647-8“˙13
5. Buterin, V.: Deterministic wallets, their advantages and their understated flaws. Bitcoin Magazine (2013)
6. ByteCoin: Untraceable transactions which can contain a secure message are inevitable (2011), https://bitcointalk.org/index.php?topic=5965.0
7. Chow, S.S.M., Hui, L.C.K., Yiu, S., Chow, K.P.: Secure hierarchical identity based signature and its application. In: ICICS 2004. pp. 480–494 (2004). https://doi.org/10.1007/978-3-540-30191-2“˙37, https://doi.org/10.1007/978-3-540-30191-2_37
8. Das, P., Erwig, A., Faust, S., Loss, J., Riahi, S.: The exact security of BIP32 wallets. In: CCS '21. pp. 1020–1042 (2021). https://doi.org/10.1145/3460120.3484807
9. Das, P., Faust, S., Loss, J.: A formal treatment of deterministic wallets. In: CCS 2019. pp. 651–668 (2019). https://doi.org/10.1145/3319535.3354236
10. 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). https://doi.org/10.1109/DESEC.2018.8625151
11. 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). https://doi.org/10.1007/s10207-019-00476-5
12. Gentry, C., Silverberg, A.: Hierarchical id-based cryptography. In: ASIACRYPT 2002. pp. 548–566 (2002). https://doi.org/10.1007/3-540-36178-2“˙34, https://doi.org/10.1007/3-540-36178-2_34
13. getmonero.org: monero (April 2014), https://www.getmonero.org
14. golang.org: The go programming language (November 2009), https://golang.org
15. Gutoski, G., Stebila, D.: Hierarchical deterministic bitcoin wallets that tolerate key leakage. In: International Conference on Financial Cryptography and Data Security. pp. 497–504. Springer (2015)
16. Liu, W., Liu, Z., Nguyen, K., Yang, G., Yu, Y.: A lattice-based key-insulated and privacy-preserving signature scheme with publicly derived public key. In: ESORICS 2020, Part II. pp. 357–377 (2020). https://doi.org/10.1007/978-3-030-59013-0“˙18
17. 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)
18. Luzio, A.D., Francati, D., Ateniese, G.: Arcula: A secure hierarchical deterministic wallet for multi-asset blockchains. In: CANS 2020. pp. 323–343 (2020). https://doi.org/10.1007/978-3-030-65411-5“˙16
19. NIST: Fips pub 186-4, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf. Accessed 10 January 2021
20. Noether, S., Mackenzie, A.: Ring confidential transactions. Ledger, vol. 1, pp. 1-18 (2016)
21. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
22. Rückert, M.: Strongly unforgeable signatures and hierarchical identity-based signatures from lattices without random oracles. In: PQCrypto 2010. pp. 182–200 (2010). https://doi.org/10.1007/978-3-642-12929-2“˙14, https://doi.org/10.1007/978-3-642-12929-2_14

23. van Saberhagen, N.: Cryptonote v 2.0 (2013), https://cryptonote.org/whitepaper.pdf
24. Todd, P.: Stealth addresses. Post on Bitcoin development mailing list, https://www. mail-archive. com/bitcoindevelopment@ lists. sourceforge. net/msg03613. html (2014)
25. Unger, N.: The pbc go wrapper (December 2018), https://github.com/Nik-U/pbc
26. Waters, B.: Efficient identity-based encryption without random oracles. In: EUROCRYPT 2005. pp. 114–127 (2005). https://doi.org/10.1007/11426639‘‘7
27. Waters, B.: Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In: CRYPTO 2009. pp. 619–636 (2009). https://doi.org/10.1007/978-3-642-03356-8‘‘36, https://doi.org/10.1007/978-3-642-03356-8_36
28. Wuille, P.: Bip32: Hierarchical deterministic wallets. https://github.com/bitcoin/bips/blob/master/bip-0032. mediawiki (2012)

A Applications Use Cases

In this section, we would like to show that our HDWSA scheme can support the appealing use cases which have led to the popularity of HDW and SA. And most importantly, our HDWSA scheme does not cause any security concerns.

Hierarchical Manangement. For an organization[14] that employs our HDWSA scheme, the root administrator can generate the root wallet key pair, then generates wallet key pairs for its direct subordinates, without needing to be involved in the management of its indirect subroutines. And further, each entity only manages its direct subordinates. On the safety of coins, each entity has privilege to only the coins belonging to itself and its direct/indirect subordinates, and the collusion of a set of malicious entities cannot make them to access the coins of other entities.

Low-maintenance wallets with easy backup and recovery. To backup the wallets for the whole organization, the root administrator only needs to backup its root wallet secret key, say $(\alpha_{ID_0}, \beta_{ID_0}) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$. For any entity in the organization, say with identifier $ID = (id_0, \ldots, id_t)$ with $ID_{|0} = ID_0$, when necessary, for example, the device hosting ID's wallet crashes, the wallet key pair $(\mathsf{wpk}_{ID}, \mathsf{wssk}_{ID}, \mathsf{wsvk}_{ID})$ can be recovered from $(\alpha_{ID_0}, \beta_{ID_0})$ by repeatedly calling the $\mathsf{WalletKeyDelegate}()$ algorithm on $ID_{|i}$ for $i = 1, \ldots, t$. Then the entity can use $(\mathsf{wpk}_{ID}, \mathsf{wsvk}_{ID})$ to scan the blockchain, find the derived verification keys belonging to it, and fetch all the related coins.

Fresh cold-address generation. Each entity can publish its wallet public key wpk_{ID} and ask the payers to generate coin-addresses by themselves, without any concerns on its coins' safety or its privacy. It is worth mentioning that if a payer maliciously uses repeated coin-address to a target entity, the entity can detect it easily and refuse to acknowledge the transaction (e.g., refuse to ship the goods.)

Flexible trust-less audits. Note that the delegation of wallet secret view key and the delegation of wallet secret spend key are actually independent, say $\beta_{ID} = H_2(Q_{ID}, \beta_{ID_{|(t-1)}} Q_{ID})$ and $\alpha_{ID} = H_1(Q_{ID}, \alpha_{ID_{|(t-1)}} Q_{ID})$. This enables our HDWSA scheme to support flexible trust-less audits. In particular, the root administrator of an organization can reveal his wallet secret view key wsvk_{ID_0} to an auditor so that the auditor can delegate the wallet secret view key for each entity in the organization and then view all coins/transactions related to any entity in the organization. On the other side, the administrator can only reveal the wallet secret view key wsvk_{ID} for a specific entity (with identifier ID) to the auditor, so that the auditor can only view the coins/transactions related to the direct and indirect entities of the entity ID. While enjoying the functionality of such a flexible audits, the entities do not need to worry the safety of their coins, which is guaranteed by the strict and formal security models and proof of our HDWSA scheme.

[14] Note that an individual user can also use our HDWSA wallet, for example, manage his wallets in a hierarchy manner or just use the root wallet key pair.

B Detailed Security Proofs

B.1 Proof of Theorem 1

Now we give the details of Theorem 1 and its proof.

The HDWSA scheme is selective wallet existentially unforgeable under the CDH assumption in the random oracle model. Specifically, assume that there exists a t-time adversary $\mathcal{A}$ that makes q_{H_i} queries to random oracles $H_i (i = 0, 1, 3, 4)$, q_D queries to the wallet key delegate oracle, q_{vka} queries to the verification key adding oracle, q_C queries to the signing key corruption oracle, q_S queries to the signing oracle, and has a non-negligible success probability in the $\mathsf{Game}_{\mathsf{swEUF}}$, then we can construct a t'-time algorithm $\mathcal{B}$ that can solve the CDH problem with a non-negligible probability where $t' = 2t + O(t^* + q_{H_0} + q_{H_1} + q_{H_3} + q_D + q_{vka} + q_C + q_S)\tau_{mul} + O(q_{H_1} + q_D + q_{vka} + q_S)\tau_p$. Here τ_{mul} (τ_p, resp.) denotes the time of performing a scalar multiplication operation (a pairing operation, resp.).

Proof. Below we show that, if there exists a PPT adversary $\mathcal{A}$ that can win $\mathsf{Game}_{\mathsf{swEUF}}$ for our HDWSA construction with non-negligible advantage $\epsilon_{\mathcal{A}}$, then we can construct a PPT algorithm $\mathcal{B}$ that can solve the CDH problem with non-negligible probability.

■ **Setup.** $\mathcal{B}$ is given an instance of CDH problem on bilinear map groups, i.e., bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e})$ and a tuple $(P, A = aP, B = bP) \in \mathbb{G}_1^3$ for unknown $a, b \in \mathbb{Z}_p^*$, and the target of $\mathcal{B}$ is to compute an element $C \in \mathbb{G}_1$ such that $C = abP$.

$\mathcal{B}$ gives $\mathsf{PP} := \big((p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e}), H_0, H_1, H_2, H_3, H_4\big)$ to $\mathcal{A}$, where H_0, H_1, H_3 and H_4 are hash functions modeled as random oracles and H_2 is a collision-resistant hash function.

To model the hash functions H_0, H_1, H_3 and H_4 as random oracles, $\mathcal{B}$ initializes the following empty lists

- $L_{H_0} = \emptyset$, each element of which will be an (identifier, hash value, scalar value) tuple.
- $L_{H_1} = \emptyset$, each element of which will be an (input 1, input 2, hash value) tuple.
- $L_{H_1^*} = \emptyset$, each element of which will be an (input 1, input 2 information, hash value information) tuple.
- $L_{H_3} = \emptyset$, each element of which will be an (input 1, input 2, input 3, hash value, index, scalar value) tuple.
- $L_{H_4} = \emptyset$, each element of which will be an (input 1, input 2, input 3, hash value) tuple.

$\mathcal{B}$ initializes an empty set $L_{wk} = \emptyset$, each element of which will be an (identifier, wallet public key, wallet secret key information) tuple $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID})$.

$\mathcal{B}$ initializes an empty set $L_{dvk} = \emptyset$, each element of which will be a (derived verification key, identifier, index) tuple (dvk, ID, j), where the index j is used to identify the tuple in H_3 that dvk is based on.

$\mathcal{A}$ submits a Level-0 identifier for the target organization's root entity, say ID_0^*.

$\mathcal{B}$ chooses uniformly random $\alpha'_{ID_0^*}, \beta_{ID_0^*} \in \mathbb{Z}_p^*$, sets $A_{ID_0^*} = \alpha'_{ID_0^*} A$, $B_{ID_0^*} = \beta_{ID_0^*} P$, and gives $\mathsf{wpk}_{ID_0^*} = (A_{ID_0^*}, B_{ID_0^*})$ to $\mathcal{A}$.

$\mathcal{B}$ adds $\big(ID_0^*, \mathsf{wpk}_{ID_0^*}, (\alpha'_{ID^*}, \beta_{ID_0^*})\big)$ to L_{wk}. Note that actually $A_{ID_0^*} = (\alpha'_{ID_0^*} a)P$, we have that the (root) wallet secret key for $\mathsf{wpk}_{ID_0^*}$ is $\mathsf{wsk}_{ID_0^*} = (\alpha_{ID_0^*}, \beta_{ID_0^*})$, where $\mathsf{wssk}_{ID_0^*} = \alpha_{ID_0^*} = \alpha'_{ID_0^*} a$ is unknown to $\mathcal{B}$ while $\mathsf{wsvk}_{ID_0^*} = \beta_{ID_0^*}$ is known to $\mathcal{B}$.

$\mathcal{A}$ commits the identifier of the target wallet, i.e., an identifier $ID^* = (id_0^*, id_1^*, \ldots, id_{t^*}^*)$ with $t^* \geq 0$ and $ID_{|0}^* = ID_0^*$.

For $i = 1, \ldots, t^*$: [15]

1. $\mathcal{B}$ chooses a uniformly random $b_{ID_{|i}^*} \in \mathbb{Z}_p^*$, sets $Q_{ID_{|i}^*} = b_{ID_{|i}^*} B$, and adds $(ID_{|i}^*, Q_{ID_{|i}^*}, b_{ID_{|i}^*})$ to L_{H_0}, implicitly setting $H_0(ID_{|i}^*) = Q_{ID_{|i}^*} = b_{ID_{|i}^*} B$.

[15] Note that if $t^* = 0$, this step is skipped.

2. $\mathcal{B}$ chooses a uniformly random $\alpha'_{ID^*_{|i}} \in \mathbb{Z}_p^*$, and sets $A_{ID^*_{|i}} = \alpha'_{ID^*_{|i}} A$.
   Note that $\mathcal{B}$ is implicitly setting $H_1(Q_{ID^*_{|i}}, \alpha_{ID^*_{|(i-1)}} Q_{ID^*_{|i}}) = \alpha_{ID^*_{|i}}$ with $\alpha_{ID^*_{|i}} = \alpha'_{ID^*_{|i}} a$, where actually $\alpha_{ID^*_{|(i-1)}} = \alpha'_{ID^*_{|(i-1)}} a$ and $\alpha_{ID^*_{|(i-1)}} Q_{ID^*_{|i}} = \alpha'_{ID^*_{|(i-1)}} a \cdot b_{ID^*_{|i}} B$ are unknown to $\mathcal{B}$.
   $\mathcal{B}$ adds $(Q_{ID^*_{|i}}, \alpha'_{ID^*_{|(i-1)}}, \alpha'_{ID^*_{|i}})$ to $L_{H_1^*}$, since $\mathcal{B}$ is unable to compute the value of $\alpha_{ID^*_{|(i-1)}} Q_{ID^*_{|i}}$ or $\alpha_{ID^*_{|i}}$.
3. $\mathcal{B}$ computes $\beta_{ID^*_{|i}} = H_2(Q_{ID^*_{|i}}, \beta_{ID^*_{|(i-1)}} Q_{ID^*_{|i}})$ and $B_{ID^*_{|i}} = \beta_{ID^*_{|i}} P$.
4. $\mathcal{B}$ sets $\mathsf{wpk}_{ID^*_{|i}} = (A_{ID^*_{|i}}, B_{ID^*_{|i}})$, then gives $\mathsf{wpk}_{ID^*_{|i}}$ to $\mathcal{A}$.[16]
5. $\mathcal{B}$ adds $\big(ID^*_{|i}, \mathsf{wpk}_{ID^*_{|i}}, (\alpha'_{ID^*_{|i}}, \beta_{ID^*_{|i}})\big)$ to L_{wk}. Note that the wallet secret key for $\mathsf{wpk}_{ID^*_{|i}}$ is $\mathsf{wsk}_{ID^*_{|i}} = (\alpha_{ID^*_{|i}}, \beta_{ID^*_{|i}})$, where $\mathsf{wssk}_{ID^*_{|i}} = \alpha_{ID^*_{|i}} = \alpha'_{ID^*_{|i}} a$ is unknown to $\mathcal{B}$ while $\mathsf{wsvk}_{ID^*_{|i}} = \beta_{ID^*_{|i}}$ is known to $\mathcal{B}$.

In addition, $\mathcal{B}$ initializes $j = 0$, and chooses a uniformly random $j^* \in \{1, \ldots, q_{H_3} + q_{vka}\}$ as its guess that (1) the j^*-th tuple in L_{H_3}, say $(input_1^*, input_2^*, input_3^*, V^*, j^*, \eta_{j^*}) \in L_{H_3}$, will satisfy $input_1^* = B_{ID^*}$ AND $\hat{e}(input_1^*, input_2^*) = \hat{e}(P, input_3^*)$ and imply $H_3(input_1^*, input_2^*, input_3^*) = V^*$, and (2) in **Output Phase** $\mathcal{A}$ will output a forged signature with respect to a derived verification key dvk^* such that $(\mathsf{dvk}^*, ID^*, j^*) \in L_{dvk}$.

Note that, as shown below, $\mathcal{A}$ may trigger a H_3-query indirectly, by making a $\mathsf{ODVKAdd}$-query. That's why $\mathcal{B}$ guesses j^* in the scope $\{1, \ldots, q_{H_3} + q_{vka}\}$.

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

– $H_0(\cdot)$: when H_0 is queried on input an identifier $ID \in \mathcal{S}_{ID}$:
  - If there exists a corresponding tuple $(ID, Q_{ID}, b_{ID}) \in L_{H_0}$, Q_{ID} is returned.
  - Otherwise, $\mathcal{B}$ chooses a uniformly random $b_{ID} \in \mathbb{Z}_p^*$, sets $Q_{ID} = b_{ID} P$, adds (ID, Q_{ID}, b_{ID}) to L_{H_0}, and returns Q_{ID}.

  Note that for a tuple $(ID, Q_{ID}, b_{ID}) \in L_{H_0}$, we have $H_0(ID) = Q_{ID} = b_{ID} B$ for $ID \in \{ID^*_{|i} \mid 1 \le i \le t^*\}$ and $H_0(ID) = Q_{ID} = b_{ID} P$ for $ID \notin \{ID^*_{|i} \mid 1 \le i \le t^*\}$.

– $H_1(\cdot, \cdot)$: when H_1 is queried on input a pair $(input_1, input_2) \in \mathbb{G}_1 \times \mathbb{G}_1$:
  - If there exists a corresponding tuple $(input_1, input_2, hval) \in L_{H_1}$, $havl$ is returned.
  - Otherwise,
    - If $input_1 \notin \{Q_{ID^*_{|i}} \mid 1 \le i \le t^*\}$, then $\mathcal{B}$ chooses a uniformly random $hval \in \mathbb{Z}_p^*$, adds $(input_1, input_2, hval)$ to L_{H_1}, and returns $hval$,
    - Otherwise (i.e., $input_1 \in \{Q_{ID^*_{|i}} \mid 1 \le i \le t^*\}$), let $(Q_{ID^*_{|i}}, \alpha'_{ID^*_{|(i-1)}}, \alpha'_{ID^*_{|i}}) \in L_{H_1^*}$ be the corresponding tuple in $L_{H_1^*}$,
      - If $\underline{\hat{e}(P, input_2) = \hat{e}(\alpha'_{ID^*_{|(i-1)}} A, input_1)}$ (**Below we denote this event by E**): note that $input_1 = b_{ID^*_{|i}} B$ so that this implies $input_2 = \alpha'_{ID^*_{|(i-1)}} a \cdot b_{ID^*_{|i}} B$, $\mathcal{B}$ outputs $\frac{1}{\alpha'_{ID^*_{|(i-1)}} b_{ID^*_{|i}}} input_2$ as its solution for the CDH problem and aborts the game.
      - Otherwise, $\mathcal{B}$ chooses a uniformly random $hval \in \mathbb{Z}_p^*$, adds $(input_1, input_2, hval)$ to L_{H_1}, and returns $hval$.

  Note that for a tuple $(input_1, input_2, hval) \in L_{H_1}$, we have that $H_1(input_1, input_2) = hval$.
  Note that for a tuple $(input_1, input_2inf, hvalinf) \in L_{H_1^*}$, it implies that $(input_1, input_2inf, hvalinf) = (Q_{ID^*_{|i}} = b_{ID^*_{|i}} B, \alpha'_{ID^*_{|(i-1)}}, \alpha'_{ID^*_{|i}})$ for some $1 \le i \le t^*$ and $H_1(b_{ID^*_{|i}} B, \alpha'_{ID^*_{|(i-1)}} \cdot a \cdot b_{ID^*_{|i}} B) = \alpha'_{ID^*_{|i}} \cdot a$.
  Note that except that the event **E** happens, no one can provide the input 2 implied by $input_2inf$ to query H_1.

– $H_3(\cdot, \cdot, \cdot)$: when H_3 is queried on input a tuple $(input_1, input_2, input_3) \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1$:
  - If there exists a corresponding tuple $(input_1, input_2, input_3, V_j, j, \eta_j) \in L_{H_3}$, V_j is returned.
  - Otherwise, sets $j = j + 1$, i.e., suppose this is the j-th distinct inputs for H_3-query,
    - If $j \neq j^*$, then $\mathcal{B}$ chooses a uniformly random $\eta_j \in \mathbb{Z}_p^*$, sets $V_j = \eta_j P$, adds $(input_1, input_2, input_3, V_j, j, \eta_j)$ to L_{H_3}, and returns V_j.

[16] Note that as $\mathcal{A}$ commits ID^* as its target wallet, it will definitely makes $\mathsf{OWKeyDelegate}(\cdot)$-query on $ID^*_{|i}$ $(i = 1, \ldots, t^*)$ before the **Output Phase**. For simplicity of description, here we directly give $\mathsf{wpk}_{ID^*_{|i}}$ to $\mathcal{A}$.

    * Otherwise (i.e. $j = j^*$),
        - If $input_1 = B_{ID^*}$ AND $\hat{e}(input_1, input_2) = \hat{e}(P, input_3)$, then $\mathcal{B}$ chooses a uniformly random $\eta_j \in \mathbb{Z}_p^*$, sets $V_j = \eta_j B$, adds $(input_1, input_2, input_3, V_j, j, \eta_j)$ to L_{H_3}, and returns V_j.
        - Otherwise, $\mathcal{B}$ aborts.

  Note that for each tuple $(input_1, input_2, input_3, V_j, j, \eta_j) \in L_{H_3}$, we have that $H_3(input_1, input_2, input_3) = V_j = \eta_j P$ for $j \neq j^*$ and $H_3(input_1, input_2, input_3) = V_j = \eta_j B$ for $j = j^*$.
- $H_4(\cdot, \cdot)$: when H_4 is queried on input a tuple $(input_1, input_2, input_3) \in (\mathbb{G}_1 \times \mathbb{G}_2) \times \mathcal{M} \times \mathbb{G}_2$:
  - If there exists a corresponding tuple $(input_1, input_2, input_3, hval) \in L_{H_4}$, $hval$ is returned.
  - Otherwise, $\mathcal{B}$ chooses a uniformly random $hval \in \mathbb{Z}_p^*$, adds $(input_1, input_2, input_3, hval)$ to L_{H_4}, and returns $hval$.
- $\mathsf{OWKeyDelegate}(\cdot)$:
  When $\mathcal{A}$ makes an $\mathsf{OWKeyDelegate}(\cdot)$ query on an entity's identifier $ID \in \mathcal{S}_{ID}$ at level $\underline{t \geq 1}$ such that $ID_{|(t-1)} \in L_{wk}$:
  - If $ID \in L_{wk}$, i.e., the wallet key pair for ID has been previously generated (due to query on $\mathsf{OWKeyDelegate}(\cdot, \cdot)$ or the committed target wallet identifier ID^*), let $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ be the tuple corresponding to ID, $\mathcal{B}$ returns wpk_{ID} to $\mathcal{A}$. (Note that $\mathsf{WalletKeyDelegate}()$ is a deterministic algorithm.)
  - Otherwise, it implies that $ID \notin \{ID^*_{|i} \mid 1 \leq i \leq t^*\}$.
    Let $(ID_{|(t-1)}, \mathsf{wpk}_{ID_{|(t-1)}}, \mathsf{wskinf}_{ID_{|(t-1)}}) \in L_{wk}$ be the tuple corresponding to $ID_{|(t-1)}$, then
    * If $ID_{|(t-1)} \in \{ID^*_{|i} \mid 0 \leq i \leq t^*\}$:
      Note that this means $ID_{|(t-1)} = ID^*_{|(t-1)}$ where $0 \leq t-1 \leq t^*$, and $\mathsf{wskinf}_{ID_{|(t-1)}} = (\alpha'_{ID^*_{|(t-1)}}, \beta_{ID^*_{|(t-1)}})$. Note that the wallet secret key for $ID_{|(t-1)}$ is $\mathsf{wsk}_{ID^*_{|(t-1)}} = (\alpha_{ID^*_{|(t-1)}}, \beta_{ID^*_{|(t-1)}})$, where $\alpha_{ID^*_{|(t-1)}} = \alpha'_{ID^*_{|(t-1)}} a$ is unknown to $\mathcal{B}$.
      $\mathcal{B}$ proceeds as below:
      1. $\mathcal{B}$ makes H_0 query on ID. As $ID \notin \{ID^*_{|i} \mid 1 \leq i \leq t^*\}$, $\mathcal{B}$ obtains $Q_{ID} = H_0(ID) = b_{ID}P$ where the value of b_{ID} is known to $\mathcal{B}$. Note that with overwhelming probability, it holds that $Q_{ID} \notin \{Q_{ID^*_{|i}} \mid 1 \leq i \leq t^*\}$.
      2. $\mathcal{B}$ makes H_1 query on $(Q_{ID}, \alpha'_{ID^*_{|(t-1)}} b_{ID} A)$. As $Q_{ID} \notin \{Q_{ID^*_{|i}} \mid 1 \leq i \leq t^*\}$, $\mathcal{B}$ obtains a value $hval \in \mathbb{Z}_p^*$, which implies that $H_1(Q_{ID}, \alpha'_{ID^*_{|(t-1)}} b_{ID} A) = hval$. $\mathcal{B}$ sets $\alpha_{ID} = hval$.
         Note that actually $\alpha'_{ID^*_{|(t-1)}} b_{ID} A = \alpha'_{ID^*_{|(t-1)}} a \cdot b_{ID} P = \alpha_{ID^*_{|(t-1)}} \cdot Q_{ID}$, this means α_{ID} is generated as in the real scheme, although $\mathcal{B}$ does not know the wallet secret spend key $\alpha_{ID^*_{t-1}}$ for $ID^*_{|(t-1)}$ (i.e. $ID_{|(t-1)}$).
      3. $\mathcal{B}$ computes $\beta_{ID} = H_2(Q_{ID}, \beta_{ID^*_{|(t-1)}} Q_{ID})$.
      4. $\mathcal{B}$ sets $A_{ID} = \alpha_{ID} P$, $B_{ID} = \beta_{ID} P$, and $\mathsf{wpk}_{ID} = (A_{ID}, B_{ID})$, then gives wpk_{ID} to $\mathcal{A}$ and adds $\big(ID, \mathsf{wpk}_{ID}, (\alpha_{ID}, \beta_{ID})\big)$ into L_{wk}. Note that the wallet secret key information $\mathsf{wskinf}_{ID} = (\alpha_{ID}, \beta_{ID})$ is exactly the wallet secret key corresponding to wpk_{ID}.
    * Otherwise, i.e., $ID_{|(t-1)} \notin \{ID^*_{|i} \mid 0 \leq i \leq t^*\}$:
      Note that for such an identifier $ID_{|(t-1)}$, the wallet secret key information $\mathsf{wskinf}_{|(t-1)}$ is exactly the wallet secret key of $ID_{|(t-1)}$, $\mathcal{B}$ runs $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID}) \leftarrow \mathsf{WalletKeyDelegate}(ID, \mathsf{wpk}_{ID_{|(t-1)}}, \mathsf{wskinf}_{ID_{|(t-1)}})$, then gives wpk_{ID} to $\mathcal{A}$ and adds $\big(ID, \mathsf{wpk}_{ID}, \mathsf{wsk}_{ID}\big)$ into L_{wk}.

  Note that from above, for a tuple $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ such that $ID \notin \{ID^*_{|i} \mid 0 \leq i \leq t^*\}$, we have that the wallet secret key of ID is exactly wskinf_{ID}, while for a tuple $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ such that $ID \in \{ID^*_{|i} \mid 0 \leq i \leq t^*\}$, supposing $\mathsf{wskinf}_{ID} = (\alpha'_{ID}, \beta_{ID})$, we have that the wallet secret key of ID is $\mathsf{wsk}_{ID} = (\alpha_{ID}, \beta_{ID})$ with $\alpha_{ID} = \alpha'_{ID} \cdot a$.
- $\mathsf{OWskCorrupt}(\cdot)$:
  When $\mathcal{A}$ makes an $\mathsf{OWskCorrupt}(\cdot)$ query on input an entity's identifier $ID \in \mathcal{S}_{ID}$ at level $\underline{t \geq 0}$ such that ID in L_{wk}:
  Let $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ be the tuple corresponding to ID.

Note that $\mathcal{A}$ should not query $\mathsf{OWskCorrupt}(\cdot)$ on $ID^*_{|i}$ for any i such that $0 \le i \le t^*$ (as required by the condition for wining the game), which implies $ID \notin \{ID^*_{|0}, ID^*_{|1}, \ldots, ID^*_{|t^*}\}$, $\mathcal{B}$ directly returns wskinf_{ID} to $\mathcal{A}$, since as shown above, for such an identifier, wskinf_{ID} is exactly the wallet secret key.

– $\mathsf{OWsvkCorrupt}(\cdot)$:
When $\mathcal{A}$ makes an $\mathsf{OWsvkCorrupt}(\cdot)$ query on input an entity's identifier $ID \in \mathcal{S}_{ID}$ at level $\underline{t \ge 0}$ such that ID in L_{wk}:
Let $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ be the tuple corresponding to ID.
Note that as shown above, for both $ID \notin \{ID^*_{|0}, ID^*_{|1}, \ldots, ID^*_{|t^*}\}$ and $ID \in \{ID^*_{|0}, ID^*_{|1}, \ldots, ID^*_{|t^*}\}$, wskinf_{ID} gives the wallet secret view key $\mathsf{wsvk}_{ID} = \beta_{ID}$, $\mathcal{B}$ directly returns wsvk_{ID}.

– $\mathsf{ODVKAdd}(\cdot,\cdot)$:
When $\mathcal{A}$ makes an $\mathsf{ODVKAdd}(\cdot,\cdot)$ query on input a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ and an entity's identifier $ID \in \mathcal{S}_{ID}$ at level $\underline{t \ge 0}$ such that $ID \in L_{wk}$:
- If there is a corresponding tuple $(\mathsf{dvk}, \underline{ID, j}) \in L_{dvk}$ for some j, $\mathcal{B}$ returns 1 to $\mathcal{A}$.
- Otherwise, let $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ be the tuple corresponding to ID with $\mathsf{wpk}_{ID} = (A_{ID}, B_{ID})$, note that $\mathcal{B}$ knows the values of β_{ID} (which is stored in wskinf_{ID}) for the wallet secret key corresponding to wpk_{ID}, it proceeds as below: $\mathcal{B}$ simulates an H_3-query on input $(B_{ID}, Q_r, \beta_{ID} Q_r)$ and let V be the returned hash value and j be the index of the corresponding H_3 tuple. $\mathcal{B}$ checks whether $Q_{vk} \stackrel{?}{=} \hat{e}(V, -A_{ID})$ holds. If it holds, $\mathcal{B}$ adds (dvk, ID, j) into L_{dvk} and returns 1 to $\mathcal{A}$, otherwise, returns 0 to $\mathcal{A}$.

Note that for each tuple in L_{dvk}, say $\big(\mathsf{dvk}, ID, j\big)$ with $\mathsf{dvk} = (Q_r, Q_{vk})$, $ID \in \mathcal{S}_{ID}$ at level $t \ge 0$, supposing the wallet public key of ID is $\mathsf{wpk}_{ID} = (A_{ID}, B_{ID})$, the corresponding H_3 tuple, say $(input_1, input_2, input_3, V_j, j, \eta_j)$, satisfies $input_1 = B_{ID}$ AND $input_2 = Q_r$ AND $input_3 = \beta_{ID} Q_r$ AND $H_3(input_1, input_2, input_3) = V_j$ AND $Q_{vk} = \hat{e}(V_j, -A_{ID})$.
Also note that for any two different derived verification keys in L_{dvk}, say $\mathsf{dvk} = (Q_r, Q_{vk})$ and $\mathsf{dvk}' = (Q'_r, Q'_{vk})$ such that $\mathsf{dvk} \ne \mathsf{dvk}'$, supposing the corresponding tuples in L_{dvk} are $\big(\mathsf{dvk}, ID, j\big)$ and $\big(\mathsf{dvk}', ID', j'\big)$ respectively, we have that $j \ne j'$, i.e., they corresponds to different inputs to H_3. This is because, supposing the wallet public keys of ID and ID' are $\mathsf{wpk}_{ID} = (A_{ID}, B_{ID})$ and $\mathsf{wpk}_{ID'} = (A_{ID'}, B_{ID'})$ respectively, $j = j'$ implies $B_{ID} = B_{ID'}$ (which equals the $input_1$ of the H_3 tuple), $Q_r = Q'_r$ (which equals the $input_2$ of the H_3 tuple), and $V = V'$ (i.e. the H_3 hash value used in Q_{vk} computation). As $B_{ID} = B_{ID'}$ further implies $ID = ID'$ (since a uniformly random β_{ID} is chosen for each ID) and then $A_{ID} = A_{ID'}$, it is implied that $Q_{vk} = Q'_{vk}$. This means that $j = j'$ implies $(Q_r, Q_{vk}) = (Q'_r, Q'_{vk})$, which is contradictory to $\mathsf{dvk} \ne \mathsf{dvk}'$. In other words, **each tuple in L_{H_3} may correspond to at most one derived verification key in L_{dvk}.**

– $\mathsf{ODSKCorrput}(\cdot)$:
When $\mathcal{A}$ makes an $\mathsf{ODSKCorrput}(\cdot)$ query on input a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ such that $\mathsf{dvk} \in L_{dvk}$:
Let $\big(\mathsf{dvk}, ID, j\big) \in L_{dvk}$ be the tuple corresponding to dvk and further let $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ be the tuple corresponding to ID and $(input_1, input_2, input_3, V_j, j, \eta_j) \in L_{H_3}$ be the corresponding H_3 tuple, $\mathcal{B}$ proceeds as below:
- If $j = j^*$, $\mathcal{B}$ aborts, since it fails to guess the right dvk^* for which the adversary will output a forged signature in the **Output Phase**.
- Otherwise (i.e. $j \ne j^*$), note that $H_3(input_1, input_2, input_3) = V_j = \eta_j P$,
  * If $ID \notin \{ID^*_{|i} \mid 0 \le i \le t^*\}$, it implies that $\mathsf{wskinf}_{ID} = (\alpha_{ID}, \beta_{ID}) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ is exactly the wallet secret key of ID, then $\mathcal{B}$ computes $\mathsf{dsk} = \alpha_{ID} \cdot \eta_j P$ and returns dsk to $\mathcal{A}$.
  * If $ID \in \{ID^*_{|i} \mid 0 \le i \le t^*\}$, it implies that $\mathsf{wskinf}_{ID} = (\alpha'_{ID}, \beta_{ID}) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ such that the wallet secret key of ID is $\mathsf{wsk}_{ID} = (\alpha_{ID} = \alpha'_{ID} \cdot a, \beta_{ID})$, then $\mathcal{B}$ computes $\mathsf{dsk} = \alpha'_{ID} \cdot \eta_j A$ and returns dsk to $\mathcal{A}$.
    Note that actually $\mathsf{dsk} = (\alpha'_{ID} \cdot a) \cdot \eta_j P$, which means that such a derived signing key is generated as in the real game, although $\mathcal{B}$ does not know the wallet secret spend key α_{ID}.

– $\mathsf{OSign}(\cdot,\cdot)$:
When $\mathcal{A}$ makes an $\mathsf{OSign}(\cdot,\cdot)$ query on input a message $m \in \mathcal{M}$ and a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ such that $\mathsf{dvk} \in L_{dvk}$:

Let $\big(\mathsf{dvk}, ID, j\big) \in L_{dvk}$ be the tuple corresponding to dvk and further let $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ be the tuple corresponding to ID and $(input_1, input_2, input_3, V_j, j, \eta_j) \in L_{H_3}$ be the corresponding H_3 tuple, $\mathcal{B}$ proceeds as below:

- If $j \neq j^*$: note that, as shown above in $\mathsf{ODSKCorrput}(\cdot)$, $\mathcal{B}$ can compute the derived signing key dsk corresponding to dvk, $\mathcal{B}$ proceeds as below:
  1. $\mathcal{B}$ chooses a uniformly random $x \in \mathbb{Z}_p^*$, then compute $X = \hat{e}(xP, P)$.
  2. $\mathcal{B}$ chooses a uniformly random $h \in \mathbb{Z}_p^*$, and adds (dvk, m, X, h) into L_{H_4}, implicitly setting $H_4(\mathsf{dvk}, m, X) = h$.
  3. $\mathcal{B}$ computes $Q_\sigma = h \cdot \mathsf{dsk} + xP \in \mathbb{G}_1$, and returns (h, Q_σ) to $\mathcal{A}$.
- Otherwise (i.e., $j = j^*$): note that it implies that $ID = ID^*$, $\mathsf{wpk}_{ID} = (A_{ID^*} = \alpha'_{ID^*}A, B_{ID^*} = \beta_{ID^*}P)$, $\mathsf{wskinf}_{ID} = (\alpha'_{ID^*}, \beta_{ID^*})$, and the wallet secret key corresponding to wpk_{ID} is actually $\mathsf{wsk}_{ID} = (\alpha'_{ID^*}a, \beta_{ID^*})$, and it also implies that $input_1 = B_{ID^*}$, $input_2 = Q_r$, $\hat{e}(input_1, input_2) = \hat{e}(P, input_3)$, and $Q_{vk} = \hat{e}(V_j, -A_{ID^*})$, $\mathcal{B}$ proceeds as below:
  1. $\mathcal{B}$ chooses uniformly random $x_1, x_2 \in \mathbb{Z}_p^*$, then compute $X = \hat{e}(x_1A, B) \cdot \hat{e}(x_2P, P)$, implicitly setting $x = x_1ab + x_2$ such that $X = \hat{e}(xP, P)$.
  2. $\mathcal{B}$ sets $h = -\frac{x_1}{\alpha'_{ID^*}\eta_{j^*}} \in \mathbb{Z}_p^*$, and adds (dvk, m, X, h) into L_{H_4}, implicitly setting $H_4(\mathsf{dvk}, m, X) = h$.
  3. $\mathcal{B}$ sets $Q_\sigma = x_2P$, and returns (h, Q_σ) to $\mathcal{A}$.

  Note that actually $Q_\sigma = x_2P = -\frac{x_1}{\alpha'_{ID^*}\eta_{j^*}}\big((\alpha'_{ID^*}a) \cdot (\eta_{j^*}B)\big) + (x_1ab + x_2)P$ $= h \cdot \big(\alpha_{ID^*} \cdot H_3(B_{ID^*}, Q_r, \beta_{ID^*}Q_r)\big) + xP$, which implies that the signature (h, Q_σ) is generated as in the real scheme, although $\mathcal{B}$ does not know the derived signing key corresponding to dvk.

■ **Output Phase.** $\mathcal{A}$ outputs a message $m^* \in \mathcal{M}$, a signature $\sigma^* = (h^*, Q_\sigma^*)$, and a derived verification key $\mathsf{dvk}^* = (Q_r^*, Q_{vk}^*)$ such that $(\mathsf{dvk}^*, ID^*, j) \in L_{dvk}$ for some index j, $\mathsf{Verify}(m^*, \sigma^*, \mathsf{dvk}^*) = 1$, $\mathsf{ODSKCorrupt}(\mathsf{dvk}^*)$ is never queried, and $\mathsf{OSign}(m^*, \mathsf{dvk}^*)$ is never queried (**Denote this event by F**).

If $j \neq j^*$, $\mathcal{B}$ aborts, since it implies that $\mathcal{B}$ did not guess the right dvk^*.

Otherwise (i.e. $j = j^*$), $\mathcal{B}$ rewinds $\mathcal{A}$ to the point where $h^* = H_4(\mathsf{dvk}^*, m^*, X^*)$ is queried and returns a new uniformly random $\tilde{h}^*$ as the hash value for $(\mathsf{dvk}^*, m^*, X^*)$. Let frk denote the event that $\mathcal{A}$ returns another valid signature $\tilde{\sigma}^* = (\tilde{h}^*, \tilde{Q_\sigma}^*)$ with regards to the same $(\mathsf{dvk}^*, m^*, X^*)$. Then we have

$$X^* = \hat{e}(Q_\sigma^*, P) \cdot (Q_{vk}^*)^{h^*} = \hat{e}(\tilde{Q_\sigma}^*, P) \cdot (Q_{vk}^*)^{\tilde{h}^*}$$

which gives

$$\begin{aligned} Q_{vk}^* &= \hat{e}((\tilde{h}^* - h^*)^{-1}(Q_\sigma^* - \tilde{Q_\sigma}^*), P) \\ &= \hat{e}(\eta_{j^*}B, -\alpha'_{ID^*}A). \end{aligned}$$

Hence, $\mathcal{B}$ can extract the solution for the CDH problem with regards to $(P, A, B) \in \mathbb{G}_1^3$ as

$$-(\eta_{j^*}\alpha'_{ID^*})^{-1}(\tilde{h}^* - h^*)^{-1}(Q_\sigma^* - \tilde{Q_\sigma}^*).$$

Probability Analysis. Let succ denote the event $\neg\mathbf{E} \wedge \mathbf{F} \wedge (j = j^*)$. Then we have

$$\begin{aligned} \Pr[\mathsf{succ}] &= \Pr[\neg\mathbf{E} \wedge \mathbf{F} \wedge (j = j^*)] \\ &= \Pr[\mathbf{F} \wedge (j = j^*)|\neg\mathbf{E}] \Pr[\neg\mathbf{E}] \\ &= \Pr[\mathbf{F} \wedge (j = j^*)|\neg\mathbf{E}](1 - \Pr[\mathbf{E}]) \\ &\geq \Pr[\mathbf{F} \wedge (j = j^*)|\neg\mathbf{E}] - \Pr[\mathbf{E}] \\ &\geq \epsilon_{\mathcal{A}}/(q_{H_3} + q_{vka}) - \Pr[\mathbf{E}] \end{aligned}$$

It is easy to see that if event **E** happens, then we can construct an algorithm $\mathcal{B}$ that can directly solve the CDH problem by using the H_1 queries made by $\mathcal{A}$. Hence we have that event **E** happens only with a negligible probability negl. Based on the assumption that $\epsilon_{\mathcal{A}}$ is non-negligible, we have

$$\epsilon_{\mathsf{succ}} = \Pr[\mathsf{succ}] \geq \epsilon_{\mathcal{A}}/(q_{H_3} + q_{vka}) - \mathsf{negl}$$

which is also non-negligible. Then based on the General Forking Lemma [2], we have

$$\Pr[\mathsf{frk}] \geq \epsilon_{\mathsf{succ}}(\frac{\epsilon_{\mathsf{succ}}}{q_{H_4}} - \frac{1}{p-1})$$

which is non-negligible. $\mathcal{B}$ can extract the solution of CDH problem under the event frk and $\mathcal{B}$'s running time is bounded by $2t + O(t^* + q_{H_0} + q_{H_1} + q_{H_3} + q_D + q_{vka} + q_C + q_S)\tau_{mul} + O(q_{H_1} + q_D + q_{vka} + q_S)\tau_p$ where τ_{mul} and τ_p denote the time of performing a scalar multiplication operation and a pairing operation, respectively.

B.2 Proof of Theorem 2

Now we give the details of Theorem 2 and its proof.

The HDWSA scheme is selective wallet unlinkable under the CDH assumption in the random oracle model. Specifically, assume that there exists a t-time adversary $\mathcal{A}$ that makes q_{H_i} queries to random oracles $H_i(i = 0, 2, 3)$, q_D queries to the wallet key delegate oracle, q_{vka} queries to the verification key adding oracle, q_S queries to the signing oracle, and has a non-negligible advantage in the $\mathsf{Game_{swWUNL}}$, then we can construct another t'-time adversary $\mathcal{B}$ that solves the CDH problem with a non-negligible probability where $t' = t + O(t^{(0)} + t^{(1)} + q_{H_0} + q_{H_2} + q_D + q_{vka} + q_S)\tau_{mul} + O(q_{H_2} + q_{H_3} + q_D + q_{vka} + q_S)\tau_p$.

Proof. Below we show that, if there exists a PPT adversary $\mathcal{A}$ that can win $\mathsf{Game_{swWUNL}}$ for our HDWSA construction with non-negligible advantage $\epsilon_{\mathcal{A}}$, then we can construct a PPT algorithm $\mathcal{B}$ that can solve the CDH problem with non-negligible probability.

■ **Setup.** $\mathcal{B}$ is given an instance of CDH problem on bilinear map groups, i.e., bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e})$ and a tuple $(P, A = aP, B = bP) \in \mathbb{G}_1^3$ for unknown $a, b \in \mathbb{Z}_p^*$, and the target of $\mathcal{B}$ is to compute an element $C \in \mathbb{G}_1$ such that $C = abP$.

$\mathcal{B}$ gives $\mathsf{PP} := ((p, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e}), H_0, H_1, H_2, H_3, H_4)$ to $\mathcal{A}$, where H_0, H_2, H_3 are hash functions modeled as random oracles and H_1, H_4 are collision-resistant hash functions.

To model the hash functions H_0, H_2, H_3 as random oracles, $\mathcal{B}$ initializes the following empty lists

- $L_{H_0} = \emptyset$, each element of which will be an (identifier, hash value, scalar value) tuple.
- $L_{H_2} = \emptyset$, each element of which will be an (input 1, input 2, hash value) tuple.
- $L_{H_2^*} = \emptyset$, each element of which will be an (input 1, input 2 information, hash value information) tuple.
- $L_{H_3} = \emptyset$, each element of which will be an (input 1, input 2, input 3, hash value) tuple.

$\mathcal{B}$ initializes an empty set $L_{wk} = \emptyset$, each element of which will be an (identifier, wallet public key, wallet secret key information) tuple $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID})$.

$\mathcal{B}$ initializes an empty set $L_{dvk} = \emptyset$, each element of which will be a (derived verification key, identifier, derived signing key) tuple $(\mathsf{dvk}, ID, \mathsf{dsk})$.

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

For $k = 0, 1$:

1. $\mathcal{B}$ chooses uniformly random $\alpha_{ID_0^{*(k)}}, \beta'_{ID_0^{*(k)}} \in \mathbb{Z}_p^*$, sets $A_{ID_0^{*(k)}} = \alpha_{ID_0^{*(k)}} P$, $B_{ID_0^{*(k)}} = \beta'_{ID_0^{*(k)}} B$, and gives $\mathsf{wpk}_{ID_0^{*(k)}} = (A_{ID_0^{*(k)}}, B_{ID_0^{*(k)}})$ to $\mathcal{A}$.
2. $\mathcal{B}$ adds $\big(ID_0^{*(k)}, \mathsf{wpk}_{ID_0^{*(k)}}, (\alpha_{ID_0^{*(k)}}, \beta'_{ID_0^{*(k)}})\big)$ to L_{wk}. Note that the (root) wallet secret key for $\mathsf{wpk}_{ID_0^{*(k)}}$ is $\mathsf{wsk}_{ID_0^{*(k)}} = (\alpha_{ID_0^{*(k)}}, \beta_{ID_0^{*(k)}})$, where $\beta_{ID_0^{*(k)}} = \beta'_{ID_0^{*(k)}} b$ is unknown to $\mathcal{B}$.

$\mathcal{A}$ commits two challenge wallet identifiers, namely, $ID^{(0)} = (id_0^{(0)}, id_1^{(0)}, \ldots, id_{t^{(0)}}^{(0)})$ and $ID^{(1)} = (id_0^{(1)}, id_1^{(1)}, \ldots, id_{t^{(1)}}^{(1)})$ such that $t^{(0)} \geq 0, t^{(1)} \geq 0$, and $ID_{|0}^{(0)}, ID_{|0}^{(1)} \in \{ID_0^{*(0)}, ID_0^{*(1)}\}$.

For $i = 1, \ldots, t^{(0)}$, [17] consider the identifier $ID_{|i}^{(0)}$:

[17] Note that if $t^{(0)} = 0$, this step is skipped.

1. $\mathcal{B}$ chooses a uniformly random $a_{ID^{(0)}_{|i}} \in \mathbb{Z}_p^*$, sets $Q_{ID^{(0)}_{|i}} = a_{ID^{(0)}_{|i}}A$, and adds $(ID^{(0)}_{|i}, Q_{ID^{(0)}_{|i}}, a_{ID^{(0)}_{|i}})$ to L_{H_0}, implicitly setting $H_0(ID^{(0)}_{|i}) = Q_{ID^{(0)}_{|i}} = a_{ID^{(0)}_{|i}}A$.
2. $\mathcal{B}$ computes $\alpha_{ID^{(0)}_{|i}} = H_1(Q_{ID^{(0)}_{|i}}, \alpha_{ID^{(0)}_{|(i-1)}} Q_{ID^{(0)}_{|i}})$ and sets $A_{ID^{(0)}_{|i}} = \alpha_{ID^{(0)}_{|i}}P$.
3. $\mathcal{B}$ chooses a uniformly random $\beta'_{ID^{(0)}_{|i}} \in \mathbb{Z}_p^*$ and sets $B_{ID^{(0)}_{|i}} = \beta'_{ID^{(0)}_{|i}}B$.
   Note that $\mathcal{B}$ is implicitly setting $H_2(Q_{ID^{(0)}_{|i}}, \beta_{ID^{(0)}_{|(i-1)}} Q_{ID^{(0)}_{|i}}) = \beta_{ID^{(0)}_{|i}}$ with $\beta_{ID^{(0)}_{|i}} = \beta'_{ID^{(0)}_{|i}}b$, where actually $\beta_{ID^{(0)}_{|(i-1)}} = \beta'_{ID^{(0)}_{|(i-1)}} b$ and $\beta_{ID^{(0)}_{|(i-1)}} Q_{ID^{(0)}_{|i}} = \beta'_{ID^{(0)}_{|(i-1)}} b \cdot a_{ID^{(0)}_{|i}}A$ are unknown to $\mathcal{B}$.
   $\mathcal{B}$ adds $(Q_{ID^{(0)}_{|i}}, \beta'_{ID^{(0)}_{|(i-1)}}, \beta'_{ID^{(0)}_{|i}})$ to $L_{H_2^*}$, since $\mathcal{B}$ is unable to compute the value of $\beta_{ID^{(0)}_{|(i-1)}} Q_{ID^{(0)}_{|i}}$ or $\beta_{ID^{(0)}_{|i}}$.
4. $\mathcal{B}$ sets $\mathsf{wpk}_{ID^{(0)}_{|i}} = (A_{ID^{(0)}_{|i}}, B_{ID^{(0)}_{|i}})$, then gives $\mathsf{wpk}_{ID^{(0)}_{|i}}$ to $\mathcal{A}$.
5. $\mathcal{B}$ adds $\big(ID^{(0)}_{|i}, \mathsf{wpk}_{ID^{(0)}_{|i}}, (\alpha_{ID^{(0)}_{|i}}, \beta'_{ID^{(0)}_{|i}})\big)$ to L_{wk}. Note that the wallet secret key for $\mathsf{wpk}_{ID^{(0)}_{|i}}$ is $\mathsf{wsk}_{ID^{(0)}_{|i}} = (\alpha_{ID^{(0)}_{|i}}, \beta_{ID^{(0)}_{|i}})$, where $\beta_{ID^{(0)}_{|i}} = \beta'_{ID^{(0)}_{|i}}b$ is unknown to $\mathcal{B}$.

For $i = 1, \ldots, t^{(1)}$, [18] consider the identifier $ID^{(1)}_{|i}$: if $ID^{(1)}_{|i} \notin L_{wk}$, then $\mathcal{B}$ executes the above 5 steps on identifier $ID^{(1)}_{|i}$ (literally, replacing '(0)' with '(1)').

■ **Probing Phase 1.** $\mathcal{A}$ can adaptively query the following oracles:

– $H_0(\cdot)$: when H_0 is queried on input an identifier $ID \in \mathcal{S}_{ID}$:
  - If there exists a corresponding tuple $(ID, Q_{ID}, a_{ID}) \in L_{H_0}$, Q_{ID} is returned.
  - Otherwise, $\mathcal{B}$ chooses a uniformly random $a_{ID} \in \mathbb{Z}_p^*$, sets $Q_{ID} = a_{ID}P$, adds (ID, Q_{ID}, a_{ID}) to L_{H_0}, and returns Q_{ID}.

  Note that for a tuple $(ID, Q_{ID}, a_{ID}) \in L_{H_0}$, we have $H_0(ID) = Q_{ID} = a_{ID}A$ for $ID \in \{ID^{(0)}_{|i} \mid 1 \le i \le t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 1 \le i \le t^{(1)}\}$ and $H_0(ID) = Q_{ID} = a_{ID}P$ for $ID \notin \{ID^{(0)}_{|i} \mid 1 \le i \le t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 1 \le i \le t^{(1)}\}$.

– $H_2(\cdot, \cdot)$: when H_2 is queried on input a pair $(input_1, input_2) \in \mathbb{G}_1 \times \mathbb{G}_1$:
  - If there exists a corresponding tuple $(input_1, input_2, hval) \in L_{H_2}$, $havl$ is returned.
  - Otherwise,
    * If $input_1 \notin \{Q_{ID^{(0)}_{|i}} \mid 1 \le i \le t^{(0)}\} \cup \{Q_{ID^{(1)}_{|i}} \mid 1 \le i \le t^{(1)}\}$, then $\mathcal{B}$ chooses a uniformly random $hval \in \mathbb{Z}_p^*$, adds $(input_1, input_2, hval)$ to L_{H_2}, and returns $hval$,
    * Otherwise (i.e., $input_1 \in \{Q_{ID^{(0)}_{|i}} \mid 1 \le i \le t^{(0)}\} \cup \{Q_{ID^{(1)}_{|i}} \mid 1 \le i \le t^{(1)}\}$), let $(Q_{ID^{(k)}_{|i}}, \beta'_{ID^{(k)}_{|(i-1)}}, \beta'_{ID^{(k)}_{|i}}) \in L_{H_2^*}$ (for some $k \in \{0,1\}$ and $i \in \{1, \ldots, t^{(k)}\}$) be the corresponding tuple in $L_{H_2^*}$,
      · If $\underline{\hat{e}(P, input_2) = \hat{e}(\beta'_{ID^{(k)}_{|(i-1)}} B, input_1)}$ (**Below we denote this event by $\mathbf{E_2}$**): note that $input_1 = a_{ID^{(k)}_{|i}}A$ so that this implies $input_2 = \beta'_{ID^{(k)}_{|(i-1)}} b \cdot a_{ID^{(k)}_{|i}}A$,
        $\mathcal{B}$ outputs $\frac{1}{\beta'_{ID^{(k)}_{|(i-1)}} a_{ID^{(k)}_{|i}}} input_2$ as its solution for the CDH problem and aborts the game.
      · Otherwise, $\mathcal{B}$ chooses a uniformly random $hval \in \mathbb{Z}_p^*$, adds $(input_1,\ input_2, hval)$ to L_{H_2}, and returns $hval$.

  Note that for a tuple $(input_1, input_2, hval) \in L_{H_2}$, we have that $H_2(input_1,\ input_2) = hval$.
  Note that for a tuple $(input_1, input_2inf, hvalinf) \in L_{H_2^*}$, it implies that $(input_1, input_2inf, hvalinf) = (Q_{ID^{(k)}_{|i}} = a_{ID^{(k)}_{|i}}A,\ \beta'_{ID^{(k)}_{|(i-1)}}, \beta'_{ID^{(k)}_{|i}})$ for some $k \in \{0,1\}$ and $1 \le i \le t^{(k)}$ and $H_2(a_{ID^{(k)}_{|i}}A, \beta'_{ID^{(k)}_{|(i-1)}} \cdot b \cdot a_{ID^{(k)}_{|i}}A) = \beta'_{ID^{(k)}_{|i}} \cdot b$. Note that except that the event $\mathbf{E_2}$ happens, no one can provide the input 2 implied by $input_2inf$ to query H_2.

[18] Note that if $t^{(1)} = 0$, this step is skipped.

- $H_3(\cdot,\cdot,\cdot)$: when H_3 is queried on input a tuple $(input_1, input_2, input_3) \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1$:
  - If there exists a corresponding tuple $(input_1, input_2, input_3, V) \in L_{H_3}$, V is returned.
  - Otherwise, $\mathcal{B}$ chooses a uniformly random $V \in \mathbb{G}_1^*$, adds $(input_1, input_2, input_3, V)$ to L_{H_3}, and returns V.

  Note that for each tuple $(input_1, input_2, input_3, V) \in L_{H_3}$, we have that $H_3(input_1, input_2, input_3) = V$.
- $\mathsf{OWKeyDelegate}(\cdot)$:
  When $\mathcal{A}$ makes an $\mathsf{OWKeyDelegate}(\cdot)$ query on an entity's identifier $ID \in \mathcal{S}_{ID}$ at level $t \geq 1$ such that $ID_{|(t-1)} \in L_{wk}$:
  - If $ID \in L_{wk}$, i.e., the wallet key pair for ID has been previously generated (due to query on $\mathsf{OWKeyDelegate}(\cdot,\cdot)$ or the committed challenge wallet identifiers $ID^{(0)}$ and $ID^{(1)}$), let $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ be the tuple corresponding to ID, $\mathcal{B}$ return wpk_{ID} to $\mathcal{A}$.
    (Note that $\mathsf{WalletKeyDelegate}()$ is a deterministic algorithm.)
  - Otherwise, it implies that $ID \notin \{ID^{(0)}_{|i} \mid 1 \leq i \leq t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 1 \leq i \leq t^{(1)}\}$.
    Let $(ID_{|(t-1)}, \mathsf{wpk}_{ID_{|(t-1)}}, \mathsf{wskinf}_{ID_{|(t-1)}}) \in L_{wk}$ be the tuple corresponding to $ID_{|(t-1)}$, then
    * If $ID_{|(t-1)} \in \{ID^{(0)}_{|i} \mid 0 \leq i \leq t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 0 \leq i \leq t^{(1)}\}$:
      Note that this means $ID_{|(t-1)} = ID^{(k)}_{|(t-1)}$ with $0 \leq t-1 \leq t^{(k)}$ for some $k \in \{0,1\}$, and $\mathsf{wskinf}_{ID_{|(t-1)}} = (\alpha_{ID^{(k)}_{|(t-1)}}, \beta'_{ID^{(k)}_{|(t-1)}})$. Note that the wallet secret key for $ID_{|(t-1)}$ is $\mathsf{wsk}_{ID^{(k)}_{|(t-1)}} = (\alpha_{ID^{(k)}_{|(t-1)}}, \beta_{ID^{(k)}_{|(t-1)}})$, where
      $\beta_{ID^{(k)}_{|(t-1)}} = \beta'_{ID^{(k)}_{|(t-1)}} b$ is unknown to $\mathcal{B}$.
      $\mathcal{B}$ proceeds as below:
      1. $\mathcal{B}$ makes H_0-query on ID.
         As $ID \notin \{ID^{(0)}_{|i} \mid 1 \leq i \leq t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 1 \leq i \leq t^{(1)}\}$, $\mathcal{B}$ obtains $Q_{ID} = H_0(ID) = a_{ID}P$ where the value of a_{ID} is known to $\mathcal{B}$.
         Note that with overwhelming probability, it holds that $Q_{ID} \notin \{Q_{ID^{(0)}_{|i}} \mid 1 \leq i \leq t^{(0)}\} \cup \{Q_{ID^{(1)}_{|i}} \mid 1 \leq i \leq t^{(1)}\}$.
      2. $\mathcal{B}$ computes $\alpha_{ID} = H_1(Q_{ID}, \alpha_{ID^{(k)}_{|(t-1)}} Q_{ID})$.
      3. $\mathcal{B}$ makes H_2 query on $(Q_{ID}, \beta'_{ID^{*}_{|(t-1)}} a_{ID}B)$.
         As $Q_{ID} \notin \{Q_{ID^{(0)}_{|i}} \mid 1 \leq i \leq t^{(0)}\} \cup \{Q_{ID^{(1)}_{|i}} \mid 1 \leq i \leq t^{(1)}\}$, $\mathcal{B}$ obtains a value $hval \in \mathbb{Z}_p^*$, which implies $H_2(Q_{ID}, \beta'_{ID^{(k)}_{|(t-1)}} a_{ID}B) = hval$.
         $\mathcal{B}$ sets $\beta_{ID} = hval$.
         Note that actually $\beta'_{ID^{(k)}_{|(t-1)}} a_{ID}B = \beta'_{ID^{(k)}_{|(t-1)}} b \cdot a_{ID}P = \beta_{ID^{(k)}_{|(t-1)}} \cdot Q_{ID}$, this means β_{ID} is generated as in the real scheme, although $\mathcal{B}$ does not know the wallet secret view key $\beta_{ID^{(k)}_{|(t-1)}}$ for $ID^{(k)}_{|(t-1)}$ (i.e. $ID_{|(t-1)}$).
      4. $\mathcal{B}$ sets $A_{ID} = \alpha_{ID}P$, $B_{ID} = \beta_{ID}P$, and $\mathsf{wpk}_{ID} = (A_{ID}, B_{ID})$, then gives wpk_{ID} to $\mathcal{A}$ and adds $\big(ID, \mathsf{wpk}_{ID}, (\alpha_{ID}, \beta_{ID})\big)$ into L_{wk}. Note that the wallet secret key information $\mathsf{wskinf}_{ID} = (\alpha_{ID}, \beta_{ID})$ is exactly the wallet secret key corresponding to wpk_{ID}.
    * Otherwise, i.e., $ID_{|(t-1)} \notin \{ID^{(0)}_{|i} \mid 0 \leq i \leq t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 0 \leq i \leq t^{(1)}\}$:
      Note that for such an identifier $ID_{|(t-1)}$, the wallet secret key information $\mathsf{wskinf}_{|(t-1)}$ is exactly the wallet secret key of $ID_{|(t-1)}$, $\mathcal{B}$ runs $(\mathsf{wpk}_{ID}, \mathsf{wsk}_{ID}) \leftarrow \mathsf{WalletKeyDelegate}(ID, \mathsf{wpk}_{ID_{|(t-1)}}, \mathsf{wskinf}_{ID_{|(t-1)}})$, then gives wpk_{ID} to $\mathcal{A}$ and adds $\big(ID, \mathsf{wpk}_{ID}, \mathsf{wsk}_{ID}\big)$ into L_{wk}.

Note that from above, for a tuple $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ such that $ID \notin \{ID^{(0)}_{|i} \mid 0 \leq i \leq t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 0 \leq i \leq t^{(1)}\}$, we have that the wallet secret key of ID is exactly wskinf_{ID}, while for a tuple $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ such that $ID \in \{ID^{(0)}_{|i} \mid 0 \leq i \leq t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 0 \leq i \leq t^{(1)}\}$, supposing

$\mathsf{wskinf}_{ID} = (\alpha_{ID}, \beta'_{ID})$, we have that the wallet secret key of ID is $\mathsf{wsk}_{ID} = (\alpha_{ID}, \beta_{ID})$ with $\beta_{ID} = \beta'_{ID} \cdot b$ for the unknown $b \in \mathbb{Z}_p^*$.

- $\mathsf{OWskCorrupt}(\cdot)$:
  When $\mathcal{A}$ makes an $\mathsf{OWskCorrupt}(\cdot)$ query on input an entity's identifier $ID \in \mathcal{S}_{ID}$ at level $\underline{t \geq 0}$ such that $ID \in L_{wk}$:
  Let $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ be the tuple corresponding to ID.
  Note that $\mathcal{A}$ shall only query $\mathsf{OWskCorrupt}(\cdot)$ on ID such that $ID \notin \{ID^{(0)}_{|i} \mid 0 \leq i \leq t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 0 \leq i \leq t^{(1)}\}$, $\mathcal{B}$ directly returns wskinf_{ID} to $\mathcal{A}$, since as shown above, for such an identifier, wskinf_{ID} is exactly the wallet secret key.
- $\mathsf{OWsvkCorrupt}(\cdot)$:
  When $\mathcal{A}$ makes an $\mathsf{OWsvkCorrupt}(\cdot)$ query on input an entity's identifier $ID \in \mathcal{S}_{ID}$ at level $\underline{t \geq 0}$ such that $ID \in L_{wk}$:
  Let $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ be the tuple corresponding to ID.
  Note that $\mathcal{A}$ shall only query $\mathsf{OWsvkCorrupt}(\cdot)$ on ID such that $ID \notin \{ID^{(0)}_{|i} \mid 0 \leq i \leq t^{(0)}\} \cup \{ID^{(1)}_{|i} \mid 0 \leq i \leq t^{(1)}\}$, $\mathcal{B}$ directly returns the wallet secret view key β_{ID} in wskinf_{ID} to $\mathcal{A}$, since as shown above, for such an identifier, wskinf_{ID} is exactly the wallet secret key.
- $\mathsf{ODVKAdd}(\cdot, \cdot)$:
  When $\mathcal{A}$ makes an $\mathsf{ODVKAdd}(\cdot, \cdot)$ query on input a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ and an entity's identifier $ID \in \mathcal{S}_{ID}$ at level $\underline{t \geq 0}$ such that $ID \in L_{wk}$:
  - If there is a corresponding tuple $(\mathsf{dvk}, ID, \mathsf{dsk}) \in L_{dvk}$ for some dsk, $\mathcal{B}$ returns 1 to $\mathcal{A}$.
  - Otherwise, let $(ID, \mathsf{wpk}_{ID}, \mathsf{wskinf}_{ID}) \in L_{wk}$ be the tuple corresponding to ID with $\mathsf{wpk}_{ID} = (A_{ID}, B_{ID})$, note that $\mathcal{B}$ knows the values of α_{ID} (which is stored in wskinf_{ID}) for the wallet secret key corresponding to wpk_{ID}, it proceeds as below:
    - If there is a tuple $(input_1, input_2, input_3, V) \in L_{H_3}$ such that the three inputs satisfy $input_1 = B_{ID}, input_2 = Q_r$ and $\hat{e}(input_1, input_2) = \hat{e}(P, input_3)$,
      $\mathcal{B}$ checks whether $Q_{vk} \overset{?}{=} \hat{e}(V, -A_{ID})$ holds.
      - If it holds, $\mathcal{B}$ computes $\mathsf{dsk} = \alpha_{ID} \cdot V$, adds $(\mathsf{dvk}, ID, \mathsf{dsk})$ into L_{dvk}, and returns 1 to $\mathcal{A}$. Note that dsk is exactly the derived signing key corresponding to dvk.
      - Otherwise, $\mathcal{B}$ returns 0 to $\mathcal{A}$.
    - Otherwise (i.e. there does not exist such a tuple in L_{H_3}),
      $\mathcal{B}$ returns 0 to $\mathcal{A}$.
      Note that when $\mathcal{A}$ makes a H_3-query on these three inputs, a uniformly random value in $\mathbb{G}_1^*$ will be chosen as the returned hash value. Thus, without making a H_3-query that produces such a tuple, the chance that $\mathsf{dvk} = (Q_r, Q_{vk})$ satisfies $Q_{vk} = \hat{e}(H_3(B_{ID}, Q_r, input_3), -A_{ID})$ is negiligible.
- $\mathsf{ODSKCorrput}(\cdot)$:
  When $\mathcal{A}$ makes an $\mathsf{ODSKCorrput}(\cdot)$ query on input a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ such that $\mathsf{dvk} \in L_{dvk}$:
  Let $\big(\mathsf{dvk}, ID, \mathsf{dsk}\big) \in L_{dvk}$ be the tuple corresponding to dvk, $\mathcal{B}$ just returns dsk to $\mathcal{A}$.
- $\mathsf{OSign}(\cdot, \cdot)$:
  When $\mathcal{A}$ makes an $\mathsf{OSign}(\cdot, \cdot)$ query on input a message $m \in \mathcal{M}$ and a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ such that $\mathsf{dvk} \in L_{dvk}$:
  Let $\big(\mathsf{dvk}, ID, \mathsf{dsk}\big) \in L_{dvk}$ be the tuple corresponding to dvk, $\mathcal{B}$ just runs $(h, Q_\sigma) \leftarrow \mathsf{Sign}(m, \mathsf{dvk}, \mathsf{dsk})$ and returns (h, Q_σ) to $\mathcal{A}$.

■ **Challenge Phase.** A random bit $c \in \{0, 1\}$ is chosen.
Note that for identifier $ID^{(c)}$, the corresponding tuple in L_{wk} is $(ID^{(c)}, \mathsf{wpk}_{ID^{(c)}}, \mathsf{wskinf}_{ID^{(c)}}) \in L_{wk}$, where $\mathsf{wpk}_{ID^{(c)}} = (A_{ID^{(c)}}, B_{ID^{(c)}})$, $\mathsf{wskinf}_{ID^{(c)}} = (\alpha_{ID^{(c)}}, \beta'_{ID^{(c)}})$, with $B_{ID^{(c)}} = \beta'_{ID^{(c)}} B$ and the wallet secret key being $\mathsf{wsk}_{ID^{(c)}} = (\alpha_{ID^{(c)}}, \beta'_{ID^{(c)}} b)$. $\mathcal{B}$ generates the challenge derived verification key $\mathsf{dvk}^* = (Q_r^*, Q_{vk}^*)$ for wallet $ID^{(c)}$ as below:

1. $\mathcal{B}$ sets $Q_r^* = A$.

2. Note that $A = aP$ and $B_{ID^{(c)}} = \beta'_{ID^{(c)}} B$,
we have that Q^*_{vk} should be $Q^*_{vk} = \hat{e}(H_3(B_{ID^{(c)}}, A, \beta'_{ID^{(c)}} aB), -A_{ID^{(c)}})$ where a is unknown to $\mathcal{B}$.
$\mathcal{B}$ chooses a uniformly random $V^* \in \mathbb{G}^*_1$, and adds $(B_{ID^{(c)}}, A, \top, V^*)$ into L_{H_3}, implicitly setting $H_3(B_{ID^{(c)}}, A, \beta'_{ID^{(c)}} aB) = V^*$, where $\top$ denotes the value of $\beta'_{ID^{(c)}} aB$ that is unknown to $\mathcal{B}$.
$\mathcal{B}$ sets $Q^*_{vk} = \hat{e}(V^*, -A_{ID^{(c)}})$.
3. $\mathcal{B}$ sets $\mathsf{dvk}^* = (Q^*_r, Q^*_{vk})$, $\mathsf{dsk}^* = \alpha_{ID^{(c)}} \cdot V^*$, adds $(\mathsf{dvk}^*, ID^{(c)}, \mathsf{dsk}^*)$ into L_{dvk}, and returns dvk^* to $\mathcal{A}$.
Note that dsk^* is exactly the derived signing key corresponding to dvk^*.

■ **Probing Phase 2.**

– $H_0(\cdot)$: when H_0 is queried on input an identifier $ID \in \mathcal{S}_{ID}$:
Same as that of **Probing Phase 1**.
– $H_2(\cdot,\cdot)$: when H_2 is queried on input a pair $(input_1, input_2) \in \mathbb{G}_1 \times \mathbb{G}_1$: Same as that of **Probing Phase 1**.
– $H_3(\cdot,\cdot,\cdot)$: when H_3 is queried on input a tuple $(input_1, input_2, input_3) \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1$:
  - If $\underline{input_1 = B_{ID^{(k)}}\text{ AND } input_2 = A \text{ AND}}$ $\underline{\hat{e}(input_3, P) = \hat{e}(input_1, input_2) \text{ for } k = 0 \text{ or } 1}$ **(Below we denote this event by $\mathbf{E}_3$)**, which implies $input_3 = \beta'_{ID^{(k)}} bA = \beta'_{ID^{(k)}} abP$, $\mathcal{B}$ outputs $\frac{1}{\beta'_{ID^{(k)}}} input_3$ as its solution for the CDH problem and aborts the game.
  - Otherwise, $\mathcal{B}$ acts as the same way as in that of **Probing Phase 1**.
– $\mathsf{OWKeyDelegate}(\cdot)$:
Same as that of **Probing Phase 1**.
– $\mathsf{OWskCorrupt}(\cdot)$:
Same as that of **Probing Phase 1**.
– $\mathsf{OWsvkCorrupt}(\cdot)$:
Same as that of **Probing Phase 1**.
– $\mathsf{ODVKAdd}(\cdot,\cdot)$:
When $\mathcal{A}$ makes an $\mathsf{ODVKAdd}(\cdot,\cdot)$ query on input a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ and an entity's identifier $ID \in \mathcal{S}_{ID}$ at level $\underline{t \geq 0}$ such that $ID \in L_{wk}$:
  - If $ID \notin \{ID^{(0)}, ID^{(1)}\}$ OR $Q_r \neq Q^*_r$, $\mathcal{B}$ acts as the same way as in that of **Probing Phase 1**.
  - Otherwise (i.e., $ID \in \{ID^{(0)}, ID^{(1)}\}$ AND $Q_r = Q^*_r$), note that $\mathcal{A}$ should not query $\mathsf{ODVKAdd}(\cdot,\cdot)$ on input (dvk^*, ID) such that $ID \in \{ID^{(0)}, ID^{(1)}\}$, we only need to consider $Q_{vk} \neq Q^*_{vk}$. In particular, if $ID \in \{ID^{(0)}, ID^{(1)}\}$ AND $Q_r = Q^*_r$ AND $Q_{vk} \neq Q^*_{vk}$, $\mathcal{B}$ directly returns 0 to $\mathcal{A}$, since
    * If $ID = ID^{(c)}$, dvk is invalid for sure and hence should be rejected.
    * If $ID = ID^{(1-c)}$, note that there has been no corresponding random oracle query made to H_3 (due to the aborting event $\mathbf{E}_3$), $\mathcal{B}$ returns 0 to $\mathcal{A}$ as in **Probing Phase 1**.
– $\mathsf{ODSKCorrput}(\cdot)$:
When $\mathcal{A}$ makes an $\mathsf{ODSKCorrput}(\cdot)$ query on input a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ such that $\mathsf{dvk} \in L_{dvk}$:
$\mathcal{B}$ acts as the same way as in that of **Probing Phase 1**.
Note that even when $\mathcal{A}$ makes an $\mathsf{ODSKCorrput}(\cdot)$ query on input the challenge derived verification key dvk^*, $\mathcal{B}$ can return the corresponding derived signing key, say $\mathsf{dsk}^* = \alpha_{ID^{(c)}} \cdot V^*$.
– $\mathsf{OSign}(\cdot,\cdot)$:
When $\mathcal{A}$ makes an $\mathsf{OSign}(\cdot,\cdot)$ query on input a message $m \in \mathcal{M}$ and a derived verification key $\mathsf{dvk} = (Q_r, Q_{vk}) \in \mathbb{G}_1 \times \mathbb{G}_2$ such that $\mathsf{dvk} \in L_{dvk}$:
$\mathcal{B}$ acts as the same way as in that of **Probing Phase 1**.
Note that even when $\mathcal{A}$ makes an $\mathsf{OSign}(\cdot,\cdot)$ query on input the challenge derived verification key dvk^*, $\mathcal{B}$ can answer the query by running the signing algorithm using the corresponding derived signing key, say $\mathsf{dsk}^* = \alpha_{ID^{(c)}} \cdot V^*$.

■ **Guess.** $\mathcal{A}$ outputs a bit $c' \in \{0, 1\}$ as its guess to c.
Note that it implies neither event $\mathbf{E_2}$ nor event $\mathbf{E_3}$ happens in this case, $\mathcal{B}$ outputs $\perp$ and aborts the game (i.e., $\mathcal{B}$ fails to solve the CDH problem).

Probability Analysis. Let $\mathsf{Game}_{\mathcal{B}}$ denote the above game simulated by $\mathcal{B}$. If no **Abort** event (i.e., neither event $\mathbf{E}_2$ nor event $\mathbf{E}_3$) happens, the $\mathsf{Game}_{\mathcal{B}}$ is the same as the original $\mathsf{Game}_{\mathsf{swWUNL}}$ except that $\mathcal{B}$ returns 0 for an $\mathsf{ODVKAdd}(\cdot,\cdot)$ query if no corresponding H_3 tuple is found whereas in a real game, the challenger could return 1 with a probability $1/(p-1)$ for each such query made by $\mathcal{A}$. **Denote $\mathbf{E}_1$ the event that $\mathcal{B}$ answers one of the $\mathsf{ODVKAdd}(\cdot,\cdot)$ queries wrongly** and we have $\Pr[\mathbf{E}_1] \le q_{vka}/(p-1)$.
If neither **Abort** nor $\mathbf{E}_1$ happens, $\mathsf{Game}_{\mathcal{B}}$ is the same as $\mathsf{Game}_{\mathsf{swWUNL}}$, so we have

$$\begin{aligned}&(\frac{1}{2}+\epsilon_{\mathcal{A}}) - \Pr[\mathcal{A} \text{ wins in } \mathsf{Game}_{\mathcal{B}}]\\ \le\,&\Pr[\mathbf{Abort} \vee \mathbf{E}_1] \le \Pr[\mathbf{Abort}] + q_{vka}/(p-1)\end{aligned}$$

Since in $\mathsf{Game}_{\mathcal{B}}$, the hash value $V^* = H_3(B_{ID^{(c)}}, A, \beta'_{ID^{(c)}}aB)$ is chosen uniformly at random and unknown to $\mathcal{A}$, dvk^* and dsk^* leak no information about c. Hence, we have $\Pr[\mathcal{A} \text{ wins in } \mathsf{Game}_{\mathcal{B}}] = 1/2$. Therefore,

$$\Pr[\mathbf{Abort}] = \Pr[\mathbf{E}_2 \vee \mathbf{E}_3] \ge \epsilon_{\mathcal{A}} - \frac{q_{vka}}{p-1}$$

and $\mathcal{B}$ solves the CDH problem with the same non-negligible probability. Also, $\mathcal{B}$'s running time is bounded by $t + O(t^{(0)} + t^{(1)} + q_{H_0} + q_{H_2} + q_D + q_{vka} + q_S)\tau_{mul} + O(q_{H_2} + q_{H_3} + q_D + q_{vka} + q_S)\tau_p$.