

Singapore Management University

## Institutional Knowledge at Singapore Management University

---

Research Collection School Of Computing and Information Systems

School of Computing and Information Systems

---

9-2022

### Secure deterministic wallet and stealth address: Key-insulated and privacy-preserving signature scheme with publicly derived public key

Zhen LIU

Guomin YANG

Duncan S. WONG

Khoa NGUYEN

Huaxiong WANG

*See next page for additional authors*

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



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

---

#### Citation

LIU, Zhen; YANG, Guomin; WONG, Duncan S.; NGUYEN, Khoa; WANG, Huaxiong; KE, Xiaorong; and LIU, Yining. Secure deterministic wallet and stealth address: Key-insulated and privacy-preserving signature scheme with publicly derived public key. (2022). *IEEE Transactions on Dependable and Secure Computing*. 19, (5), 2934-2951.

Available at: [https://ink.library.smu.edu.sg/sis\\_research/7323](https://ink.library.smu.edu.sg/sis_research/7323)




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

---

**Author**

Zhen LIU, Guomin YANG, Duncan S. WONG, Khoa NGUYEN, Huaxiong WANG, Xiaorong KE, and Yining LIU

# Secure Deterministic Wallet and Stealth Address: Key-Insulated and Privacy-Preserving Signature Scheme With Publicly Derived Public Key

Zhen Liu , Guomin Yang , Duncan S. Wong, Khoa Nguyen, Huaxiong Wang , Xiaorong Ke, and Yining Liu

**Abstract**—Deterministic Wallet (DW) and Stealth Address (SA) mechanisms have been widely adopted in the cryptocurrency community, due to their virtues on functionality and privacy protection, which come from a key derivation mechanism that allows an arbitrary number of derived keys to be generated from a master key. However, these algorithms suffer a vulnerability that, when one derived key is compromised somehow, the damage is not limited to the leaked derived key only, but to the master key and in consequence all derived keys are compromised. In this article, we introduce and formalize a new signature variant, called Key-Insulated and Privacy-Preserving Signature Scheme with Publicly Derived Public Key (PDPKS), which fully captures and improves the functionality, security, and privacy requirements of DW and SA. We propose a PDPKS construction and prove its security and privacy in the random oracle model. Furthermore, we implement the construction with parameters for 128-bit security, and the results show that it is practically efficient for the setting of cryptocurrencies. With its solid guarantee on functionality, security and privacy, as well as its practical efficiency, our PDPKS construction provides a practical cryptographic tool that refines DW and SA, without security vulnerability.

**Index Terms**—Signature scheme, publicly derived public key, key-insulated security, privacy, cryptocurrency, stealth addresses, deterministic wallets

## 1 INTRODUCTION

SINCE the introduction of Bitcoin [2] in 2008, in the past decade, cryptocurrency has been undergoing a quick and explosive development, with thousands of different crypto-coins available to date, most of which are Bitcoin-like. A Bitcoin-like cryptocurrency is a ledger consisting of a series of transactions, and Digital Signatures [3] are used to authorize and authenticate the transactions. More specifically, each coin is a transaction-output represented by a (public key, value) pair, where the public key specifies the owner of the coin (i.e., the payee of the transaction) and the value specifies the denomination of the coin. When the owner wants to spend a coin  $cn$  on public key  $pk$ , acting as a payer, he needs to issue a new transaction consuming  $cn$

and outputting new coins (i.e., new transaction-outputs) assigned to the payees' public keys, and sign this new transaction using his secret signing key  $sk$  corresponding to  $pk$ . Due to the nature of digital signature, the public can be convinced that such a transaction is authorized and authenticated to spend the input coin  $cn$ . In other words, the public key acts as the *coin-receiving address*, while the secret signing key acts as *coin-spending key*. As a user may have multiple coins, a cryptocurrency *Wallet* is used to manage the (public key, secret key) pairs for the wallet owner, including generating, storing, using, and erasing the keys, etc. A preliminary wallet may generate each key pair randomly and independently by running the key generation algorithm of the underlying signature scheme in a typical manner. On the other hand, a type of advanced wallet mechanisms, called *Deterministic Wallets* [4], [5], [6], has been proposed to achieve some appealing virtues, such as low-maintenance, easy backup and recovery, supporting functionalities required by popular applications, and so on. Deterministic wallets have been very popular, and nearly every Bitcoin-like cryptocurrency client either already has a deterministic wallet implemented or is planning to create one. However, as shown later, the existing deterministic wallet algorithms may suffer a fatal vulnerability, which seriously limits their applications.

On the other side, while protecting user privacy is one of the most desired features of cryptocurrency, it has been generally acknowledged that Bitcoin only provides *pseudonymity*, which is pretty weak and does not provide *untraceability* or *unlinkability* [7], [8]. Different techniques and cryptocurrencies have been proposed to provide stronger privacy, for example,

- Zhen Liu is with the Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: liuzhen@sjtu.edu.cn.
- Guomin Yang is with the University of Wollongong, Wollongong, NSW 2522, Australia. E-mail: gyang@uow.edu.au.
- Duncan S. Wong is with CryptoBLK, Hong Kong. E-mail: duncanwong@cryptoblk.io.
- Khoa Nguyen and Huaxiong Wang are with the School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore 639798, Singapore. E-mail: {khoantt, HXWang}@ntu.edu.sg.
- Xiaorong Ke is with the Huawei Technologies Company, Ltd., Shenzhen 518129, China. E-mail: kexiaorong@sjtu.edu.cn.
- Yining Liu is with the University of Southern California, Los Angeles, CA 90007 USA. E-mail: ynyin24@sjtu.edu.cn.

Manuscript received 15 Oct. 2019; revised 28 Mar. 2021; accepted 1 May 2021.  
Date of publication 10 May 2021; date of current version 2 Sept. 2022.  
(Corresponding author: Zhen Liu.)  
Digital Object Identifier no. 10.1109/TDSC.2021.3078463

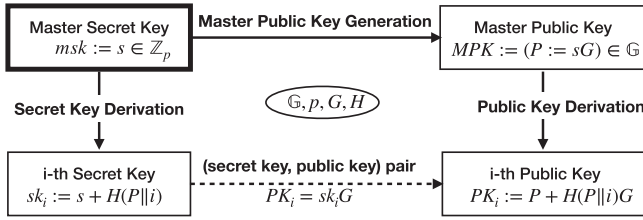


Fig. 1. Deterministic wallet algorithm.

Dash, ZCash, Monero, etc. Among the privacy protection technologies for cryptocurrencies, *Stealth Address* [8], [9] is regarded as a simple but effective and efficient way to enhance privacy, and has been widely adopted by many cryptocurrencies. Particularly, it is a part of the core protocol of Monero [10], which is the most popular privacy-centric cryptocurrency, with a market capitalization valued at approximate 2 billion USD, ranking the 10th in all the existing cryptocurrencies [11]. However, as shown later, the Stealth Address algorithm, the one used in Monero as example, also suffers a fatal vulnerability.

Below we show that the vulnerabilities lie in the inherent designs of the deterministic wallet and stealth address algorithms, in the sense that the damage of a minor fault will spread and incur destructive consequences. In this work, we formalize a new cryptographic concept and propose a provably secure construction, which provides a practical and solid solution to address these problems.

### 1.1 Deterministic Wallets for Bitcoin

Literally, in a deterministic wallet, all the public keys and secret keys can be deterministically derived from a ‘seed’. Fig. 1 shows the essence of the deterministic wallet algorithm. Actually, a specification of deterministic wallet based on this algorithm has been accepted as Bitcoin standard BIP32 [6], and other existing deterministic wallets, for example *Electrum* wallet [5], also use the similar algorithms. More specifically, let  $\mathbb{G}$  be an additive cyclic group of prime order  $p$ ,  $G \in \mathbb{G}$  be a generator of  $\mathbb{G}$ , and  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$  be a cryptographic hash function. A randomly and uniformly chosen  $s \in \mathbb{Z}_p$  works as the *master secret key* (i.e., the ‘seed’) for a deterministic wallet. Then the *master public key* is computed as  $P = sG$ , and for any index  $i$ , the  $i$ th public key could be derived from the master public key as  $PK_i = P + H(P||i)G$ , without needing to use the master secret key, while the corresponding  $i$ th secret key could be derived from the master secret key as  $sk_i = s + H(P||i)$ . Note that  $(sk_i, PK_i)$  satisfies  $PK_i = sk_i G$  and forms a valid (secret key, public key) pair for some Discrete Logarithm based signature schemes, for example, ECDSA (Elliptic Curve Digital Signature Algorithm) [12], which is used by most Bitcoin-like cryptocurrencies.

In the community, the deterministic wallets are advertised for the following use cases, as summarized in [13]:

- (1) *Low-maintenance wallets with easy backup and recovery.* With the algorithm in Fig. 1, to backup his deterministic wallet, a user only needs to backup the master secret key  $s$ , and when necessary, for example, the hardware where his deterministic wallet is stored breaks down, he can reconstruct the complete wallet from the master secret key.

- (2) *Freshly generated cold addresses.* In cryptocurrencies, cold address mechanisms are used to reduce the exposure chance of secret keys, namely, only the public keys are stored on a vulnerable online server (referred to as ‘hot storage’), while the corresponding secret keys are kept safe in offline storage (referred to as ‘cold storage’) until they are needed to generate signatures to spend the coins, and each public key (and corresponding secret key) is used only once to have better security and privacy. As each public key is used only once, cold address mechanisms protect user privacy in the sense that the public could not link the coins belonging to the same owner. The algorithm in Fig. 1 provides a *convenient* way to implement cold address mechanisms, namely, it allows a user to easily generate and store only the public keys on hot storage, while the corresponding secret keys are generated only when they are needed to spend the corresponding coins.

- (3) *Trustless audits.* The algorithm in Fig. 1 allows a user to reveal his master public key to third-party auditors, then the auditors could view all the transactions related to the corresponding wallet, since they can compute all the public keys in the wallet by using the master public key and the possible indexes. Note that the user is assured that his coins are safe from the theft by the auditor since the master secret key or derived secret keys could not be computed from the master public key and/or the derived public keys.

- (4) *Hierarchical Wallet allowing a treasurer to allocate funds to departments.* The algorithm in Fig. 1 allows a treasurer of a large company to create child key pairs for each department within the company, so that the treasurer will have the master public/secret key for everything, but each department will only have the key to their own part of the funds.

However, to use the deterministic wallet algorithm, the users have to be very aware in that, besides the master secret key, they also need to keep the master public key and *all* the derived secret keys secret and safe. This is because, if an attacker somehow obtains the master public key  $P$  and any derived secret key, say  $sk_i$  for some index  $i$ , he can compute the master secret key by  $s \leftarrow sk_i - H(P||i)$ , and further compromise the wallet completely. However, in practice these awareness requirements may be difficult to meet. In particular, to generate the derived public keys conveniently, the master public key is often stored in the vulnerable and online hot storage. For the derived secret keys, when they are used to sign a transaction, the signature computation is often performed on a relatively insecure device (e.g., a mobile device or an Internet-connected host) which cannot be trusted to maintain secrecy of the secret key, as pointed out by Dodis *et al.* [14], [15]. As a result, for the use case of *freshly generated cold addresses*, even if an attacker only obtains the master public key somehow, it could break the privacy-protection feature claimed by cold address mechanisms, by behaving like an auditor. And for the use case of *treasurer allocating funds to departments*, once a department manager who knows a child/derived secret key  $sk_i$  obtains the master public key somehow, he can steal all the funds of the company. In addition, *the deterministic wallet algorithm*

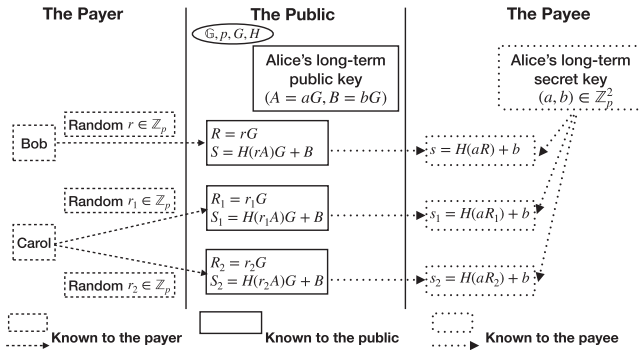


Fig. 2. Stealth address algorithm in monero.

cannot be used to simultaneously implement the treasurer and the auditor use cases, otherwise the auditor may collude with some department manager who knows a child/derived secret key  $sk_i$  to compromise the master secret key and then steal all the funds of the company.

## 1.2 Stealth Address in Monero

While deterministic wallets focus on the management of the keys in a wallet, the goal of stealth address is to send money to a certain *publicly visible* master key in such a way that this key does not appear in the ledger at all, so that users' privacy gets more protection. Thus, a crucial difference between stealth address and deterministic wallet is whether the master public key is allowed to be publicly visible. Note that the above advertised use cases of deterministic wallet heavily rely on the assumption that the master public key is kept secret.

As a privacy-centric cryptocurrency, to achieve unlinkable payments, Monero adopts a stealth address algorithm proposed in CryptoNote [8], as shown in Fig. 2. In particular, each user could choose random  $(a, b) \in \mathbb{Z}_p^2$  as his master secret key (also referred to as *long-term secret key*) and keep it secret, while publishing  $(A = aG, B = bG)$  publicly as his master public key (also referred to as *long-term public key*). *On the functionality*, for each transaction, the payer chooses a random  $r \in \mathbb{Z}_p$  and computes a derived public/verification key  ${}^1dvk = (R = rG, S = H(rA)G + B)$  from the payee's long-term public key  $(A, B)$ , and uses  $(R, S)$  as the coin-receiving address for the payee in the transaction. On the other side, from the view of a payee, with his long-term public key  $(A, B)$  and long-term secret key  $(a, b)$ , he can check whether he is the intended recipient of a coin on fresh public/verification key  $dvk = (R, S)$ , by checking  $S \stackrel{?}{=} H(aR)G + B$ , and if the equation holds, he can compute  $s = H(aR) + b$  as his secret/signing key to spend the coin, since  $(S, s)$  satisfies  $S = sG$  and forms a valid (public/verification key, secret/signing key) pair for a signature scheme. <sup>2</sup>*On*

1. Note that it is not required that the long-term public key and secret key forms a key pair for a signature scheme. To avoid confusion, we use (public/verification key, secret/signing key) to denote the key pair for signature scheme, where it is emphasized that verification key is public and signing key is secretly held.

2. Besides using the above stealth address algorithm to hide the payee, Monero hides the payer and transaction amount (i.e., the coin's value) using the techniques based on Linkable Ring Signature [16] and Pedersen Commitment [17] respectively. But all these functionalities are built on the basis of the above stealth address algorithm, as the derived key pair  $(S, s)$  serves as the coin-receiving address and coin-spending key.

*the privacy*, from the view of the public, the coin-receiving address  $(R, S)$  does not leak any information that can be linked to the payee's long-term public key. This is due to the Diffie-Hellman Key Exchange [18] part (i.e.,  $rA = aR = raG$ ), since the public cannot compute the value of  $raG$  from  $A$  and  $R$ . *On the security*, intuitively, for a coin-receiving address  $(R, S)$ , only the payee can derive the corresponding secret/signing key  $s = H(aR) + b$ , since only the payee knows the value of  $b$  for the corresponding long-term key. This is why  $B$  is added in  $S$ .

In summary, the main advantage of the stealth address algorithm is that every coin-receiving address is unique *by default* (unless the payer uses the same random data for each of his transactions to the same payee), so that there is no such issue as "address reuse" by design and no observer can determine if any transactions were sent to a specific long-term public key. And importantly, this is achieved in a very convenient manner, as each user only needs to publish one long-term public key, and anyone (acting as a payer) can derive an arbitrary number of fresh public/verification keys from the long-term public key of a user (acting as a payee), while there is no interaction needed between the payer and the payee. Also the payee can compute the secret/signing keys corresponding to the fresh public/verification keys without any interaction with the payer. Actually, due to its virtues in functionality, privacy and "security", the above algorithm and similar variants have been widely adopted by the cryptocurrency community to implement stealth addresses.

However, we would like to point out that, the above stealth address algorithm suffers a vulnerability which may cause fatal damages. In particular, consider the example in Fig. 2, namely, the payer Carol derives two public/verification keys  ${}^1dvk_1 = (R_1 = r_1G, S_1 = H(r_1A)G + B)$  and  ${}^2dvk_2 = (R_2 = r_2G, S_2 = H(r_2A)G + B)$  for the same payee Alice with long-term public key  $(A, B)$ . Suppose Carol somehow compromises one of the two secret/signing keys, say  $s_1 = H(aR_1) + b$ . Note that Carol knows the value of  $r_1$ , she can compute the value of  $b$  by  $b \leftarrow s_1 - H(r_1A)$ . So, Carol can compute the secret/signing key corresponding to  ${}^2dvk_2$ , by  $s_2 \leftarrow H(r_2A) + b$ , since she also knows the value of  $r_2$ . Furthermore, if Carol colludes with other payers who sent coins to Alice, they can compromise all the secret/signing keys for the related coins, for example, colluding with Bob in Fig. 2, Carol and Bob can compute the secret/signing key corresponding to  $(R, S)$  by  $s \leftarrow H(rA) + b$ , where  $r$  is provided by Bob. Actually, as long as *one* derived secret/signing key is compromised, the corresponding long-term secret key is not safe any more, and all coins to the fresh public/verification keys derived from the corresponding long-term public key in the past and the future are in danger of being stolen.

As a result, the users in Monero must be very aware in that, they not only need to keep their long-term secret keys safe, but also need to keep *all* the derived secret/signing keys for their coins absolutely safe, even after the coins have been spent, since leaking *one* derived secret/signing key may lead to the complete leakage of *all* the secret/signing keys derived from the same long-term key. Note that the users in Monero are not warned about this vulnerability, and the security heavily depends on the implementations



and the users' awareness and behaviors. However, in practice keeping all the derived secret/signing keys (i.e., co-spenders) safe is a difficult task, as pointed out by Dodis *et al.* [14], [15], "*cryptographic computations (decryption, signature generation, etc.) are often performed on a relatively insecure device (e.g., a mobile device or an Internet-connected host) which cannot be trusted to maintain secrecy of the private key.*" In addition, it is worth mentioning that, while the deterministic wallet is only an optional tool for Bitcoin, the stealth address algorithm is a part of the core protocol for Monero. That is, the damages caused by the vulnerability seem to be inevitable for Monero, unless it is fixed.

### 1.3 Related Work

The community has noticed the deterministic wallet algorithm's vulnerability that once the master public key and one secret key are compromised, the master secret key and the whole wallet will be compromised. In particular, the authors of BIP32 standard [6] noticed this vulnerability and compensated for it by allowing for "hardened" child secret key that can be compromised without also compromising the master secret key. But the cost is that the public keys cannot be generated from the master public key, i.e., it cannot support the use cases of 'freshly generated cold addresses' or 'trustless audits'. Buterin [19] called attention to this vulnerability, by announcing open-source software that cracks BIP32 [6] and Electrum wallets [5], but was pessimistic on fixing this vulnerability. As an attempt to fix this vulnerability, Gutoski and Stebila [13] proposed a deterministic wallet that can tolerate the leakage of up to  $m$  derived secret keys with a master public key size of  $O(m)$ . In essence, Gutoski and Stebila's algorithm improves the difficulty of compromising the master secret key  $m$  times at the price of  $O(m)$  times larger master public key, but does not eliminate this vulnerability, in the sense that if an attacker compromises the master public key and  $m$  secret keys, it can compromise the master secret key. Thus, the algorithm by Gutoski and Stebila [13] still heavily relies on the users' awareness and suffers the problems discussed in Section 1.1. For example, when being used to simultaneously implement the treasurer and the auditor use cases, the parameter  $m$  must be larger than the possible maximum number of the departments.

For the vulnerability in Monero's stealth address algorithm, it is somewhat surprising that it has not been noticed in the community. This might be because that the algorithm allows the master public key to be publicly visible, and that it seems that the value of  $b$  could not be computed from the master public key  $(A, B)$ , one secret key  $s = H(aR) + b$ , and the corresponding public key  $(R, S)$ . It is worth mentioning that Courtois and Mercer [20] pointed out that, when the stealth address algorithm works together with ECDSA, if the signature scheme is implemented incorrectly so that two ECDSA signatures use the same randomness, the two payers who generated the two public keys corresponding to the two 'bad' signatures may collude and compromise the master secret key. To address this problem, they proposed an enhanced stealth address algorithm by incorporating Gutoski and Stebila's method [13] and gets similar results as that of [13], i.e., their algorithm improves the difficulty of compromising the master secret key  $m$  times at the price of

$O(m)$  times larger master public key, but does not eliminate the problem.

### 1.4 Our Results

Note that for the vulnerabilities of the deterministic wallet and stealth address algorithms, as well as the problem pointed out by Courtois and Mercer [20], the essence is that the derived secret keys are not insulated from each other, neither from the master secret key, so that one derived secret key being compromised will lead to the compromising of the master secret key and all other derived secret keys. From a cryptography security point of view, this is a fatal flaw, as when a minor fault happens (say, one derived secret key is compromised somehow), the damage spreads and the whole system collapses. As a counterexample, for an old-style wallet where each key pair is generated independently by running the key generation algorithm of the signature scheme, if a secret key is compromised, only the coins on the corresponding public key may be stolen, without affecting the security of other keys or coins.

Naturally, we want to enjoy both *the virtues* of deterministic wallet and stealth address, including the functionalities and privacy-protection features, and *the security* of conventional wallet and address mechanism. Intuitively, this may be achieved by a cryptographic primitive where the security model allows the derived secret key to be corrupted by the adversaries while the security and privacy rely only on the secrecy of the master secret key, and the damage of derived secret key being compromised will not spread at all.

In this work, we introduce and formalize a new signature variant, called Key-Insulated and Privacy-Preserving Signature Scheme with Publicly Derived Public Key (PDPKS),<sup>3</sup> and propose a provably secure and practically efficient construction, which provides a solution that offers the virtues of deterministic wallet and stealth address, while eliminating the security vulnerabilities completely.

More specifically, the contributions of this work consists of the following:

- (1) We formalize the concept of PDPKS, namely, a cryptographic primitive that inherently supports deterministic wallets and stealth addresses. PDPKS captures the functionality, security, and privacy requirements that a deterministic wallet and/or stealth address algorithm should satisfy, and the existing and future deterministic wallet and/or stealth address algorithms can be investigated and designed under the definitions and models of this concept. In particular,
  - On the functionality, each user publishes his master public key, and anyone can derive an arbitrary number of public/verification keys from a master public key without requiring any interaction, while only the owner of the master public key can generate the corresponding secret/signing keys, also without interactions. Each user's master secret key consists of *master*

3. We abbreviate this signature variant to PKPDS to emphasize its functionality feature, namely, Signature Scheme with Publicly Derived Public Key.

TABLE 1  
Efficiency of Our PDPKS Construction With 128-Bit Security: Time

Signing Time	Verification Time	Setup Time	Master Key Generation Time	Verification Key Derivation Time	Verification Key Check Time	Signing Key Derivation Time
4.323 ms	7.298 ms	9.008 ms	1.951 ms	2.884 ms	1.926 ms	3.315 ms

<sup>1</sup>The average time is obtained by running our implementation on a usual computation environment, i.e., a virtual machine with Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz, 1GB Memory, and the operating system ubuntu 16.04 LTS.

secret view key and master secret spend key, where the master secret view key is sufficient to check whether a verification key is derived from the corresponding master public key and the whole master secret key is needed to generate the signing key for a derived verification key.

These enable the functionalities of deterministic wallet, including key derivation from a seed (master key), freshly generated cold addresses, trustless audits, and treasurers, and also enable the functionalities of stealth address, including that the verification key can be derived by anyone (rather than only by the key owner) and no interactions are needed. Note that while the master secret spend key is used to guarantee the safety of the coins, the master secret view key can be used to control the granularity of privacy, such as that in trustless audits.

- The security relies on the secrecy of the master spend key only. Corrupting the master secret view key and the signing keys for other derived verification keys will not enable an attacker to steal the coins on a target derived verification key.

This security guarantee enables the users to use PDPKS to implement deterministic wallets and stealth addresses, with strong confidence on the safety of their coins. For example, PDPKS allows the users to simultaneously implement the treasurer and the auditor use cases, without security concerns.

- On the privacy, the derived verification keys and corresponding signatures do not leak any information that can be linked to the origin master public key or other derived verification keys.

This captures the privacy requirement of stealth address, in the sense that not only the coin-receiving addresses (i.e., the verification keys) but also the spending transactions (which contain the signatures with respect to the verification keys) leak no information about the origin master public keys or other verification keys. It is worth mentioning that the existing stealth addresses only informally discussed that the derived verification keys seems to be unlinkable to the master public key.

- (2) We not only propose and formalize the requirements (by formal definitions and models) as above, but also propose a practical PDPKS construction that satisfies the requirements.

- We propose a (partially) generic approach on how to obtain a PDPKS construction that satisfies the functionality, security, and privacy requirements.
- We propose a concrete PDPKS construction and formally prove its security and privacy in the random oracle model.
- We implement the proposed construction with parameters providing 128-bit security. As shown in Tables 1 and 2, the signature size, the keys' sizes, the signing time, and the verification time are practical for the settings of cryptocurrencies.

In summary, with its functionality, security, and privacy protection features, as well as the practical efficiency, our PDPKS construction provides a practical cryptographic tool that inherently supports the use cases of deterministic wallet and stealth address, without security vulnerabilities.

Remark: In the preliminary version [1] of this work, we introduced the concept of PDPKS, including its algorithms, its security model, and its privacy model for master public key unlinkability, proposed a PDPKS construction, and proved its security and privacy. In this extended version, to have a practical cryptographic tool for deterministic wallets and stealth addresses, (1) we extend the algorithm definitions to support the separation of master secret view key and master secret spend key, and extend the security model accordingly; (2) we add a privacy model to capture the derived verification key unlinkability; (3) we modify the construction in the preliminary version to satisfy the extended definitions, and show that its security and privacy can be proven in the extended and new models; and (4) we implement the construction with parameters for 128-bit security, and present the concrete efficiency results. In addition, (5) we propose another PDPKS construction as another instance of our (partially) generic constructing approach.

## 1.5 Outline

In Section 2, we review the techniques related to privacy protection and key insulation, and present our approach to construct a PDPKS scheme. In Section 3, we formalize the

TABLE 2  
Efficiency of Our PDPKS Construction With 128-Bit Security: Sizes of Signature and Keys

Signature Size	Verification Key Size	Signing Key Size	Master Public Key Size	Master Secret View Key Size	Master Secret Spend Key Size
116 Byte	232 Byte	58 Byte	232 Byte	58 Byte	58 Byte

definition, the security model, and the privacy models for PDPKS. In Section 4 we propose a PDPKS construction, and prove its security and privacy in Section 5. In Sections 6 and 7 we discuss the applications and present our implementation of the proposed PDPKS construction. In Section 8, we give another PDPKS construction as another instance of our construction approach. The paper is concluded in Section 9.

## 2 RELATED TECHNIQUES AND OUR APPROACH

### 2.1 Techniques Related to Privacy Protection

While Blind Signature [21] hides the really signed messages from the signer and Group Signature [22] and Ring Signature [23] hide the identity of the real signer from the public, the PDPKS signature in this work focuses on (public) key privacy, i.e., breaking the link between the derived public/verification keys (and corresponding signatures) and the underlying long-term public key. From the view point of motivations in practice, namely in cryptocurrency, this is to protect the privacy of the payees of the transactions, as the derived public/verification keys are used to specify the owner of the output coins. Note that in Monero [10], a variant of ring signature, namely Linkable Ring Signature [16], has been adopted to hide the payer, while the above discussed stealth address algorithm is used to hide the payee.

While the privacy protection concerns in cryptocurrencies motivate us to investigate the (public) key privacy problem for digital signature in this paper, Bellare *et al.* [24] has considered a similar problem in the setting of public key encryption in 2001, where *key privacy* requires that an eavesdropper in possession of a ciphertext cannot tell which specific key, out of a set of known public keys, is the one under which the ciphertext was created, meaning the recipient is anonymous from the view point of the adversary.<sup>4</sup>

Recently, a new notion named “Signatures with Flexible Public Key” was proposed in [25]. It allows a signer of a digital signature scheme to derive new public and private key pairs that fall in the same “equivalent class”. This new primitive has various applications including stealth addresses for cryptocurrencies. However, as the primitive does not consider the attack that an adversary corrupts the derived keys, when it is used to implement stealth addresses, it suffers the same security issue as in the stealth address algorithm for Monero illustrated above.

### 2.2 Techniques Related to Key Insulation

Motivated by the fact that in practice signature computation is often performed on a relatively insecure device (e.g., a mobile device or an Internet-connected host) which cannot be trusted to maintain secrecy of the secret key, Dodis *et al.* [14], [15] introduced key-insulated signature scheme, where the lifetime of the protocol is divided into  $N$  distinct periods, and at the beginning of each period a temporary secret key is derived and will be used by the insecure device to sign messages during that period. The security of key-insulated signature scheme means that even if an adversary corrupts  $t$  temporary secret keys, it will be unable to forge a

signature on a new message for any of the remaining  $N - t$  periods. Note that the key-insulated signature scheme does not consider the privacy-protection problem, and it is not applicable to the setting of cryptocurrency, where the verification key (serving as coin-receiving address) and signing key (serving as coin-spending key) are unrelated to time periods.

In Identity-based Cryptography (IBC) [26], [27], there is an entity referred to as Private Key Generator (PKG), who publishes the system master public key MPK and holds the system master secret key MSK. For any identity string ID, PKG can generate a corresponding user secret key  $sk_{ID}$ , which can be used to decrypt ciphertexts encrypted under (MPK, ID) as public key (in Identity-based Encryption (IBE) system) or sign a message to produce a signature that can be verified by (MPK, ID) as verification key (in Identity-based Signature (IBS) system). In a secure IBC system, unbounded leakage of user secret keys will not affect the security of the master secret key or other identities’ user secret keys. In other words, the user secret keys in IBC are insulated from each other. On privacy, user/identity anonymity *inside one instantiation* has been studied much, known as anonymous IBE [28], where the ciphertext hides the identity used to create it. And the IBE by Agrawal *et al.* [29] considered the *master public key privacy among multiple instantiations*, where the ciphertext hides the master public key used to create it. However, the master public key privacy for IBS may be more complicated than that in IBE, since the master public key and the identity need to be known by the public who verify the signature. In addition, IBS with master public key privacy seems to lack motivations in practice. As a result, IBS with master public key privacy has not been considered as so far.

### 2.3 Our Construction Approach

Besides introducing and formalizing PDPKS, including its definition and models for security and privacy, we also propose a concrete construction with provable security and privacy. Below we briefly describe our construction approach.

Note that what we need is a signature scheme where (1) each public/verification key can be derived from a (long-term, unchanged) master public key, and the corresponding secret/signing key can be computed from the verification key and the long-term master secret key; (2) the (verification key, signing key) pairs are insulated from each other, namely one being compromised will not affect others; and (3) the verification keys, as well as the signatures, could not be linked to the original long-term master public key. For the requirements (1) and (2), it is natural to consider the Identity-Based Signature (IBS) [26], [27], [30], which supports verification key derivation and can tolerate unbounded leakage of the user secret/signing keys. The challenge is how to achieve the privacy described by requirement (3).

To construct a PDPKS, we start from an IBS scheme as below:

- Each user, say  $U_i$ , runs an instantiation of the IBS scheme and acts as the PKG for the instantiation, namely, publishes the system master public key of IBS as his long-term master public key of PDPKS,

4. We borrow the term “key privacy” from [24], although its meaning for digital signature in this paper is very different from that for public key encryption in [24].



and holds the master secret key as his long-term master secret key, denoted by  $MPK_i$  and  $MSK_i$ , respectively.

- When issuing a transaction with  $U_i$  as the payee, the payer creates a random string (i.e., identity)  $ID$  and sets  $vk = (MPK_i, ID)$  as the fresh public/verification key for the output coin. Note that  $MPK_i$  being included in  $vk$  is to ensure that only the intended payee (i.e., the owner of  $MPK_i$ ) can generate the corresponding secret/signing key  $sk_{vk}$ .
- For any coin with a fresh public/verification key, say  $vk = (MPK_i, ID)$ , the intended payee can run the IBS' Key Extract algorithm  $sk_{vk} \leftarrow \text{IBS.KeyExtract}(MPK_i, ID, MSK_i)$  and set  $sk_{vk}$  as the secret/signing key, and then spend the coin by generating a valid signature  $\sigma$ , which can be verified by the IBS' Verify algorithm  $\text{IBS.Verify}(MPK_i, ID, M, \sigma)$ , where  $M$  is the signed message.

Note that using IBS in such a way does not suffer the drawbacks of the key-escrow problem in IBS,<sup>5</sup> since each user acts as PKG for the identities for himself, and actually is making use of the key-escrow functionality. Such an intuitive construction seems to address the requirements (1) and (2), but does not provide privacy at all, as  $vk = (MPK_i, ID)$  contains the corresponding long-term master public key  $MPK_i$ . To provide privacy required by PDPKS, the verification algorithm should take only the verification key  $vk$ , the message, and the signature  $\sigma$  as inputs, and  $vk$  and  $\sigma$  should not leak any information about the corresponding MPK. As discussed previously in Section 2.2, IBS with such a master public key privacy property has not been considered or researched as so far. In this work, motivated by the vulnerabilities of the stealth address algorithm for Monero and the deterministic wallet algorithm for Bitcoin, we focus on the formalization and construction of PDPKS, rather than IBS with master public key privacy, which lacks motivations in practice. To construct a PDPKS from IBS using the above approach, we need the IBS scheme to have the following property (referred to as MPK-pack-able Property):

- The master public key MPK of the IBS scheme can be divided into two parts CMPK and IMPK, where CMPK are the common parameters shared by all the instantiations of the IBS scheme, for example, the underlying groups, while IMPK are the particular parameters for each individual instantiation, for example, the public parameters generated from the master secret keys of the instantiations.
- There is a function  $F$  and a verification algorithm  $\text{Verify}_F$  such that
  - (1) An attacker, who does not know the value of  $ID$ , cannot learn any partial information about IMPK from the value of  $F(MPK, ID)$ , where  $ID$  is a random string.
  - (2) The signature does not leak any partial information about IMPK.

- (3) For any master public key MPK, any random  $ID$ , any message  $M$ , and any signature  $\sigma$ , it holds that  $\text{Verify}_F(\text{CMPK}, F(\text{MPK}, ID), M, \sigma) = \text{IBS.Verify}(\text{MPK}, ID, M, \sigma)$ .

Intuitively, with such an IBS scheme, we can generate  $ID$  using the non-interactive Diffie-Hellman Key Exchange mechanism (as shown in Fig. 2) to prevent the attacker from knowing the value of  $ID$ , and set  $vk = (R, F(\text{MPK}, ID))$  where  $R = rG$  is the randomness chosen by the payer for the Diffie-Hellman Key Exchange mechanism, so that we can achieve the privacy requirement of PDPKS. Note that the ideas behind the above requirements are that the verification key should be derived from MPK and  $ID$ , but leak no information about IMPK, and we use the function  $F$  to perform this derivation operation. In addition, the value of the function  $F$  should be independent from the message and signature, that is why  $F$  takes only MPK and  $ID$  as inputs.

In this work, to obtain a PDPKS construction by the above approach, we investigated three existing IBS schemes [31], [32], [33], which have very different construction structures. Finally, we find that the IBS schemes in [31], [33] have the above MPK-pack-able property, while the IBS scheme in [32] does not have. We also investigated the three generic transformations [34], which transform standard signature schemes, convertible identification schemes, and hierarchical identity based encryption schemes to IBS schemes, as well as the presented IBS instantiations in [34]. Also, we found that none of the resulting generic IBS constructions or the concrete IBS instantiations in [34] has the above MPK-pack-able property. This is not surprising, as the master public key privacy has not been considered in IBS. Based on the IBS schemes in [33], we construct a PDPKS scheme formally and prove its security and privacy in the random oracle model. Note that even given an IBS scheme with the MPK-pack-able property, it is not trivial to obtain a secure PDPKS construction, especially the proofs. Roughly speaking, on the construction, inspired by the stealth address algorithm in Monero, we generate the identity using the non-interactive Diffie-Hellman Key Exchange mechanism (as shown in Fig. 2). On the proof, implied by the above approach, the security proofs are comparatively easy, by a reduction to the security of the underlying IBS scheme, while the privacy proofs need more efforts. More specifically, our techniques include using parallel/double public keys (one for proving security and one for proving privacy) and using  $H(rG, (ra)G)$  rather than  $H((ra)G)$  as in the stealth address algorithm for Monero. All these techniques are to enable the proof of privacy.

To further demonstrate our approach, in the preliminary version [1, Appendix A] we showed the details that the IBS scheme in [32] does not have the MPK-pack-able property. In this extended version, in Section 8 we show that the IBS scheme in [31] has the MPK-pack-able property, and we can obtain a PDPKS construction and the corresponding proof using the techniques similar to that of our construction in Section 4.

We would like to point out that the above approach of transforming an IBS scheme to a PDPKS scheme is not the unique way to construct PDPKS schemes. Although the ideas and techniques in IBS could be useful tools for constructing PDPKS, we do not want to limit the construction

5. The key-escrow problem in IBS is that the PKG can generate and know the secret key  $sk_{ID}$  for any identity  $ID$ , which is regarded as the main drawback of IBS.

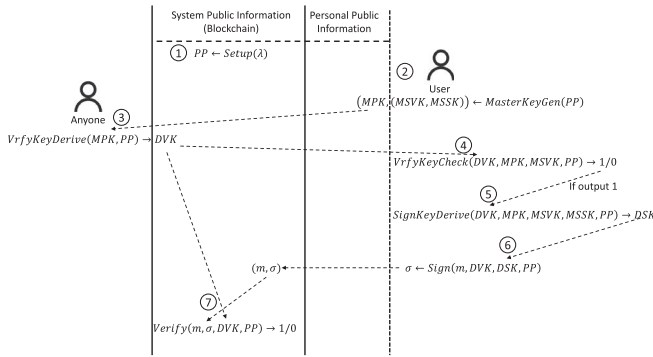


Fig. 3. System model.

of PDPKS to being from IBS only and would like to be open to other approaches. That is why we formalize the concept of PDPKS, rather than extending the IBS concept.

### 3 KEY-INSULATED AND PRIVACY-PRESERVING SIGNATURE SCHEME WITH PUBLICLY DERIVED PUBLIC KEY

In this section, we formalize the notion of Key-Insulated and Privacy-Preserving Signature Scheme with Publicly Derived Public Key (PDPKS).

In Section 3.1, we formalize the comprising algorithms which capture the functionalities of deterministic wallets and stealth addresses. Roughly speaking, as shown in Fig. 3, (1) the  $\text{Setup}()$  algorithm is run to generate the system public parameters; (2) each user can run the  $\text{MasterKeyGen}()$  algorithm to generate his master public key  $\text{MPK}$  and master secret key  $\text{MSK}$  (consisting of master secret view key  $\text{MSVK}$  and master secret spend key  $\text{MSSK}$ ), and publish his master public key; (3) Any user can run the  $\text{VrfyKeyDerive}()$  algorithm on input a target  $\text{MPK}$  and obtain a freshly generated unique derived verification key  $\text{DVK}$ , and this captures the key derivation functionalities of deterministic wallets and stealth addresses; (4) For a user with master public key  $\text{MPK}$ , given a derived verification key  $\text{DVK}$ , he can run the  $\text{VrfyKeyCheck}()$  algorithm with his  $\text{MSVK}$  to check whether  $\text{DVK}$  is derived from his  $\text{MPK}$ , and this captures that a user can check whether a coin with coin-receiving address  $\text{DVK}$  on the blockchain belongs to him; (5) For a user with master public key  $\text{MPK}$ , given a derived verification key  $\text{DVK}$  that was derived from his  $\text{MPK}$ , he can run the  $\text{SignKeyDerive}()$  algorithm with his master secret key ( $\text{MSVK}, \text{MSSK}$ ) to generate the signing key  $\text{DSK}$  corresponding to  $\text{DVK}$ , and (6) further run the  $\text{Sign}()$  algorithm with  $\text{DVK}$  and  $\text{DSK}$  to generate a signature, which can be used to authorize and authenticate a transaction spending the coins on  $\text{DVK}$ ; (7) Any one can run the  $\text{Verify}()$  algorithm with a  $\text{DVK}$  to check the validity of any given message-signature pair  $(m, \sigma)$ , and this captures that the master public keys do not need to appear on the blockchain, which is a basic functionality feature of stealth addresses.

In Section 3.2, we define the security models of PDPKS to capture the security requirements on deterministic wallets and stealth addresses, namely, the security should rely on the secrecy of the master spend key only. That means, for a target derived verification key, even if it was generated by the attacker from a target master public key, and even if the

attacker corrupts the master secret view key and an arbitrary number of signing keys for other derived verification keys, the attacker is not able to forge a valid signature to steal the coins on the target derived verification key.

In Section 3.3, we define the privacy models of PDPKS to capture the privacy-protection requirements that the derived verification keys and corresponding signatures do not leak any information that can be linked to the origin master public key or other derived verification keys.

#### 3.1 Algorithm Definition

A PDPKS scheme consists of the following polynomial-time algorithms:

- $\text{Setup}(\lambda) \rightarrow \text{PP}$ . This is a probabilistic algorithm. On input a security parameter  $\lambda$ , the algorithm runs in polynomial time in  $\lambda$ , and outputs system public parameters  $\text{PP}$ .

*The system public parameters  $\text{PP}$  are common parameters used by all users in the system, including the underlying groups, hash functions, etc.*

- $\text{MasterKeyGen}(\text{PP}) \rightarrow (\text{MPK}, \text{MSK})$ . This is a probabilistic algorithm. On input the system public parameters  $\text{PP}$ , the algorithm outputs a (master public key, master secret key) pair  $(\text{MPK}, \text{MSK})$ , where  $\text{MSK} := (\text{MSVK}, \text{MSSK})$  consists two parts, namely, master secret view key  $\text{MSVK}$  and master secret spend key  $\text{MSSK}$ .

*Each user runs  $\text{MasterKeyGen}$  algorithm to generate his long-term (master public key, master secret key) pair.*

- $\text{VrfyKeyDerive}(\text{MPK}, \text{PP}) \rightarrow \text{DVK}$ . This is a probabilistic algorithm. On input a master public key  $\text{MPK}$  and the system public parameters  $\text{PP}$ , the algorithm outputs a derived verification key  $\text{DVK}$ .<sup>6</sup>

*Anyone can run this algorithm to generate a fresh public/verification key from a master public key.*

- $\text{VrfyKeyCheck}(\text{DVK}, \text{MPK}, \text{MSVK}, \text{PP}) \rightarrow 1/0$ . This is a deterministic algorithm. On input a derived verification key  $\text{DVK}$ , a master public key  $\text{MPK}$  and corresponding master secret view key  $\text{MSVK}$ , and the system public parameters  $\text{PP}$ , the algorithm outputs a bit  $b \in \{0, 1\}$ , with  $b = 1$  meaning that  $\text{DVK}$  is a valid derived verification key generated from  $\text{MPK}$  and  $b = 0$  otherwise.

*With the master secret view key  $\text{MSVK}$  corresponding to a master public key  $\text{MPK}$ , a user can use this algorithm to check whether a derived verification key is derived from this  $\text{MPK}$ .*

*When applied in cryptocurrency, this algorithm enables a user (the owner of a master key) to check whether he is the intended recipient of a coin on the verification key. Also, this algorithm enables a user to reveal his master secret view key to the auditors, to support trustless-audit.*

- $\text{SignKeyDerive}(\text{DVK}, \text{MPK}, \text{MSVK}, \text{MSSK}, \text{PP}) \rightarrow \text{DSK}$  or  $\perp$ . On input a derived verification key  $\text{DVK}$ , a (master public key, master secret key) pair  $(\text{MPK}, (\text{MSVK}, \text{MSSK}))$ , and the system public

6. From now on, due to the clear definition, we use 'verification key' and 'signing key', rather than 'public/verification key' and 'secret/signing key', respectively.

parameters  $PP$ , the algorithm outputs a derived signing key  $DSK$ , or  $\perp$  implying that  $DVK$  is not a valid verification key derived from  $MPK$ .

The owner of a master key can use this algorithm to compute the signing key corresponding to a given derived verification key, if the verification key was indeed derived from this master public key.

- $\text{Sign}(m, DVK, DSK, PP) \rightarrow \sigma$ . On input a message  $m$  in the message space  $\mathcal{M}$ , a derived (verification key, signing key) pair  $(DVK, DSK)$ , and the system public parameters  $PP$ , the algorithm outputs a signature  $\sigma$ .
- $\text{Verify}(m, \sigma, DVK, PP) \rightarrow 1/0$ . This is a deterministic algorithm. On input a (message, signature) pair  $(m, \sigma)$ , a derived verification key  $DVK$ , and the system public parameters  $PP$ , the algorithm outputs a bit  $b \in \{0, 1\}$ , with  $b = 1$  meaning valid and  $b = 0$  meaning invalid.

**Correctness.** The scheme must satisfy the following correctness property: For any message  $m \in \mathcal{M}$ , suppose

$$\begin{aligned} PP &\leftarrow \text{Setup}(\lambda), \\ (MPK, (MSVK, MSSK)) &\leftarrow \text{MasterKeyGen}(PP), \\ DVK &\leftarrow \text{VrfyKeyDerive}(MPK, PP), \\ DSK &\leftarrow \text{SignKeyDerive}(DVK, MPK, (MSVK, MSSK), PP), \end{aligned}$$

it holds that

$$\begin{aligned} \text{VrfyKeyCheck}(DVK, MPK, MSVK, PP) &= 1 \text{ and} \\ \text{Verify}(m, \text{Sign}(m, DVK, DSK, PP), DVK, PP) &= 1. \end{aligned}$$

*Remark:*(1) Note that the above definition is open on whether the  $\text{SignKeyDerive}$  and  $\text{Sign}$  algorithms are probabilistic or deterministic, which may depend on the concrete constructions. (2) Separating the master secret key into two parts, say master secret view key and master secret spend key, is to capture the practical scenarios that in cryptocurrencies, the master secret view key is used much more frequently than the master secret spend key,<sup>7</sup> and as a result, the master secret view key faces much higher risk of being compromised than the master secret spend key. The separation enables the key owner to keep the master secret spend key as safe as possible, rather than unnecessarily facing the same high risk as the master secret view key. In addition, this separation and the corresponding security model also enable PDPKS to be used to implement Deterministic Wallet, supporting the trustless audits use case, even simultaneously with the treasurer use case, by offering the master secret view key to the auditors, without needing to worry the coins are stolen by the auditors.<sup>8</sup>

7. In the setting of stealth addresses, from the view point of a key owner, for each received new transaction on the blockchain, he needs to use the master secret view key to check whether he is the target recipient of the coins represented by the transaction outputs, whereas only when he wants to spend a coin owned by him, he needs his master secret spend key to generate a secret signing key and spend that coin.

8. While a similar separation was introduced in stealth address algorithms of [8], [10], in this work we formalize this separation and more importantly we formalize the corresponding security models, which formally capture that even if an adversary compromises the master secret view key, he is not able to generate the secret/signing keys for the derived public/verification keys or steal the corresponding coins.

### 3.2 Security Model

As PDPKS is actually a signature scheme, the security requirement is that, for a given derived verification key, as long as the corresponding signing key and the master secret spend key are safe (i.e., not compromised by the adversaries), no adversary can forge a valid signature with respect to this derived verification key, even if the target verification key was generated by the adversary from a target master public key, and the adversary compromises the master secret view key and an arbitrary number of derived signing keys for other verification keys. The following security model captures this security requirement.

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

- **Setup Phase.**  $\text{Setup.PP} \leftarrow \text{Setup}(\lambda)$  is run and  $PP$  are given to  $\mathcal{A}$ .  $(MPK, (MSVK, MSSK)) \leftarrow \text{MasterKeyGen}(PP)$  is run and  $MPK$  is given to  $\mathcal{A}$ . An empty set  $L_{dvk} = \emptyset$  is initialized.<sup>9</sup>
- **Probing Phase.**  $\mathcal{A}$  can adaptively query the following oracles:
  - **Verification Key Adding Oracle  $\text{ODVKAdd}(\cdot)$ :**  
On input a derived verification key  $DVK$ , this oracle returns  $b \leftarrow \text{VrfyKeyCheck}(DVK, MPK, MSVK, PP)$  to  $\mathcal{A}$ . If  $b = 1$ , set  $L_{dvk} = L_{dvk} \cup \{DVK\}$ .  
This captures that  $\mathcal{A}$  can try and test whether the derived verification keys generated by him are accepted by the owner of  $MPK$ .
  - **Signing Key Corruption Oracle  $\text{ODSKCorrupt}(\cdot)$ :**  
On input a derived verification key  $DVK$  in  $L_{dvk}$ , this oracle returns  $DSK \leftarrow \text{SignKeyDerive}(DVK, MPK, MSVK, MSSK, PP)$  to  $\mathcal{A}$ .  
This captures that  $\mathcal{A}$  can obtain the derived signing keys for some existing valid derived verification keys of its choice.
  - **Signing Oracle  $\text{OSign}(\cdot, \cdot)$ :**  
On input a message  $m \in \mathcal{M}$  and a derived verification key  $DVK \in L_{dvk}$ , this oracle returns  $\sigma \leftarrow \text{Sign}(m, DVK, DSK, PP)$  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.
  - **Master Secret View Key Corruption Oracle  $\text{OMSVKCorrupt}(\cdot)$ :**  
This oracle returns  $MSVK$  to  $\mathcal{A}$ .  
This captures that  $\mathcal{A}$  can obtain the master secret view key somehow. For example, the key owner may reveal his master secret view key to an auditor, but the auditor may become malicious.
- **Output Phase.**  $\mathcal{A}$  outputs a message  $m^* \in \mathcal{M}$ , a derived verification key  $DVK^* \in L_{dvk}$ , and a signature  $\sigma^*$ .  $\mathcal{A}$  succeeds in the game if  $\text{Verify}(m^*, \sigma^*, DVK^*, PP) = 1$  under the restriction that (1)  $\text{ODSKCorrupt}(DVK^*)$  is never queried, and (2)  $\text{OSign}(m^*, DVK^*)$  is never queried.

9. This set is defined only for describing the game easier, storing the derived verification keys derived from the target master public key.



*Remark:* (1) Note that the adversary in the above model is allowed to generate derived verification keys and corrupt the corresponding signing keys of its choice. This captures the security requirement that the derived verification keys should be insulated from each other, i.e., for any specific derived verification key, even if all other verification keys derived from the same master public key are corrupted, the specific one is still safe. With such a security requirement, the security flaws in Monero's stealth address and Bitcoin's deterministic wallet are avoided. (2) Note that the adversary in the above model is allowed to obtain the master secret view key. This formally guarantees that for the trustless-audit use case, the auditor, who knows the master secret view key, is not able to steal the coins. (3) Although it is suggested that in cryptocurrency each public/verification key, as the coin address, is used only once, in this work we do not restrict PDPKS to one-time signature scheme, which requires that for each verification key the signing oracle can be queried at most once. Our proposed PDPKS provides stronger security, namely, even if the users use a derived key pair multiple times, the system is still safe. This enables the resulting PDPKS to resist more attacks in practice, for example, an attacker may somehow make the user to generate multiple signatures with respect to a target derived verification key.

### 3.3 Privacy Model

To capture the privacy-protection requirements that the derived verification keys and corresponding signatures do not leak any information that can be linked to the origin master public key or other derived verification keys, the privacy of a PDPKS scheme needs to consider two cases:

- *Case I:* Given a derived verification key and corresponding signatures, an adversary should not be able to tell which master public key, out of a set of known master public keys, is the one from which the verification key was derived.
- *Case II:* Given two derived verification keys and corresponding signatures, an adversary should not be able to tell whether they are generated from the same master public key.

Below we define the key privacy for Case I (referred to as master public key unlinkability) and Case II (referred to as derived verification key unlinkability) respectively, and prove that the key privacy for Case I implies that for Case II.

**Definition 2.** A PDPKS scheme is master public key unlinkable (MPK-UNL), if for all PPT adversaries  $\mathcal{A}$ , the advantage of  $\mathcal{A}$  in the following game  $\text{Game}_{\text{MPKUNL}}$ , denoted by  $\text{Adv}_{\mathcal{A}}^{\text{mpkunl}}$ , is negligible.

- Setup.  $\text{PP} \leftarrow \text{Setup}(\lambda)$  is run and  $\text{PP}$  are given to  $\mathcal{A}$ .  $(\text{MPK}_0, (\text{MSVK}_0, \text{MSSK}_0)) \leftarrow \text{MasterKeyGen}(\text{PP})$  and  $(\text{MPK}_1, (\text{MSVK}_1, \text{MSSK}_1)) \leftarrow \text{MasterKeyGen}(\text{PP})$  are run, and  $\text{MPK}_0, \text{MPK}_1$  are given to  $\mathcal{A}$ . An empty set  $L_{\text{dvk}} = \emptyset$  is initialized.<sup>10</sup>
- Phase 1.  $\mathcal{A}$  can adaptively query the following oracles:
  - Verification Key Adding Oracle  $\text{ODVKAdd}(\cdot, \cdot)$ :  
On input a derived verification key  $\text{DVK}$  and a

master public key  $\text{MPK} \in \{\text{MPK}_0, \text{MPK}_1\}$ , this oracle returns  $b \leftarrow \text{VrfyKeyCheck}(\text{DVK}, \text{MPK}, \text{MSVK}, \text{PP})$  to  $\mathcal{A}$ , where  $\text{MSVK}$  is the master secret view key corresponding to  $\text{MPK}$ . If  $b = 1$ , set  $L_{\text{dvk}} = L_{\text{dvk}} \cup \{\text{DVK}\}$ .

- Signing Key Corruption Oracle  $\text{ODSKCorrupt}(\cdot)$ :  
On input a derived verification key  $\text{DVK}$  in  $L_{\text{dvk}}$ , this oracle returns  $\text{DSK} \leftarrow \text{SignKeyDerive}(\text{DVK}, \text{MPK}, \text{MSVK}, \text{MSSK}, \text{PP})$  to  $\mathcal{A}$ , where  $\text{MPK}$  is the master public key that  $\text{DVK}$  is derived from, and  $(\text{MSVK}, \text{MSSK})$  is the master secret key corresponding to  $\text{MPK}$ .
- Signing Oracle  $\text{OSign}(\cdot, \cdot)$ :  
On input a message  $m \in \mathcal{M}$  and a derived verification key  $\text{DVK}$  in  $L_{\text{dvk}}$ , this oracle returns  $\sigma \leftarrow \text{Sign}(m, \text{DVK}, \text{DSK}, \text{PP})$  to  $\mathcal{A}$ , where  $\text{DSK}$  is a signing key corresponding to  $\text{DVK}$ .
- Challenge. A random bit  $b \in \{0, 1\}$  is chosen,  $\text{DVK}^* \leftarrow \text{VrfyKeyDerive}(\text{MPK}_b, \text{PP})$  is given to  $\mathcal{A}$ . Set  $L_{\text{dvk}} = L_{\text{dvk}} \cup \{\text{DVK}^*\}$ .
- Phase 2. Same as Phase 1, except that  $\text{ODVKAdd}(\text{DVK}^*, \text{MPK}_i)$  (for  $i \in \{0, 1\}$ ) cannot be queried.
- Guess.  $\mathcal{A}$  outputs a bit  $b' \in \{0, 1\}$  as its guess to  $b$ .  
 $\mathcal{A}$  succeeds in the game if  $b = b'$ . The advantage of  $\mathcal{A}$  is  $\text{Adv}_{\mathcal{A}}^{\text{mpkunl}} = |\Pr[b' = b] - \frac{1}{2}|$ .

*Remark:* (1) Note that the adversary in the above model is allowed to query  $\text{OSign}(\cdot, \text{DVK}^*)$ . This captures the privacy protection requirement in cryptocurrency that even after the owner of a coin (on a verification key) signs a transaction and spends the coin, the signature does not leak information that links the coin (and the transaction) to the owner's master public key. (2) Note that the adversary is even allowed to query  $\text{ODSKCorrupt}()$  on  $\text{DVK}^*$ . This implies that, even given the derived signing key corresponding to the challenge derived verification key, an adversary cannot tell which master public key the verification key is derived from.

**Definition 3.** A PDPKS scheme is derived verification key unlinkable (DVK-UNL), if for all PPT adversaries  $\mathcal{A}$ , the advantage of  $\mathcal{A}$  in the following game  $\text{Game}_{\text{DVKUNL}}$ , denoted by  $\text{Adv}_{\mathcal{A}}^{\text{dvkunl}}$ , is negligible.

- Setup. Same as that of  $\text{Game}_{\text{MPKUNL}}$ .
- Phase 1. Same as that of  $\text{Game}_{\text{MPKUNL}}$ .
- Challenge. A random bit  $c \in \{0, 1\}$  is chosen. Compute  $\text{DVK}_0^* \leftarrow \text{VrfyKeyDerive}(\text{MPK}_c, \text{PP})$ .  
A random bit  $b \in \{0, 1\}$  is chosen. If  $b = 0$ , compute  $\text{DVK}_1^* \leftarrow \text{VrfyKeyDerive}(\text{MPK}_c, \text{PP})$ , otherwise compute  $\text{DVK}_1^* \leftarrow \text{VrfyKeyDerive}(\text{MPK}_{1-c}, \text{PP})$ .  $(\text{DVK}_0^*, \text{DVK}_1^*)$  is given to  $\mathcal{A}$ . Set  $L_{\text{dvk}} = L_{\text{dvk}} \cup \{\text{DVK}_0^*, \text{DVK}_1^*\}$ .
- Phase 2. Same as Phase 1, except that  $\text{ODVKAdd}(\text{DVK}_j^*, \text{MPK}_i)$  (for  $j, i \in \{0, 1\}$ ) can be queried on at most one  $j \in \{0, 1\}$ .
- Guess.  $\mathcal{A}$  outputs a bit  $b' \in \{0, 1\}$  as its guess to  $b$ , i.e., guess whether  $\text{DVK}_0^*$  and  $\text{DVK}_1^*$  are derived from the same master public key.  
 $\mathcal{A}$  succeeds in the game if  $b = b'$ . The advantage of  $\mathcal{A}$  is  $\text{Adv}_{\mathcal{A}}^{\text{dvkunl}} = |\Pr[b' = b] - \frac{1}{2}|$ .

10. The set is defined only for describing the game easier, storing the derived verification keys derived from the target master public keys.



*Remark:* Note that the adversary in the above model is allowed to query  $\text{OSign}(\cdot, \text{DVK}_j^*)$  for  $j \in \{0, 1\}$  and  $\text{ODSKCorrupt}(\text{DVK}_j^*)$  (for  $j \in \{0, 1\}$ ).

The following theorem shows that the privacy of *Case II* is implied by that of *Case I*.

**Theorem 1.** *If a PDPKS scheme is master public key unlinkable, then it is also derived verification key unlinkable.*

**Proof.** Let  $\Pi$  be a PDPKS scheme, and  $\Pi$  is master public key unlinkable (w.r.t. Definition 2). Below we prove that  $\Pi$  is derived verification key unlinkable (w.r.t. Definition 3).  $\square$

Suppose there exists an adversary  $\mathcal{A}$  can win  $\text{Game}_{\text{DVKUNL}}$  for  $\Pi$  with non-negligible advantage, we can construct an algorithm  $\mathcal{B}$  that wins  $\text{Game}_{\text{MPKUNL}}$  with non-negligible advantage, which is contradict to  $\Pi$  is master public key unlinkable. Consider the following game where  $\mathcal{B}$  is interacting with a challenger  $\mathcal{C}$  to attack the master public key unlinkability of  $\Pi$  in  $\text{Game}_{\text{MPKUNL}}$ , while from  $\mathcal{A}$ 's point of view,  $\mathcal{A}$  is attacking the derived verification key unlinkability of  $\Pi$  in  $\text{Game}_{\text{DVKUNL}}$ .

**Setup.**  $\mathcal{B}$  is given  $\text{PP}$ ,  $\text{MPK}_0$  and  $\text{MPK}_1$ , then  $\mathcal{B}$  forwards  $\text{PP}$ ,  $\text{MPK}_0$  and  $\text{MPK}_1$  to  $\mathcal{A}$ .

**Phase 1.** When  $\mathcal{A}$  makes query to the oracles  $\text{ODVKAdd}(\cdot, \cdot)$ ,  $\text{ODSKCorrupt}(\cdot)$ ,  $\text{OSign}(\cdot, \cdot)$ ,  $\mathcal{B}$  just makes the same query to  $\mathcal{C}$ , and forwards the results to  $\mathcal{A}$ .

**Challenge.**  $\mathcal{B}$  receives  $\text{DVK}^*$ , which is derived from  $\text{MPK}_i$  by the challenger  $\mathcal{C}$ .

$\mathcal{B}$  chooses a random  $\tilde{c} \in \{0, 1\}$ , and computes  $\widetilde{\text{DVK}}_0^* \leftarrow \text{VrfyKeyDerive}(\text{MPK}_{\tilde{c}}, \text{PP})$ .  $\mathcal{B}$  sets  $\text{DVK}_1^* = \text{DVK}^*$ .

$\mathcal{B}$  chooses a random  $\tilde{c} \in \{0, 1\}$ . If  $\tilde{c} = 0$ ,  $\mathcal{B}$  sets  $(\text{DVK}_0^*, \text{DVK}_1^*) = (\text{DVK}_0^*, \text{DVK}_1^*)$ , otherwise,  $\mathcal{B}$  sets  $(\text{DVK}_0^*, \text{DVK}_1^*) = (\text{DVK}_1^*, \text{DVK}_0^*)$ .  $\mathcal{B}$  returns  $(\text{DVK}_0^*, \text{DVK}_1^*)$  to  $\mathcal{A}$ .

$\mathcal{B}$  sets  $L_{\text{dvk}} = L_{\text{dvk}} \cup \{\text{DVK}_0^*, \text{DVK}_1^*\}$ .

**Phase 2.** At the begin of *Phase 2*,  $\mathcal{B}$  makes query  $\text{ODVKAdd}(\text{DVK}_0^*, \text{MPK}_{\tilde{c}})$  to  $\mathcal{C}$ . Note that  $\text{DVK}_0^*$  is honestly derived from  $\text{MPK}_{\tilde{c}}$ ,  $\mathcal{C}$  will return 1 to  $\mathcal{B}$ , implying  $\text{DVK}_0^*$  is accepted by  $\mathcal{C}$  as a valid derived verification key and later when  $\mathcal{B}$  makes query  $\text{ODSKCorrupt}()$  on  $\text{DVK}_0^*$ ,  $\mathcal{C}$  will responds with a corresponding derived signing key. Also, later when  $\mathcal{B}$  makes query  $\text{ODVKAdd}(\cdot, \cdot)$  on  $(\text{DVK}_0^*, \text{MPK})$  for  $\text{MPK} \in \{\text{MPK}_0, \text{MPK}_1\}$ ,  $\mathcal{C}$  will answer accordingly.

Then, when  $\mathcal{A}$  makes query to oracles  $\text{ODSKCorrupt}(\cdot)$ ,  $\text{OSign}(\cdot, \cdot)$ ,  $\mathcal{B}$  makes the same query to  $\mathcal{C}$ , and forwards the results to  $\mathcal{A}$ ; when  $\mathcal{A}$  makes query to oracles  $\text{ODVKAdd}(\cdot, \cdot)$  on input  $(\text{DVK}, \text{MPK})$ ,

- If  $\text{DVK} \neq \widetilde{\text{DVK}}_1^*$ :  $\mathcal{B}$  makes the same query to  $\mathcal{C}$ , and forwards the results to  $\mathcal{A}$ .
- If  $\text{DVK} = \widetilde{\text{DVK}}_1^*$ : note that  $\mathcal{B}$  is not allowed to make query  $\text{ODVKAdd}(\text{DVK}^*, \cdot)$  to  $\mathcal{C}$ ,  $\mathcal{B}$  aborts the game, and returns a random  $b' \in \{0, 1\}$  to  $\mathcal{C}$ . Denote this event by  $\mathbf{E}$ .

**Guess.**  $\mathcal{A}$  outputs a bit  $\tilde{b}' \in \{0, 1\}$  as its guess on whether  $\text{DVK}_0^*$  and  $\text{DVK}_1^*$  are from the same master public key.  $\mathcal{B}$  sets  $b' = \tilde{b}' \oplus \tilde{c}$  and returns  $b'$  to  $\mathcal{C}$ . Note that

- $\tilde{b}' = 0$  implies  $\mathcal{A}$  is guessing that  $\text{DVK}_0^*$  and  $\text{DVK}_1^*$  are derived from the same public key, and this implies

that  $\text{DVK}^*$  is also derived from  $\text{MPK}_{\tilde{c}}$ . Thus, if  $\tilde{c} = 0$ ,  $\mathcal{B}$  sets  $b' = 0$ , otherwise  $\mathcal{B}$  sets  $b' = 1$ . This means  $\mathcal{B}$  sets  $b' = \tilde{b}' \oplus \tilde{c}$ .

- $\tilde{b}' = 1$  implies  $\mathcal{A}$  is guessing that  $\text{DVK}_0^*$  and  $\text{DVK}_1^*$  are derived from different public keys, and this implies that  $\text{DVK}^*$  is derived from  $\text{MPK}_{1-\tilde{c}}$ . Thus, if  $\tilde{c} = 0$ ,  $\mathcal{B}$  sets  $b' = 1$ , otherwise  $\mathcal{B}$  sets  $b' = 0$ . This means  $\mathcal{B}$  sets  $b' = \tilde{b}' \oplus \tilde{c}$ .

**Analysis.** The advantage of  $\mathcal{B}$  in  $\text{Game}_{\text{MPKUNL}}$  is

$$\begin{aligned} \text{Adv}_{\mathcal{B}} &= |\Pr[b' = b] - 1/2| \\ &= |\Pr[b' = b|\mathbf{E}] \cdot \Pr[\mathbf{E}] + \Pr[b' = b|\neg\mathbf{E}] \cdot \Pr[\neg\mathbf{E}] - \frac{1}{2}|. \end{aligned}$$

Note that when event  $\mathbf{E}$  happens, we have  $\Pr[b' = b|\mathbf{E}] = 1/2$ , and when the event  $\mathbf{E}$  does not happen, we have

$$\Pr[b' = b|\neg\mathbf{E}] = \Pr[\tilde{b}' \oplus \tilde{c} = b] = \Pr[\tilde{b}' = b \oplus \tilde{c}].$$

Thus, we have

$$\begin{aligned} \text{Adv}_{\mathcal{B}} &= \left| \frac{1}{2} \cdot \Pr[\mathbf{E}] + \Pr[\tilde{b}' = b \oplus \tilde{c}] \cdot \Pr[\neg\mathbf{E}] - \frac{1}{2} \right| \\ &= |\Pr[\tilde{b}' = b \oplus \tilde{c}] \cdot \Pr[\neg\mathbf{E}] - \frac{1}{2}(1 - \Pr[\mathbf{E}])| \\ &= |\Pr[\tilde{b}' = b \oplus \tilde{c}] \cdot \Pr[\neg\mathbf{E}] - \frac{1}{2}\Pr[\neg\mathbf{E}]| \\ &= |\Pr[\tilde{b}' = b \oplus \tilde{c}] - \frac{1}{2}| \cdot \Pr[\neg\mathbf{E}]. \end{aligned}$$

Note that  $\mathcal{A}$  is allowed to query  $\text{ODVKAdd}(\text{DVK}_j^*, \text{MPK}_i)$  (for  $j, i \in \{0, 1\}$ ) on at most one  $j \in \{0, 1\}$  and the distributions of  $(\text{DVK}_0^*, \text{DVK}_1^*)$  and  $(\text{DVK}_1^*, \text{DVK}_0^*)$  are indistinguishable to  $\mathcal{A}$ , we have that  $\Pr[\mathbf{E}] \leq 1/2$ . Also note that the advantage of  $\mathcal{A}$  in  $\text{Game}_{\text{DVKUNL}}$  is

$$\begin{aligned} \text{Adv}_{\mathcal{A}} &= |\Pr[\tilde{b}' = 0|\tilde{c} = b] + \Pr[\tilde{b}' = 1|\tilde{c} = 1 - b] - 1/2| \\ &= |\Pr[\tilde{b}' = \tilde{c} \oplus b] - 1/2|. \end{aligned}$$

Thus, we have  $\text{Adv}_{\mathcal{B}} = \text{Adv}_{\mathcal{A}} \cdot (1 - \Pr[\mathbf{E}]) \geq \frac{1}{2} \cdot \text{Adv}_{\mathcal{A}}$ .  $\square$

Theorem 1 provides a formal conclusion that, for the privacy in PDPKS scheme, it is sufficient to consider the master public key unlinkability only.

## 4 OUR CONSTRUCTION

In this section, we first present some preliminaries, including the bilinear groups and the assumptions, then we propose a PDPKS construction, which is obtained by applying our approach described in Section 2.3 to the IBS scheme by Barreto *et al.* [33].

### 4.1 Preliminaries

#### 4.1.1 Bilinear Map Groups [35]

Let  $\lambda$  be a security parameter and  $p$  be a  $\lambda$ -bit prime number. Let  $\mathbb{G}_1$  and  $\mathbb{G}_2$  be two additive cyclic groups of order  $p$ ,  $\mathbb{G}_T$  be a multiplicative cyclic group of order  $p$ , and  $P, Q$  be generators of  $\mathbb{G}_1$  and  $\mathbb{G}_2$  respectively.  $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$  are bilinear map groups if there exists a bilinear map  $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$  satisfying the following properties:

- (1) **Bilinearity:**  $\forall (S, T) \in \mathbb{G}_1 \times \mathbb{G}_2, \forall a, b \in \mathbb{Z}, e(aS, bT) = e(S, T)^{ab}$ .
- (2) **Non-degeneracy:**  $e(P, Q)$  is a generator of  $\mathbb{G}_T$ .
- (3) **Computability:**  $\forall (S, T) \in \mathbb{G}_1 \times \mathbb{G}_2, e(S, T)$  is efficiently computable.
- (4) There exists an efficient, publicly computable (but not necessarily invertible) isomorphism  $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$  such that  $\psi(Q) = P$ .

One can set  $\mathbb{G}_1 = \mathbb{G}_2, P = Q$ , and take  $\psi$  to be the identity map.

#### 4.1.2 Assumptions

The security of our PDPKS construction relies on the  $q$ -Strong Diffie-Hellman ( $q$ -SDH) Assumption [36], while the privacy relies on the Computational Diffie-Hellman (CDH) Assumption [37] on bilinear groups.

**Definition 4 ( $q$ -SDH Assumption).** [33], [36] *The  $q$ -SDH problem in  $(\mathbb{G}_1, \mathbb{G}_2)$  is defined as follows: given a  $q + 2$ -tuple  $(P, Q, \beta Q, \beta^2 Q, \dots, \beta^q Q) \in \mathbb{G}_1 \times \mathbb{G}_2^{q+1}$  as input, output a pair  $(c, \frac{1}{c+\beta}P)$  with  $c \in \mathbb{Z}_p^*$ . An algorithm  $\mathcal{A}$  has advantage  $\epsilon$  in solving  $q$ -SDH in  $(\mathbb{G}_1, \mathbb{G}_2)$  if*

$$\Pr \left[ \mathcal{A}(P, Q, \beta Q, \beta^2 Q, \dots, \beta^q Q) = \left( c, \frac{1}{c+\beta}P \right) \right] \geq \epsilon,$$

where the probability is over the random choice of  $\beta$  in  $\mathbb{Z}_p^*$  and the random bits consumed by  $\mathcal{A}$ .

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

**Definition 5 (CDH Assumption).** [37] *The CDH problem in  $\mathbb{G}_2$  is defined as follows: given a tuple  $(Q, A = aQ, B = bQ) \in \mathbb{G}_2^3$  as input, output  $C = abQ \in \mathbb{G}_2$ . An algorithm  $\mathcal{A}$  has advantage  $\epsilon$  in solving CDH in  $\mathbb{G}_2$  if  $\Pr [ \mathcal{A}(Q, aQ, bQ) = abQ ] \geq \epsilon$ , where the probability is over the random choice of  $a, b \in \mathbb{Z}_p^*$  and the random bits consumed by  $\mathcal{A}$ .*

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

## 4.2 Construction

- **Setup** $(\lambda) \rightarrow \text{PP}$ . On input a security parameter  $\lambda$ , the algorithm chooses bilinear map groups  $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \psi)$  of prime order  $p > 2^\lambda$ , generators  $Q \in \mathbb{G}_2, P = \psi(Q) \in \mathbb{G}_1, g = e(P, Q)$ , and hash functions  $H_1 : \mathbb{G}_2 \times \mathbb{G}_2 \rightarrow \mathbb{Z}_p^*, H_2 : \{0, 1\}^* \times \mathbb{G}_T \rightarrow \mathbb{Z}_p^*$ . The algorithm outputs public parameters

$$\text{PP} := (p, (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \psi), P, Q, g, H_1, H_2),$$

and the message space is  $\mathcal{M} = \{0, 1\}^*$ .

- **MasterKeyGen** $(\text{PP}) \rightarrow (\text{MPK}, (\text{MSVK}, \text{MSSK}))$ . On input the system public parameters  $\text{PP}$ , the algorithm chooses random  $\alpha, \beta \in \mathbb{Z}_p^*$ , then outputs a master public key  $\text{MPK}$  and corresponding master secret key  $\text{MSK} := (\text{MSVK}, \text{MSSK})$  as

$$\text{MPK} := (Q_{\text{pub},1}, Q_{\text{pub},2}) = (\alpha Q, \beta Q) \in \mathbb{G}_2 \times \mathbb{G}_2,$$

$$\text{MSVK} = \alpha, \quad \text{MSSK} = \beta.$$

- **VrfyKeyDerive** $(\text{MPK}, \text{PP}) \rightarrow \text{DVK}$ . On input  $\text{MPK} = (Q_{\text{pub},1}, Q_{\text{pub},2}) \in \mathbb{G}_2 \times \mathbb{G}_2$  and the system public parameters  $\text{PP}$ , the algorithm chooses random  $r \in \mathbb{Z}_p^*$ , and outputs a derived verification key

$$\begin{aligned} \text{DVK} &:= (Q_r, Q_{\text{vk}}) \\ &= (rQ, H_1(rQ, rQ_{\text{pub},1})Q + Q_{\text{pub},2}) \in \mathbb{G}_2 \times \mathbb{G}_2. \end{aligned}$$

- **VrfyKeyCheck** $(\text{DVK}, \text{MPK}, \text{MSVK}, \text{PP}) \rightarrow 1/0$ . On input  $\text{DVK} = (Q_r, Q_{\text{vk}}) \in \mathbb{G}_2 \times \mathbb{G}_2, \text{MPK} = (Q_{\text{pub},1}, Q_{\text{pub},2}) \in \mathbb{G}_2 \times \mathbb{G}_2, \text{MSVK} = \alpha \in \mathbb{Z}_p^*$ , and the system public parameters  $\text{PP}$ , the algorithm checks whether  $Q_{\text{vk}} \stackrel{?}{=} H_1(Q_r, \alpha Q_r)Q + Q_{\text{pub},2}$ . If it holds, the algorithm outputs 1, otherwise outputs 0.
- **SignKeyDerive** $(\text{DVK}, \text{MPK}, \text{MSVK}, \text{MSSK}, \text{PP}) \rightarrow \text{DSK}$  or  $\perp$ . On input  $\text{DVK} = (Q_r, Q_{\text{vk}}) \in \mathbb{G}_2 \times \mathbb{G}_2, \text{MPK} = (Q_{\text{pub},1}, Q_{\text{pub},2}) \in \mathbb{G}_2 \times \mathbb{G}_2, \text{MSVK} = \alpha \in \mathbb{Z}_p^*$ , and  $\text{MSSK} = \beta \in \mathbb{Z}_p^*$ , the algorithm checks whether  $Q_{\text{vk}} \stackrel{?}{=} H_1(Q_r, \alpha Q_r)Q + Q_{\text{pub},2}$ . If it holds, the algorithm outputs a derived signing key

$$\text{DSK} := P_{sk} = \frac{1}{H_1(Q_r, \alpha Q_r) + \beta} P \in \mathbb{G}_1,$$

otherwise, outputs  $\perp$ .

- **Sign** $(m, \text{DVK}, \text{DSK}, \text{PP}) \rightarrow \sigma$ . On input a message  $m \in \mathcal{M}$ , a derived verification key  $\text{DVK} = (Q_r, Q_{\text{vk}}) \in \mathbb{G}_2 \times \mathbb{G}_2$ , a signing key  $\text{DSK} = P_{sk} \in \mathbb{G}_1$ , and the system public parameters  $\text{PP}$ , the algorithm
  - (1) picks a random  $x \in \mathbb{Z}_p^*$  and computes  $X = g^x$ ,
  - (2) sets  $h = H_2(m, X) \in \mathbb{Z}_p^*$ ,
  - (3) computes  $P_\sigma = (x + h)P_{sk} \in \mathbb{G}_1$ ,
 and outputs  $\sigma = (h, P_\sigma)$  as a signature for  $m$ .
- **Verify** $(m, \sigma, \text{DVK}, \text{PP}) \rightarrow 1/0$ . On input a message  $m \in \mathcal{M}$ , a signature  $\sigma = (h, P_\sigma) \in \mathbb{Z}_p^* \times \mathbb{G}_1$ , a derived verification key  $\text{DVK} = (Q_r, Q_{\text{vk}}) \in \mathbb{G}_2 \times \mathbb{G}_2$ , and the system public parameters  $\text{PP}$ , the algorithm checks whether  $h \stackrel{?}{=} H_2(m, e(P_\sigma, Q_{\text{vk}})g^{-h})$  holds. If it holds, the algorithm outputs 1, otherwise 0.

**Correctness.** For any message  $m \in \mathcal{M}$ , it is easy to verify that (1)  $\text{VrfyKeyCheck}(\text{DVK}, \text{MPK}, \text{MSVK}, \text{PP}) = 1$ , since  $\alpha Q_r = \alpha rQ = rQ_{\text{pub},1}$ , and

(2)  $\text{Verify}(m, \text{Sign}(m, \text{DVK}, \text{DSK}, \text{PP}), \text{DVK}, \text{PP}) = 1$ , since

$$\begin{aligned} e(P_\sigma, Q_{\text{vk}})g^{-h} &= e((x + h)P_{sk}, Q_{\text{vk}})g^{-h} \\ &= e(P_{sk}, Q_{\text{vk}})^{x+h}g^{-h} = g^{x+h}g^{-h} = g^x = X. \end{aligned}$$

## 5 PROOFS OF SECURITY AND PRIVACY

The following theorem states that the proposed PDPKS construction in Section 4 is secure (w.r.t. Definition 1) in the random oracle model.

**Theorem 2.** *The PDPKS scheme is secure under the  $q$ -SDH assumption in the random oracle model provided that  $q_{h_1} + q_a \leq q$ , where  $q_{h_1}$  and  $q_a$  denote the number of queries to the random oracle  $H_1$  and the verification key adding oracle, respectively.*

**Proof.** Note that when compared with the preliminary version of this work [1],

- the construction in Section 4 is identical to that in [1, Section IV], except that the master secret key  $MSK = (\alpha, \beta)$  is separated into two parts  $MSVK = \alpha$  and  $MSSK = \beta$ ;
- the security model in Definition 1 is identical to that in [1, Definition 1], except that the adversary is allowed to query an additional Master Secret View Key Corruption Oracle  $O_{MSVKCorrupt}()$ .

Note that in the proof of [1, Lemma 2], the algorithm  $\mathcal{B}$  chooses the value of  $\alpha$  by itself. As a result, we can directly modify the proof of [1, Theorem 1] to a proof for our Theorem 2 here, namely, when the adversary makes the query  $O_{MSVKCorrupt}()$ ,  $\mathcal{B}$  returns  $\alpha$  to the adversary. We refer to [1] for the proof details.  $\square$

The following theorem states that the proposed PDPKS construction in Section 4 is master public key unlinkable (w.r.t. Definition 2)

**Theorem 3.** *The PDPKS scheme is master public key unlinkable under the CDH assumption in the random oracle model. Specifically, assume that there exists an attacker  $\mathcal{A}$  that runs within time  $t$  and makes  $q_{h_1}$  queries to random oracle  $H_1$ ,  $q_a$  queries to the verification key adding oracle, and  $q_s$  queries to the signing oracle, and wins  $\text{Game}_{\text{MPKSUNL}}$  with advantage  $\epsilon$ , then there exists an algorithm  $\mathcal{B}$  that runs within time  $\bar{t} = t + O((q_{h_1} + q_s)\tau_{\text{mult}}) + O((q_{h_1} + q_a)\tau_p) + O(q_s\tau_{\text{exp}})$ , where  $\tau_{\text{exp}}$  denotes the time for an exponentiation operation in  $\mathbb{G}_T$ , and solves the CDH problem with probability at least  $\epsilon - q_a/p$ .*

**Proof.** Note that when compared with the preliminary version [1] of this work, the separation of  $MSK$  into  $MSVK$  and  $MSSK$  does not affect the privacy model, the proof here for master public key unlinkability is identical to that of public key strongly unlinkability in [1, Theorem 2]. We refer to [1] for the proof details.  $\square$

## 6 APPLICATIONS

As PDPKS is defined to capture the functionality, privacy, and security requirements for stealth address, the proposed PDPKS construction naturally provides a secure and convenient tool for implementing stealth addresses in cryptocurrencies. Below we show that the proposed PDPKS can also support the use cases of deterministic wallet very well, and importantly, without the security vulnerabilities.

- (1) *Low-maintenance wallets with easy backup and recovery.* To backup his deterministic wallet, a user only needs to backup the master secret key  $(\alpha, \beta)$ . When needed, he could reconstruct the complete wallet, by using  $\alpha$  to scan the ledger to find the transaction-outputs belonging to him, and using  $(\alpha, \beta)$  to generate the corresponding derived signing keys.
- (2) *Freshly generated cold addresses.* With the PDPKS scheme, a user can publish his master public key on a hot storage, without affecting the security or privacy. To use the cold address mechanisms, he can easily generate derived verification keys as he needs. In addition, a user even can ask the payer to generate

the derived verification keys for the transaction sending coins to him, and he only needs to check and ensure that no derived verification key is used more than once.

- (3) *Trustless audits.* For a user with master secret view key  $MSVK = \alpha$  and master secret spend key  $MSSK = \beta$ , revealing  $MSVK = \alpha$  to an auditor will enable the auditor to view all the transactions related to his master public key  $MPK = (Q_{\text{pub},1} = \alpha G, Q_{\text{pub},2} = \beta G)$ , since for any transaction-output with verification key  $DVK = (Q_r, Q_{vk})$ , the auditor can run the  $\text{VrfyKeyCheck}()$  algorithm, i.e., check whether  $Q_{vk} \stackrel{?}{=} H_1(Q_r, \alpha Q_r)Q + Q_{\text{pub},2}$  holds. The security proof formally provides solid confidence that the auditors are not able to steal the users' coins.
- (4) *Hierarchical Wallet allowing a treasurer to allocate funds to departments.* The treasurer does not need to worry that the department managers collude to steal the funds of other departments, no matter how many of them collude.
- (5) *Simultaneously implementing the treasurer and the auditor use cases.* The treasurer does not need to worry that the department managers and the auditors collude to steal the funds of other departments, no matter how many of them collude.

We would like to point out that, our PDPKS construction can be directly used by a cryptocurrency as the underlying signature scheme and provides the functionality and privacy-protection features with solid security guarantee, but it does not mean a direct solution for fixing the Monero's security vulnerability discussed in Section 1. More specifically, while our PDPKS construction provides a secure stealth address algorithm, Monero's cryptographic protocols include not only the mentioned stealth address algorithm, but also a linkable ring signature scheme where the public key and signing key are generated using the stealth address algorithm. To fix the Monero's security vulnerability discussed in Section 1, a new suite of protocols (including the linkable ring signature) have to be re-designed based on our PDPKS construction (as the stealth address algorithm).

## 7 IMPLEMENTATION

On the implementation, note that our construction is using a type-2 pairing [38] and does not need to hash to  $\mathbb{G}_2$ , so it can be implemented based on any pairing friendly curve [38]. We select the Barreto-Naehrig (BN) curve [39], which has been well studied and regarded as an efficient and popular curve for high security level, say 128-bits of security or higher. On the concrete parameter for achieving 128-bits security, we adopt the parameter recommended in the recent work by Barbulescu and Duquesne [40, Section 6.1], i.e., the BN curve with parameter  $u = 2^{114} + 2^{101} - 2^{14} - 1$ , which implies that the group order  $p$  is 462-bits, the elements in  $\mathbb{G}_1$  and  $\mathbb{G}_2$  are 462-bits and 924-bits respectively. It is worth mentioning that a 256-bits prime  $p$ , and the resulting 256-bits  $\mathbb{G}_1$  and 512-bits  $\mathbb{G}_2$  are supposed to match the 128-bit security level according to the NIST recommendations [41], which are however now invalidated by Kim and Barbulescu's recent progress on number field sieve



TABLE 3  
Comparison With ECDSA-Based DW and SA: Efficiency

	Signing Time	Verification Time	Signature Size	Verification key Size	Signing Key Size	Master Public Key Size	Master Secret Key Size
ECDSA-based DW <sup>1</sup>	0.020 ms	0.074 ms	64 Byte	33 Byte	32 Byte	33 Byte	32 Byte
ECDSA-based SA <sup>1</sup>	0.020 ms	0.074 ms	64 Byte	66 Byte	32 Byte	66 Byte	32 Byte + 32 Byte
PDPKS in this work	4.323 ms	7.298 ms	116 Byte	232 Byte	58 Byte	232 Byte	58 Byte + 58 Byte

<sup>1</sup>ECDSA is widely used in the cryptocurrency community, as the underlying signature algorithm. For these cryptocurrencies, as discussed previously, deterministic wallet (DW) and stealth address (SA) may be implemented using the methods in Figs. 1 and 2 respectively, but with the security vulnerabilities as discussed in Section 1.

TABLE 4  
Comparison With ECDSA-Based DW and SA: Functionality, Security, and Privacy

	Support DW Functionality?	(DW) Support Auditor and Treasurer Simultaneously?	Support SA Functionality?	Secure against Derived Secret Key Leakage?	Allow Master Public Key Leakage? <sup>1</sup>	Formally Modeled and Proved? <sup>2</sup>
ECDSA-based DW	✓	×	×	×	×	×
ECDSA-based SA	×	NA	✓	×	✓	×
PDPKS in this work	✓	✓	✓	✓	✓	✓

<sup>1</sup>The functionality and privacy-protection features of ECDSA-based DW heavily rely on the secrecy of master public key. Once the master public key is leaked somehow, for example due to its exposure at the hot storage, some claimed functionality and privacy-protection features will fail.

<sup>2</sup>Formalized definitions and models, and the provable security and privacy will provide strong confidence on the functionality, security, and privacy.

algorithm for discrete logarithms in  $F_{p^n}$  [42]. That is why we use the above parameter recommended by Barbulescu and Duquesne [40, Section 6.1], which has taken into account the attacking algorithm in [42].

On the concrete implementation, we implement our PDPKS construction using C-language, based on the BN462 curve in MCL-library [43], achieving 128-bit security. Our source codes are available at [44]. To compare with ECDSA, which is widely used in cryptocurrencies, we also run the 128-bit secure ECDSA implementation of MCL [43] in the same environment (namely, a virtual machine with Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz, 1GB Memory, and the operating system ubuntu 16.04 LTS.). Table 3 shows the comparison of signing time and verification time, as well as the sizes of keys, between the deterministic wallet and stealth address algorithms based on ECDSA (using the algorithms in Figs. 1 and 2 respectively) and our PDPKS. Note that the signing and verification of our PDPKS construction are slower than that of ECDSA, and the key sizes are larger, but still practical for the settings of cryptocurrency. It is worth pointing out that, whereas the deterministic wallets and stealth addresses based on ECDSA may suffer the security and functionality flaws as discussed in Section 1, our PDPKS construction, with its solid guarantee on security and privacy, provides an appropriate (namely, *secure* and *practical*) candidate for the underlying signature scheme of cryptocurrencies, especially when deterministic wallet and/or stealth address are considered. Note that for the settings of cryptocurrency, security is more important than efficiency, and it is worth to take such a modest cost on efficiency to address the security concerns. It is also worth pointing out that, with formal definitions and models, as well as the provable security and privacy, our PDPKS construction provides much stronger confidence on the functionality, security, and privacy. For example, the existing stealth address algorithms only

*informally discussed* that the derived verification keys seem to be unlinkable to the origin master public key, while our PDPKS construction *proves* that its derived verification keys and corresponding signatures do not leak any information that can be linked to the origin master public key and other derived verification keys, which fully captures the privacy requirements in cryptocurrencies. Table 4 shows the comparison of functionality, security and privacy between the deterministic wallet and stealth address algorithms based on ECDSA and our PDPKS.

## 8 ANOTHER PDPKS CONSTRUCTION

Below we review the IBS construction in [31, Section 2], show that it has the MPK-pack-able Property, <sup>11</sup> and present a PDPKS construction obtained by applying our approach in Section 2.3 to this IBS.

### 8.1 The IBS Scheme in [31, Section 2]

- **Setup**( $\lambda$ )  $\rightarrow$  (PP, MSK). On input a security parameter  $\lambda$ , the algorithm chooses bilinear map groups  $(\mathbb{G}, \mathbb{G}_T, e)$  of prime order  $p > 2^\lambda$ , generators  $P \in \mathbb{G}$ , and hash functions  $H_1 : \{0, 1\}^* \rightarrow \mathbb{G}^*$ ,  $H_2 : \{0, 1\}^* \times \mathbb{G}_T \rightarrow \mathbb{Z}_p^*$ , where  $\mathbb{G}^* = \mathbb{G} \setminus \{0\}$ . <sup>12</sup>The algorithm selects random  $\beta \in \mathbb{Z}_p^*$  and computes  $B = \beta P \in \mathbb{G}$ , then outputs public parameters PP and master secret key MSK as  $\text{PP} := (p, (\mathbb{G}, \mathbb{G}_T, e), P, B, H_1, H_2)$ ,  $\text{MSK} := \beta$ .  
The message space is  $\mathcal{M} = \{0, 1\}^*$ .

<sup>11</sup>Note that we slightly changed the variable names in the IBS construction.

<sup>12</sup>Both the constructions and the underlying CDH assumption are on the bilinear map groups where  $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$ ,  $P = Q$ , and  $\phi$  is the identity map.



- **KeyExtract**(ID, PP, MSK)  $\rightarrow$  SK<sub>ID</sub>. On input an arbitrary identity ID  $\in \{0, 1\}^*$ , the system public parameters PP, and the master secret key MSK, the algorithm outputs a private key SK<sub>ID</sub> for the identity ID as  $\text{SK}_{\text{ID}} = \beta H_1(\text{ID}) \in \mathbb{G}$ .
- **Sign**( $m, \text{ID}, \text{PP}, \text{SK}_{\text{ID}}$ )  $\rightarrow \sigma$ . On input a message  $m \in \{0, 1\}^*$ , an identity ID  $\in \{0, 1\}^*$ , the system public parameters PP, and a private key SK<sub>ID</sub> for the identity ID, the algorithm
  - (1) picks a random  $x \in \mathbb{Z}_p^*$  and a random  $P_1 \in \mathbb{G}$ , and computes  $X = e(P_1, P)^x \in \mathbb{G}_T$ ,
  - (2) sets  $h = H_2(m, X) \in \mathbb{Z}_p^*$ ,
  - (3) computes  $P_\sigma = h \text{SK}_{\text{ID}} + x P_1 \in \mathbb{G}$ , and outputs  $\sigma = (h, P_\sigma)$  as a signature for message  $m$  and identity ID.
- **Verify**( $m, \sigma, \text{ID}, \text{PP}$ )  $\rightarrow 1/0$ . On input a message  $m \in \{0, 1\}^*$ , a signature  $\sigma = (h, P_\sigma) \in \mathbb{Z}_p^* \times \mathbb{G}$ , an identity ID  $\in \{0, 1\}^*$ , and the system public parameters PP, the algorithm outputs  $b = 1$  if and only if  $h = H_2(m, e(P_\sigma, P) \cdot e(H_1(\text{ID}), -B)^h)$ .

**Note that the above IBS scheme has the MPK -pack-able property in the sense that**

$$\text{CMPK} := (p, (\mathbb{G}, \mathbb{G}_T, e), P, H_1, H_2), \quad \text{IMPK} := (B),$$

$$F(\text{PP}, \text{ID}) := e(H_1(\text{ID}), -B).$$

## 8.2 Another PDPKS Construction

- **Setup**( $\lambda$ )  $\rightarrow$  PP. On input a security parameter  $\lambda$ , the algorithm chooses bilinear map groups  $(\mathbb{G}, \mathbb{G}_T, e)$  of prime order  $p > 2^\lambda$ , generators  $P \in \mathbb{G}$ , and hash functions  $H_1 : \{0, 1\}^* \rightarrow \mathbb{G}^*$ ,  $H_2 : \{0, 1\}^* \times \mathbb{G}_T \rightarrow \mathbb{Z}_p^*$ , where  $\mathbb{G}^* = \mathbb{G} \setminus \{0\}$ . The algorithm outputs public parameters

$$\text{PP} := (p, (\mathbb{G}, \mathbb{G}_T, e), P, H_1, H_2),$$

and the message space is  $\mathcal{M} = \{0, 1\}^*$ .

- **MasterKeyGen**(PP)  $\rightarrow$  (MPK, (MSVK, MSSK)). On input the system public parameters PP, the algorithm chooses random  $\alpha, \beta \in \mathbb{Z}_p^*$ , then outputs a master public key MPK and corresponding master secret key  $\text{MSK} := (\text{MSVK}, \text{MSSK})$  as

$$\text{MPK} := (A, B) = (\alpha P, \beta P) \in \mathbb{G} \times \mathbb{G},$$

$$\text{MSVK} = \alpha, \quad \text{MSSK} = \beta.$$

- **VrfyKeyDerive**(MPK, PP)  $\rightarrow$  DVK. On input MPK = (A, B)  $\in \mathbb{G} \times \mathbb{G}$  and the system public parameters PP, the algorithm chooses random  $r \in \mathbb{Z}_p^*$ , and outputs a derived verification key

$$\text{DVK} := (R, T_{vk}) = (rP, e(H_1(rP, rA), -B)) \in \mathbb{G} \times \mathbb{G}_T.$$

- **VrfyKeyCheck**(DVK, MPK, MSVK, PP)  $\rightarrow 1/0$ . On input DVK = (R, T<sub>vk</sub>)  $\in \mathbb{G} \times \mathbb{G}_T$ , MPK = (A, B)  $\in \mathbb{G} \times \mathbb{G}$ , MSVK =  $\alpha \in \mathbb{Z}_p^*$ , and the system public parameters PP, the algorithm checks whether  $T_{vk} \stackrel{?}{=} e(H_1(R, \alpha R), -B)$ .

If it holds, the algorithm outputs 1, otherwise outputs 0.

- **SignKeyDerive**(DVK, MPK, MSVK, MSSK, PP)  $\rightarrow$  DSK or  $\perp$ . On input DVK = (R, T<sub>vk</sub>)  $\in \mathbb{G} \times \mathbb{G}_T$ , MPK = (A, B)  $\in \mathbb{G} \times \mathbb{G}$ , MSVK =  $\alpha \in \mathbb{Z}_p^*$ , MSSK =  $\beta \in \mathbb{Z}_p^*$ , and the system public parameters PP, the algorithm checks whether  $T_{vk} \stackrel{?}{=} e(H_1(R, \alpha R), -B)$ . If it holds, the algorithm outputs a derived signing key

$$\text{DSK} := S_{sk} = \beta H_1(R, \alpha R) \in \mathbb{G},$$

otherwise, outputs  $\perp$ .

- **Sign**( $m, \text{DVK}, \text{DSK}, \text{PP}$ )  $\rightarrow \sigma$ . On input a message  $m \in \mathcal{M}$ , a derived verification key DVK = (R, T<sub>vk</sub>)  $\in \mathbb{G} \times \mathbb{G}_T$ , a signing key DSK =  $S_{sk} \in \mathbb{G}$ , and the system public parameters PP, the algorithm
  - (1) picks a random  $x \in \mathbb{Z}_p^*$ , and computes  $X = e(P, P)^x \in \mathbb{G}_T$ ,<sup>13</sup>
  - (2) sets  $h = H_2(m, X) \in \mathbb{Z}_p^*$ ,
  - (3) computes  $P_\sigma = h S_{sk} + x P \in \mathbb{G}$ , and outputs  $\sigma = (h, P_\sigma)$  as a signature for  $m$ .
- **Verify**( $m, \sigma, \text{DVK}, \text{PP}$ )  $\rightarrow 1/0$ . On input a message  $m \in \mathcal{M}$ , a signature  $\sigma = (h, P_\sigma) \in \mathbb{Z}_p^* \times \mathbb{G}$ , a derived verification key DVK = (R, T<sub>vk</sub>)  $\in \mathbb{G} \times \mathbb{G}_T$ , and the system public parameters PP, the algorithm checks whether  $h \stackrel{?}{=} H_2(m, e(P_\sigma, P) \cdot (T_{vk})^h)$  holds. If it holds, the algorithm outputs 1, otherwise 0.

**Correctness.** For any message  $m \in \mathcal{M}$ , it is easy to verify that (1) VrfyKeyCheck(DVK, MPK, MSVK, PP) = 1, since  $\alpha R = \alpha r P = r A$ , and

- (2) Verify( $m, \text{Sign}(m, \text{DVK}, \text{DSK}, \text{PP}), \text{DVK}, \text{PP}$ ) = 1, since

$$\begin{aligned} e(P_\sigma, P) \cdot (T_{vk})^h &= e(h S_{sk} + x P, P) \cdot e(H_1(rP, rA), -B)^h \\ &= e(h \beta H_1(R, \alpha R), P) \cdot e(x P, P) \cdot e(H_1(rP, rA), -B)^h \\ &= e(H_1(R, \alpha R), \beta P)^h \cdot e(P, P)^x \cdot e(H_1(rP, rA), B)^{-h} \\ &= X. \end{aligned}$$

**Theorem 4.** *The PDPKS construction is secure (w.r.t. Definition 1) under the CDH assumption in the random oracle model.*

**Proof.** Similar to the security proof for the PDPKS construction in Section 4, we reduce the security of this PDPKS construction to the security of the IBS construction in [31, Section 2], using tuple (rP, rA) as the identity for the IBS construction.  $\square$

**Theorem 5.** *The PDPKS construction is master public key unlinkable (w.r.t. Definition 2) under the CDH assumption in the random oracle model. Specifically, assume that there exists an attacker  $\mathcal{A}$  that runs within time  $t$  and makes  $q_{h_i}$  queries to random oracles  $H_i (i = 1, 2)$ ,  $q_a$  queries to the verification key adding oracle, and  $q_s$  queries to the signing oracle, and wins the Game<sub>MPKUNL</sub> with advantage  $\epsilon$ , then there exists an algorithm  $\mathcal{B}$  that runs within time  $\bar{t} = t + O((q_{h_1} + q_s)\tau_{\text{mult}}) + O((q_a q_{h_1} + q_s)\tau_p)$ , and solves the CDH problem with probability at least  $\epsilon - q_a/p$ .*

13. Note that in the underlying IBS scheme,  $P_1$  appears only in the form of  $xP_1$ , here it is sufficiently secure to use only random  $x$ .

**Proof.** Similar to the privacy proof (referring to the preliminary version [1, Theorem 2]) for the PDPKS construction in Section 4, if there exists a PPT adversary  $\mathcal{A}$  that can win  $\text{Game}_{\text{MPK}_{\text{UNL}}}$  for our PDPKS construction with non-negligible advantage, then we can construct a PPT algorithm  $\mathcal{B}$  that can solve the CDH problem with non-negligible probability.

In particular, given an instance of CDH problem on bilinear groups, i.e., bilinear groups  $(\mathbb{G}, \mathbb{G}_T, e)$  of prime order  $p$ , generator  $P \in \mathbb{G}$ , and a tuple  $(\tilde{A} = aP, \tilde{B} = bP) \in \mathbb{G} \times \mathbb{G}$  for unknown  $a, b \in \mathbb{Z}_p^*$ , the target of  $\mathcal{B}$  is to compute an element  $C \in \mathbb{G}$  such that  $C = abP$ .

To simulate the PDPKS construction to  $\mathcal{A}$ ,  $\mathcal{B}$  chooses random  $\alpha'_0, \beta_0, \alpha'_1, \beta_1 \in \mathbb{Z}_p^*$ , sets  $A^{(0)} = \alpha'_0 \tilde{A}, B^{(0)} = \beta_0 P, A^{(1)} = \alpha'_1 \tilde{A}, B^{(1)} = \beta_1 P$ , and gives  $\text{MPK}_0 := (A^{(0)}, B^{(0)}), \text{MPK}_1 := (A^{(1)}, B^{(1)})$  to  $\mathcal{A}$ . Note that the secret keys corresponding to  $\text{MPK}_0$  and  $\text{MPK}_1$  are  $\text{MSVK}_0 := \alpha'_0 a, \text{MSSK}_0 := \beta_0, \text{MSVK}_1 := \alpha'_1 a, \text{MSSK}_1 := \beta_1$  respectively, where  $\mathcal{B}$  does not know the value of  $a$ .

Note that  $\mathcal{B}$  knows the values of  $\beta_0$  and  $\beta_1$ , so that it is able to answer  $\mathcal{A}$ 's queries to the  $\text{ODVKAdd}(\cdot, \cdot), \text{ODSKCorrupt}(\cdot), \text{OSign}(\cdot, \cdot)$  oracles. The challenge derived verification key is also generated in a similar way, namely,

*Challenge.* A random bit  $i^* \in \{0, 1\}$  is chosen.  $\mathcal{B}$  generates the challenge derived verification key  $\text{DVK}^* = (R^*, T_{vk}^*)$  from  $\text{MPK}_{i^*}$  as follows:

- (1) Set  $R^* = \tilde{B}$ .
- (2) Note that  $\tilde{B} = bP$  and  $T_{vk}^*$  should be  $T_{vk}^* = e(H_1(\tilde{B}, bA^{(i^*)}), -B^{(i^*)}) = e(H_1(\tilde{B}, b\alpha'_{i^*} aP), -B^{(i^*)})$ , where  $a$  and  $b$  are unknown to  $\mathcal{B}$ .  $\mathcal{B}$  chooses a random  $hval^* \in \mathbb{Z}_p^*$ , and adds  $(\tilde{B}, \top, hval^*, hval^*P)$  to  $L_{H_1}$ , where  $\top$  is a special symbol to denote the value of  $\alpha'_{i^*} abP$  that is unknown by  $\mathcal{B}$ .  $\mathcal{B}$  sets  $T_{vk}^* = e(hval^*P, -B^{(i^*)})$  and gives  $\text{DVK}^* = (R^*, T_{vk}^*)$  to  $\mathcal{A}$ .
- (3)  $\mathcal{B}$  sets  $\text{DSK}^* = (\beta_{i^*} hval^*)P$  and adds  $(\text{DVK}^*, \text{DSK}^*, 0, \tilde{B}, \top, i^*)$  to  $L_{dvk}$ . The rest of the proof and analysis can follow those in the preliminary version [1, Theorem 2].  $\square$

## 9 CONCLUSION

In this work, we introduced and formalized a new signature variant, called Key-Insulated and Privacy-Preserving Signature Scheme with Publicly Derived Public Key (PDPKS), including definition, security model, and privacy models. And we proposed a PDPKS construction, and proved its security and privacy in the random oracle model. On the functionality, anyone can derive an arbitrary number of fresh public verification keys from a user's master public key, without interactions with the key owner, while only the key owner can generate the corresponding signing keys from his master secret key. On the privacy, the derived verification keys and corresponding signatures do not leak any information that can be linked to the original master public key or other derived verification keys. On the security, the

derived keys are insulated from each other, namely, for any specific derived verification key, even if an adversary corrupts all other derived signing keys, the adversary cannot forge a valid signature with respect to it.

We implemented the proposed PDPKS construction with the parameters for 128-bit security, and the results show that it is practically efficient for the settings of cryptocurrency.

With these functionality, security, and privacy protection features, as well as the practical efficiency, our PDPKS construction could be a convenient, secure and practical cryptographic tool for building privacy-preserving cryptocurrencies, providing solid confidence due to its provable security and privacy. Particularly, the proposed PDPKS construction can be used to implement secure stealth addresses, and can be used to implement deterministic wallets and the related appealing use cases, without security concerns.

## ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant 62072305 and 61672339, in part by the National Cryptography Development Fund under Grant MMJJ20170111, in part by the National Key Research and Development Program of China under Grant 2020YFA0309705, in part by the Foundation of Science and Technology on Information Assurance Laboratory under Grant KJ-17-109, in part by the Gopalakrishnan—NTU Presidential Postdoctoral Fellowship 2018, in part by the National Research Foundation, Prime Minister's Office, Singapore, through its Strategic Capability Research Centres Funding Initiative, in part by the Singapore Ministry of Education under Grant MOE2016-T2-2-014 (S), and in part by the Abelian Foundation. A preliminary version [1] of this work appears on EuroS&P 2019.

## REFERENCES

- [1] Z. Liu, G. Yang, D. S. Wong, K. Nguyen, and H. Wang, "Key-insulated and privacy-preserving signature scheme with publicly derived public key," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2019, pp. 215–230.
- [2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: <http://bitcoin.org/bitcoin.pdf>
- [3] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [4] G. Maxwell, "Deterministic wallets," Jun. 2011. [Online]. Available: <https://bitcointalk.org/index.php?topic=19137>
- [5] Electrum.org, "Electrum lightweight bitcoin wallet," Nov. 2011. [Online]. Available: <https://electrum.org>
- [6] P. Wuille, "Bip32: Hierarchical deterministic wallets," Feb. 2012. [Online]. Available: <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>
- [7] T. Okamoto and K. Ohta, "Universal electronic cash," in *Proc. 11th Annu. Int. Cryptol. Conf. Adv. Cryptology*, 1991, pp. 324–337.
- [8] N. van Saberhagen, "Cryptonote v 2.0," 2013. [Online]. Available: <https://cryptonote.org/whitepaper.pdf>
- [9] P. Todd, "Stealth addresses," 2014. [Online]. Available: <https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-January/004020.html>

- [10] S. Noether and A. Mackenzie, "Ring confidential transactions," *Ledger*, vol. 1, pp. 1–18, 2016.
- [11] CoinMarketCap, "Top 100 cryptocurrencies by market capitalization," Accessed: Oct. 4, 2018. [Online]. Available: <https://coinmarketcap.com>
- [12] NIST, "FIPS pub 186–4," Accessed: Jun. 2, 2018. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>
- [13] G. Gutoski and D. Stebila, "Hierarchical deterministic bitcoin wallets that tolerate key leakage," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, 2015, pp. 497–504.
- [14] Y. Dodis, J. Katz, S. Xu, and M. Yung, "Key-insulated public key cryptosystems," in *Proc. Int. Conf. Theory Appl. Cryptogr. Techn., Adv. Cryptol.*, 2002, pp. 65–82.
- [15] Y. Dodis, J. Katz, S. Xu, and M. Yung, "Strong key-insulated signature schemes," in *Proc. PKC*, 2003, pp. 130–144.
- [16] J. K. Liu, V. K. Wei, and D. S. Wong, "Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract)," in *Proc. Australas. Conf. Inf. Secur. Privacy*, 2004, pp. 325–335.
- [17] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Proc. Annu. Int. Cryptol. Conf.*, 1991, pp. 129–140.
- [18] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.
- [19] V. Buterin Ed., "Deterministic wallets, their advantages, and their understated flaws," in *Bitcoin Mag.*, Nov. 2013. [Online]. Available: <http://bitcoinmagazine.com/8396/deterministic-wallets-advantages-flaw/>
- [20] N. T. Courtois and R. Mercer, "Stealth address and key management techniques in blockchain systems," in *Proc. 3rd Int. Conf. Inf. Syst. Secur. Privacy*, 2017, pp. 559–566.
- [21] D. Chaum, "Blind signatures for untraceable payments," in *Proc. Adv. Cryptology*, 1982, pp. 199–203.
- [22] D. Chaum and E. van Heyst, "Group signatures," in *Proc. Workshop Theory Appl. Cryptogr. Techn.*, 1991, pp. 257–265.
- [23] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Proc. 7th Int. Conf. Theory Appl. Cryptol. Inf. Secur.: Adv. Cryptol.*, 2001, pp. 552–565.
- [24] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, "Key-privacy in public-key encryption," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2001, pp. 566–582.
- [25] M. Backes, L. Hanzlik, K. Kluczniak, and J. Schneider, "Signatures with flexible public key: A unified approach to privacy-preserving signatures (full version)," *IACR Cryptol. ePrint Arch.*, 2018. [Online]. Available: <http://eprint.iacr.org/2018/191>
- [26] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proc. Workshop Theory Appl. Cryptographic Techn.*, 1984, pp. 47–53.
- [27] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *Proc. Annu. Int. Cryptol. Conf.*, 2001, pp. 213–229.
- [28] X. Boyen and B. Waters, "Anonymous hierarchical identity-based encryption (without random oracles)," in *Proc. Annu. Int. Cryptol. Conf.*, 2006, pp. 290–307.
- [29] S. Agrawal, D. Boneh, and X. Boyen, "Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE," in *Proc. Annu. Cryptol. Conf.*, 2010, pp. 98–115.
- [30] M. Bellare, C. Namprempre, and G. Neven, "Security proofs for identity-based identification and signature schemes," *J. Cryptol.*, vol. 22, no. 1, pp. 1–61, 2009.
- [31] F. Hess, "Efficient identity based signature schemes based on pairings," in *Proc. Efficient Identity Based Signature Schemes Based Pairings*, 2002, pp. 310–324.
- [32] J. C. Cha and J. H. Cheon, "An identity-based signature from gap Diffie-Hellman groups," in *Proc. Int. Workshop Public Key Cryptogr.*, 2003, pp. 18–30.
- [33] P. S. L. M. Barreto, B. Libert, N. McCullagh, and J. Quisquater, "Efficient and provably-secure identity-based signatures and signcryption from bilinear maps," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2005, pp. 515–532.
- [34] E. Kiltz and G. Neven, "Identity-based signatures." Accessed: Sep. 1, 2018. [Online]. Available: <http://homepage.ruhr-unibochum.de/Eike.Kiltz/papers/ibschapter.pdf>
- [35] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *J. Cryptol.*, vol. 17, no. 4, pp. 297–319, 2004.
- [36] D. Boneh and X. Boyen, "Short signatures without random oracles," in *Proc. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2004, pp. 56–73.
- [37] B. Waters, "Efficient identity-based encryption without random oracles," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2005, pp. 114–127.
- [38] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Appl. Math.*, vol. 156, no. 16, pp. 3113–3121, 2008.
- [39] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Proc. Int. Workshop Sel. Areas Cryptogr.*, 2005, pp. 319–331.
- [40] R. Barbulescu and S. Duquesne, "Updating key size estimations for pairings," *IACR Cryptol. ePrint Arch.*, 2017. [Online]. Available: <http://eprint.iacr.org/2017/334>
- [41] National Institute of Standards and Technology (NIST), "Recommendation for key management, part 1: General (revised)," Jul. 2012. [Online]. Available: <https://csrc.nist.gov/publications/detail/sp/800-57-part-1/revised/archive/2007-03-01>
- [42] T. Kim and R. Barbulescu, "Extended tower number field sieve: A new complexity for the medium prime case," in *Proc. Annu. Int. Cryptol. Conf., Part I*, 2016, pp. 543–571.
- [43] M. Shigeo, "MCL: A portable and fast pairing-based cryptography library," 2012. [Online]. Available: <https://github.com/herumi/mcl>
- [44] X. Ke, Y. Liu, and Z. Liu, "PDPKS: Key-insulated and privacy-preserving signature scheme with publicly derived public key." Accessed: Sep. 30, 2019. [Online]. Available: <https://github.com/AppliedCryptoLab/pdpks>



**Zhen Liu** received the dual PhD degrees in computer science from the City University of Hong Kong and Shanghai Jiao Tong University in 2013. He is currently an associate professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include applied cryptography, studying provable security, and designing cryptographic primitives, such as encryption and signature schemes, for the research problems motivated by practical applications.



**Guomin Yang** received the PhD degree in computer science from the City University of Hong Kong in 2009. From 2009 to 2012, he was a research scientist with Temasek Laboratories, National University of Singapore. He is currently an associate professor with the School of Computing and Information Technology, University of Wollongong, Australia. His research interests include applied cryptography and network security. He was the recipient of the prestigious ARC DECRA Fellowship in 2015.



**Duncan S. Wong** is the founder and currently the CEO of CryptoBLK, a global Blockchain technology company. He was in charge of the FinTech Initiative, Hong Kong's largest R&D center, ASTRI. He was also a professor with the City University of Hong Kong for 13 years.



**Khoa Nguyen** received the PhD degree from Nanyang Technological University (NTU) in 2014. He is currently a senior research fellow with NTU. His research interests include the areas of cryptography and information security, in particular, the designs of privacy-preserving cryptographic protocols. He was the recipient of the prestigious Gopalakrishnan - NTU Presidential Postdoctoral Fellowship in 2018.



**Xiaorong Ke** received the bachelor's degree in computer science from Shanghai Jiaotong University in 2020. He is currently a software development engineer with Huawei Technologies Company, Ltd.



**Huaxiong Wang** received the first PhD degree in mathematics from the University of Haifa, Israel, in 1996 and the second PhD degree in computer science from the University of Wollongong, Australia, in 2001. Since 2006, he has been an associate professor with the Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore. His research interests include cryptography, information security, coding theory, and theoretical computer science. He is currently the program cochair of Asiacrypt 2020 and 2021.



**Yining Liu** received the BS degree in computer science and technology from Shanghai Jiao Tong University, Shanghai, China, in 2016. She is currently working toward the MS degree in computer science with the Andrew and Erna Viterbi School of Engineering, University of Southern California.

▷ For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/csdl](http://www.computer.org/csdl).