Key-Insulated and Privacy-Preserving Signature Scheme with Publicly Derived Public Key *

Zhen Liu[1], Guomin Yang[2], Duncan S. Wong[3], Khoa Nguyen[4], and Huaxiong Wang[4]

[1] Shanghai Jiao Tong University, China.
liuzhen@sjtu.edu.cn

[2] University of Wollongong, Australia.
gyang@uow.edu.au

[3] CryptoBLK and Abelian Foundation.
duncanwong@cryptoblk.io

[4] Nanyang Technological University, Singapore.
khoantt@ntu.edu.sg, HXWang@ntu.edu.sg

Abstract. Since the introduction of Bitcoin in 2008, cryptocurrency has been undergoing a quick and explosive development. At the same time, privacy protection, one of the key merits of cryptocurrency, has attracted much attention by the community. In this paper, we identify a security vulnerability of the privacy-preserving key derivation algorithm of Monero, which is one of the most popular privacy-centric cryptocurrencies. To provide a formal treatment for the problem, we introduce and formalize a new signature variant, called Key-Insulated and Privacy-Preserving Signature Scheme with Publicly Derived Public Key (PDPKS), which forms a convenient and robust cryptographic tool for building privacy-preserving cryptocurrencies. Specifically, PDPKS allows anyone to derive new signature verification keys for a user, say Alice, based on her long-term public-key, while only Alice can derive the signing keys corresponding to those verification keys. In terms of privacy, given a derived verification key and valid signatures with respect to it, an adversary is not able to link them to the underlying long-term public key; and given two verification keys and corresponding valid signatures, an adversary cannot tell whether the verification keys are derived from the same long-term public key. A distinguishing security feature of PDPKS, with the above functionality and privacy features, is that the derived keys are independent/insulated from each other, namely, compromising the signing key associated with a verification key does not allow an adversary to forge a valid signature for another verification key, even if both verification keys are derived from the same long-term public key.

We formalize the notion of PDPKS and propose a practical and proven secure construction, which fixes the identified security vulnerability in Monero and provides a more robust solution for implementing

* This work is supported by Abelian Foundation, and forms part of the work from the Abelian Foundation. This work was inspired by our previous work [27] on the whitepaper of Abelian Coin (ABE).

the so-called stealth addresses for cryptocurrencies. Also, our PDPKS scheme can be used to fix the similar vulnerability in the deterministic wallet algorithm for Bitcoin.

Keywords: Signature Scheme, Publicly Derived Public Key, Key-Insulated Security, Privacy

1 Introduction

Since the introduction of Bitcoin [28] in 2008, in the past decade, cryptocurrency has been undergoing a quick and explosive development, with thousands of different crypto-coins available to date. 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* [33,37]. Different techniques and cryptocurrencies have been proposed to provide stronger privacy, for example, Dash, ZCash, Monero, etc. Among the existing privacy-centric cryptocurrencies, Monero is the most popular, with a market capitalization valued at approximate 2 billion USD, ranking the 10th in all the existing cryptocurrencies [16]. However, as shown later, there is a security vulnerability in Monero's core cryptographic protocol, which could cause serious damages to the security of Monero in terms of allowing attackers to steal others' Monero coins. In addition, a similar security concern on the key derivation algorithm for deterministic wallet [42] has been noticed [12,22], while to the best of our knowledge, as so far it has not been solved completely. In this work, we formalize a new cryptographic concept and propose a provably secure construction, which provide a practical and solid solution to address these problems.

1.1 Preliminaries of Cryptocurrency

Bitcoin-like cryptocurrencies, for example Monero [32], maintain a ledger consisting of a series of transactions, and Digital Signature [35] is 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 of a coin cn with public key pk wants to spend the coin (i.e. act 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*. In Bitcoin, while the transactions, as well as the coins, are related to only the public keys and there are no bindings between the public keys and user identities, user privacy is protected in the sense that *"the public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone"*[28]. However, such a privacy-preserving mechanism (i.e., pseudonymity) is pretty weak, for example, if a participant uses one public key to receive multiple coins from multiple transactions, then the public can link these transactions to a common owner.

1.2 A Vulnerability in Monero's Cryptographic Protocol

To achieve unlinkable payments, Monero adopts a one-time derived public key mechanism proposed in CryptoNote [37], where the receiving address of each coin (by default) is a fresh public key derived from payee's long-term public key and payer's random data. As pointed out in [37], the main advantage of such a derived public key solution 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 or link two coin-receiving addresses (as well as the corresponding coins and transactions) together. And importantly, this is achieved in a very convenient manner, as each participant only needs to publish one long-term public key, and anyone (acting as a payer) can generate an arbitrary number of fresh public keys from the long-term public key of a participant (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 keys without any interaction with the payer. In particular, as shown in Fig. 1, each participant publishes his long-term public key (A, B), serving as a pseudonym without revealing his real identity in practice, and keeps corresponding long-term secret key (a, b) safe, where $a, b \in \mathbb{Z}_p$ are standard elliptic curve private keys and $A = aG, B = bG$ are standard elliptic curve public keys, with G being a base point. For each transaction, the payer chooses a random $r \in \mathbb{Z}_p$ and computes

a derived public/verification key[5] $dvk = (R = rG, S = H(rA)G + B)$ from the payee's long-term public key (A, B), where $H : \{0,1\}^* \to \mathbb{Z}_p$ is a cryptographic hash function, 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 receiver 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[6]. On 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 [17] 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, and particularly, the payer cannot spend the coin either, since he does not know the value of b. This is why B is added in S. Actually, due to its virtues in functionality, privacy and "security", the above algorithm has also been adopted by the cryptocurrency community to achieve *stealth addresses* [40].

However, as shown below, the above public key derivation algorithm, which Monero is using as its core protocol, suffers a security vulnerability, from both theory and practice point of view. From the theory point of view, the derived (public/verification key, secret/signing key) pairs should be insulated from each other, namely, if one derived secret/signing key was compromised, the security of other derived keys should not be affected. However, in the above algorithm for Monero, if a payer transfers multiple coins to the same payee, each with a derived fresh public/verification key, and subsequently compromises one of them by obtaining the corresponding derived secret/signing key, then the payer will be able to compute all the derived

[5] 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.

[6] Besides using the above one-time public key derivation 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 [26] and Pedersen Commitment [34] respectively. But all these functionalities are built on the basis of the above public key derivation algorithm, as the derived key pair (S, s) serves as the coin-receiving address and coin-spending key.

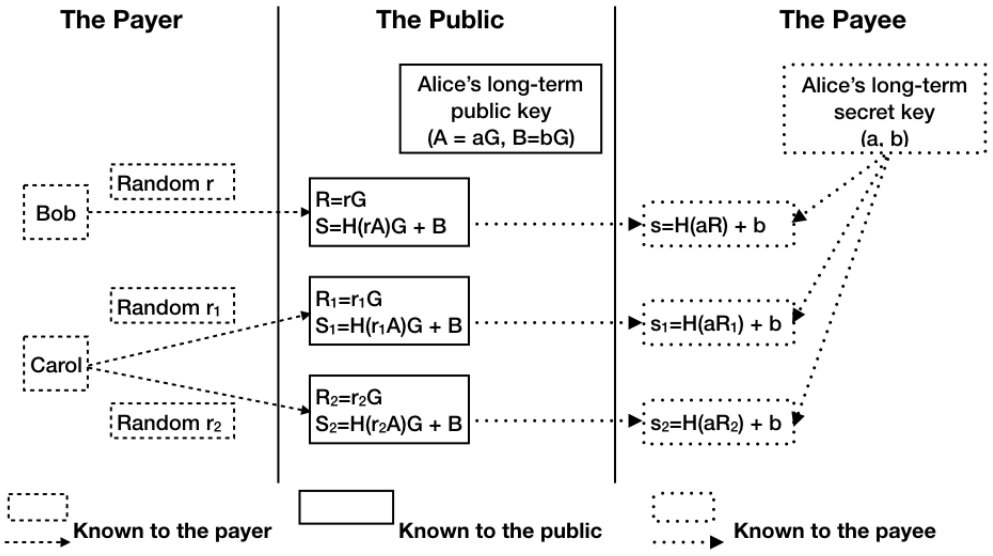

Fig. 1. Monero Public Key Derivation Algorithm.

secret/signing keys for the coins he sent to the payee. In particular, consider the example in Fig. 1, namely, the payer Carol derives two public/verification keys $dvk_1 = (R_1 = r_1G, S_1 = H(r_1A)G+B)$ and $dvk_2 = (R_2 = r_2G, S_2 = H(r_2A)G + B)$ for the same payee Alice with long-term public key (A, B). Suppose Carol corrupts one of the two secret/signing keys, say $s_1 = H(aR_1) + b$. Note that Carol knows the value of r_1, so that 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 dvk_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 corrupts all the secret/signing keys for the related coins, for example, colluding with Bob in Fig. 1, 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 public key is not safe any more, and all coins to the fresh public/verification keys derived from this long-term public key in the past and the future are in danger of being stolen.

From the practice point of view, the participants in Monero 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. Note that the latter is an additional requirement, but the participants in Monero are not warned about this, and whether the latter is achieved depends on the implementations and the participants' awareness and behavior, rather than the built-in protection

provided by cryptographic techniques. Actually, in practice keeping all the derived secret/signing keys (i.e. coin-spending keys) safe is a difficult task. Note that, as pointed out by Dodis et al. [18,19], which motivated the key-insulated cryptography, *"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 the setting of Monero, to avoid the leakage of the derived signing keys, the users and the wallet software may be very careful, namely, only when needing to generate a signature to spend a coin, the corresponding signing key is derived/generated, and once the signature is generated, the signing key is erased. However, such careful behaviors and awareness may fail in some situations. Imagine a user's phone has a Monero wallet and accidentally failed to erase all the derived secret/signing keys. If there is one yet-to-be-erased derived secret/signing key being stolen, for example, via a malware attack against the user's Android phone, the attacker is able to compromise the user's long-term secret key as shown above. For the conventional key management mechanism, where each (public key, secret key) pair is generated independently, in such a situation, only the coin on the corresponding public key may be affected. On the contrast, for Monero, such a situation could cause much more serious damages, as the attacker could derive all the secret/signing keys for the same long-term public key. In conclusion, we need a cryptographic solution to provide built-in protection, so that the security and convenience could be achieved simultaneously for the privacy-preserving techniques of the key-derivation algorithm.

1.3 A Vulnerability in Deterministic Wallet for Bitcoin

While the public key derivation algorithm in Fig. 1 is the core protocol for Monero, a similar algorithm, shown in Fig. 2, is used as the key generation/derivation algorithm for Deterministic Wallet [29, Chapter 4.2], which is regarded as an effective solution for Bitcoin to manage the cold storage[7] addresses conveniently and to simultaneously achieve the privacy of receiving each coin at a fresh cold address. In particular, the cold side generates and holds *secret key generation information* (k, y), while sending the corresponding *address generation information* $(k, Y = yG)$ to the hot side. Thus, the hot side generates a new public $pk_i = Y + H(k\|i)G$ and corresponding address $H'(pk_i)$

[7] "Hot storage" is convenient but somewhat risky, while "cold storage" is offline, so that it's safer and more secure, but not as convenient. Having hot storage and cold storage separate is also motivated by the concerns on the secrecy of the coin-spending keys. More details are referred to [29, Chapter 4.2].

sequentially each time it wants to send coins to the cold side, where H and H' are cryptographic hash functions. When the cold side reconnects, it generates addresses sequentially and checks the block chain for transfers to those addresses until it reaches an address that hasn't received any coins. If the cold side wants to send some coins, say on i-th address, back to the hot side or spend them some other way, it can generate the corresponding secret key $sk_i = y + H(k\|i)$. Note that this key derivation algorithm works well due to the fact that for Bitcoin's underlying signature scheme, namely ECDSA (Elliptic Curve Digital Signature Algorithm) [31], $pk_i = Y + H(k\|i)G$ and $sk_i = y + H(k\|i)$ form a valid (public key, secret key) pair as $Y = yG$. The advantage of this key derivation algorithm is that it provides a convenient way to manage the cold addresses with the privacy of receiving each coin at a fresh cold address. The convenience lies in that the algorithm allows the cold side to use an essentially unbounded number of addresses and the hot side to know about these addresses, but *with only a short, one-time communication between the two sides*. Actually, a specification of Hierarchical Deterministic wallets based on this mechanism was proposed in 2012 and subsequently accepted as Bitcoin standard BIP32 [42]. However, similar to the key derivation algorithm of Monero, the convenience comes at the price that the derived keys are not insulated from each other any more. As a result, the key derivation algorithm for Bitcoin's Deterministic Wallet suffers the same security vulnerability, namely, if an attacker compromises a hot storage, i.e. obtaining the address generation information (k, Y), and compromises one secret key, say i-th secret key $x_i = y + H(k\|i)$ and the value of i, then he can compute the value of y by $y \leftarrow x_i - H(k\|i)$, and subsequently can compute all the secret keys for the addresses generated from (k, Y).

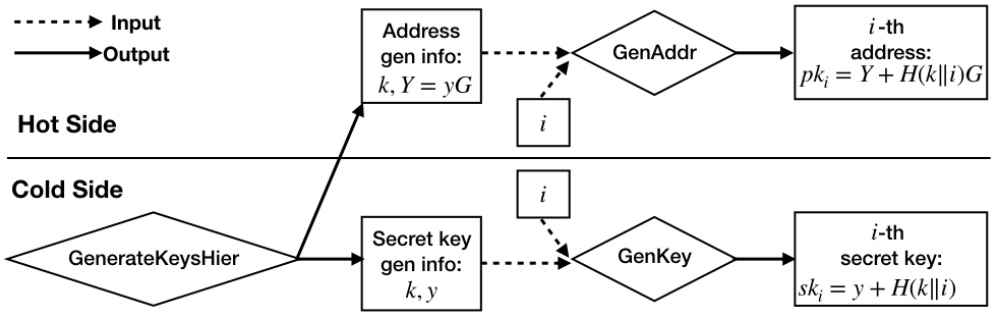

Fig. 2. Schema of a deterministic wallet [29, Chapter 4.2].

1.4 Related Work

For the above vulnerability in Monero, it is essentially the same as that in the deterministic wallet for Bitcoin, although it additionally uses the techniques of Diffie-Hellman Key Exchange, which enables it to provide more powerful functionality, privacy and convenience. It is somewhat surprising that this vulnerability in Monero has not been noticed in the community, while the above vulnerability in deterministic wallet for Bitcoin has been noticed. In particular, Buterin [12] called attention to this vulnerability in deterministic wallet, announcing open-source software that cracks BIP32 [42] and Electrum wallets [20], but was pessimistic on fixing this vulnerability, *"can this (vulnerability) be fixed? The answer seems to be no; ... If this is indeed true, then raising awareness is the only solution, ..."*[12]. As an attempt to fix this vulnerability, Gutoski and Stebila [22] proposed a deterministic wallet that can tolerate the leakage of up to m derived secret/signing keys with a 'master public key' (i.e. address generation information) size of $O(m)$. More specifically, the master public key is $(Y_1 = y_1G, \ \ldots, \ Y_m = y_mG)$ while the corresponding 'master private key' (i.e. key generation information) is $(y_1, \ \ldots, \ y_m)$. The i-th derived public key and secret key are $pk_i \leftarrow \sum_{j=1}^{m} \alpha_i Y_i$ and $sk_i \leftarrow \sum_{j=1}^{m} \alpha_i y_i$ respectively, where $(\alpha_1, \ldots, \alpha_m) \leftarrow H(Y_1, \ldots, Y_m, i)$ could be generated by a hash fucntion H. Obviously, once an attacker obtains m derived secret keys somehow, it can compute the master secret key $(y_1, \ \ldots, \ y_m)$ by solving the m equations with m variables. Besides the limitation on the number of compromised secret keys and the overhead of $O(m)$, it is worth mentioning that the work in [22] does not consider privacy protection. Actually, given a derived public key and a set of potential master public keys, it is easy to find the one from which the derived public key is generated from, since an attacker can try the i's and compute the corresponding $(\alpha_1, \ldots, \alpha_m)$'s efficiently.

Note that in our previous work [27], we considered this problem, and schemes satisfying the security models in our work at [27] would not have the similar vulnerabilities to that in the core protocol of Monero or the deterministic wallets for Bitcoin.

1.5 Our Results

In this paper, motivated by the vulnerabilities in the core protocol of Monero and the deterministic wallets for Bitcoin, we introduce and formalize a new signature variant, called Key-Insulated and

Privacy-Preserving Signature Scheme with Publicly Derived Public Key (PDPKS)[8], and propose a provably secure and practically efficient construction, so that we fix the vulnerabilities completely by a cryptographic scheme.

In particular, on the functionality, PDPKS provides a convenient way to enable receiving each coin at a fresh/unique address, namely, anyone can derive public/verification keys from a long-term public key without requiring any interaction, while only the owner of the long-term public key can generate the corresponding secret/signing keys, also without interactions. On the security, we formalize a security model for PDPKS, which ensures the derived keys are completely independent/insulated from each other, i.e., for any specific derived public/verification key dvk, even if an adversary corrupts all other derived public and secret keys from the same long-term key, the adversary cannot forge a valid signature with respect to dvk. On the privacy, we formalize two privacy models for PDPKS, both of which captures practical needs and requirements, namely, one capturing that an adversary should not be able to link a given public/verification key (with corresponding signatures) to its underlying long-term public key, and the other capturing that an adversary should not be able to tell whether two public/verification keys (with their corresponding signatures) are derived from the same long-term public key. And we prove that the latter is implied by the former and hence we only need to focus on one privacy model.

With its functionality, security, and privacy-preserving features, PDPKS is especially applicable to cryptocurrency, where the payer of a transaction specifies a public/verification key as the address of a coin for the intended payee (i.e. the owner of the coin). As highlighted above, independence among multiple coin-spending keys and privacy-protection are crucial concerns in this scenario. To demonstrate these properties are achievable, we propose a practical PDPKS construction, and prove its security and privacy in the random oracle model.

1.6 Related Techniques and Our Approach

1.6.1 Techniques Related to Privacy-protection

While Blind Signature [14] hides the really signed messages from the signer and Group Signature [15] and Ring Signature [36] hide the identity of the real signer from the public, the PDPKS sig-

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

nature in this paper 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, as well as the link among the derived public/verification keys (and corresponding signatures) from the same 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 [32], a variant of ring signature, namely Linkable Ring Signature [26], has been adopted to hide the payer, while the above discussed public/verification key derivation mechanism is used to hide the payee.

While the privacy-preserving concerns in cryptocurrencies motivate us to investigate the (public) key-privacy problem for digital signature in this paper, Bellare et al. [5] 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 receiver is anonymous from the view point of the adversary.[9] It is worth mentioning that the key-private encryption scheme in [5] has been used by Zerocash [7] (in 2014) as one of the cryptographic components to enhance privacy. Note that Zerocash [7] enhanced privacy in cryptocurrency by zero-knowledge proof techniques, rather than by the signature variants as in Monero.

Recently, a new notion named "Signatures with Flexible Public Key" was proposed in [1]. 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 also gives a way to implement the stealth addresses for cryptocurrencies. Nevertheless, it suffers the same security issue as in Monero and the Deterministic Wallet for Bitcoin illustrated above.

1.6.2 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. [18,19] 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

[9] We borrow the term "key-privacy" from [5], although its meaning for digital signature in this paper is very different from that for public key encryption in [5].

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 key-insulated signature scheme does not consider the privacy-preserving 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. We borrow the term "key-insulated" in the sense that the derived signing keys are completely independent from each other and the security of any specific derived signing key will not be affected even if all other derived signing keys are corrupted.

In Identity-based Cryptography [38,9], 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 ciphertext encrypted under $(\mathsf{MPK}, \mathsf{ID})$ as public key (in Identity-based Encryption (IBE) system) or sign a message to produce a signature that can be verified by $(\mathsf{MPK}, \mathsf{ID})$ 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 independent/insulated from each other. On privacy, user/identity anonymity *inside a system* has been studied. In particular, in an anonymous IBE [11], the attackers can not distinguish between $C_0 \leftarrow Enc(\mathsf{MPK}, \mathsf{ID}_0, M)$ and $C_1 \leftarrow Enc(\mathsf{MPK}, \mathsf{ID}_1, M)$ for any message M and identities $\mathsf{ID}_0 \neq \mathsf{ID}_1$, unless it has a secret key for ID_0 or ID_1. However, *master public key privacy among multiple systems* has not been considered as fo far. In particular, consider two instantiations of an IBE scheme, with master public keys MPK_0 and MPK_1 respectively. The master-public-key privacy requires that an attacker should be unable to distinguish between $C_0 \leftarrow Enc(\mathsf{MPK}_0, \mathsf{ID}_0, M)$ and $C_1 \leftarrow Enc(\mathsf{MPK}_1, \mathsf{ID}_1, M)$ for any message M, and identities $(\mathsf{ID}_0, \mathsf{ID}_1)$. This is somewhat similar to the public key encryption with key privacy by Bellare et al. [5], but seems to be less motivated, which may be the reason why it has been considered yet. The master-public-key privacy for IBS may be more complicated than that in IBE, since the the master public key and the identity need to be known by the public who verify the signature. Also, IBS with master-public-key privacy seems to lack of motivation and has not been considered as fo far.

1.6.3 Our Construction Approach

Besides introducing and formalizing PDPKS, including its definition and models for security and privacy, we also present a construction approach in this work, as well as a concrete construction with provable security and privacy. Below we briefly present 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) public key, and the corresponding secret/signing key can be computed from the verification key and the long-term 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 public key, neither to those from the same long-term public key. For the requirements (1) and (2), it is natural to consider the Identity-Based Signature (IBS) [38,9,6], 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).

Note that the key-escrow problem in IBS, i.e. PKG can generate and know the secret key sk_{ID} for any identity ID, is unacceptable in the setting of cryptocurrencies, we could not apply anonymous IBS in cryptocurrencies to address the privacy problem. Instead, to construct a PDPKS, we start from an IBS scheme in a trick, which is simple but effective, and matches the cryptocurrency setting well, as below:

- Each participant, say P_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 public key of PDPKS, and holds the master secret key as his long-term secret key, denoted by MPK_i and MSK_i respectively.
- When issuing a transaction with P_i as the payee, the payer creates a random string (i.e. identity) ID and sets $vk = (\mathsf{MPK}_i, \mathsf{ID})$ 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 $vk = (\mathsf{MPK}_i, \mathsf{ID})$, the intended payee can run the IBS' Key Extract algorithm $sk_{vk} \leftarrow \mathsf{IBS.KeyExtract}(\mathsf{MPK}_i, \mathsf{ID}, \mathsf{MSK}_i)$ and set sk_{vk} as the secret/signing key, and then spend the coin by generating a valid signature σ, which can be verified by the IBS' Verify algorithm $\mathsf{IBS.Verify}(\mathsf{MPK}_i, \mathsf{ID}, M, \sigma)$, where M is the signed message.

Note that using IBS in such a way does not suffer the key-escrow problem any more, since each participant 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 $vk = (\mathsf{MPK}_i, \mathsf{ID})$ contains the corresponding long-term public key MPK_i. To provide privacy required by PDPKS, the verification algorithm should takes only the verification key vk, the message, and the signature σ as inputs, and vk and σ should not leak any information about the corresponding MPK. Note that such a privacy requirement is just what we discussed previously in Sec. 1.6.2, namely, IBS with master-public-key privacy. However, it seems that due to its lack of motivation, IBS with master-public-key privacy has not been considered or researched as fo far. In this work, motivated by the practical vulnerabilities in Monero and Bitcoin wallet, we focus on the formalization and construction of PDPKS, rather than IBS with master-public-key privacy. To construct a PDPKS from IBS scheme using 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 Verify_F such that
  1. An attacker, who does know the value of ID, cannot learn any partial information about IMPK from the value of $F(\mathsf{MPK}, \mathsf{ID})$, where ID is a random string.
  2. The signature does not leak leak any partial information about IMPK.
  3. For any master public key MPK, any random ID, any message M, and any signature σ, it holds that $\mathsf{Verify}_F(\mathsf{CMPK}, F(\mathsf{MPK}, \mathsf{ID}), M, \sigma) = \mathsf{IBS.Verify}(\mathsf{MPK}, \mathsf{ID}, M, \sigma)$.

Intuitively, with such an IBS scheme, we can generate ID using Diffie-Hellman Key Exchange Protocol to prevent the attacker from knowing the value of ID, and set $vk = (R, F(\mathsf{MPK}, \mathsf{ID}))$ where $R = rG$ is the randomness to run the Diffie-Hellman protocol, 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 above approach, we investigated three existing IBS schemes [23,13,3], which have very different construction structures. Finally, we find that the IBS schemes in [23,3] have the above MPK-pack-able property, while the IBS scheme in [13] does not have. This is not surprising, as the master-public-key privacy has not been considered in IBS. Based on the IBS schemes in [23,3], we construct two PDPKS schemes formally, and prove their security and privacy in the random oracle model. Roughly speaking, on the construction, inspired by the algorithm in Monero, we generate the identity using Diffe-Hellman Key Exchange Protocol. On the proof, implied by the above approach, the security proofs are comparatively easy, by a reduction to the security of 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 Monero. All these techniques are to enable the proof of privacy.

We would like to point out that the above approach of transferring an IBS scheme to a PDPKS scheme is not the unique way to construct PDPKS schemes. Also, we would like to point out that the PDPKS concept formalized in this work is well motivated by the practical requirements in cryptocurrencies, and PDPKS may be a meaningful motivation to the research on IBS with master-public-key privacy, while the ideas and techniques in IBC could be useful tools for constructing PDPKS, but we do not want to limit the construction of PDPKS to being from IBS. That is why we formalize the concept of PDPKS, rather than extending the IBS concept.

1.7 Outline

In Sec. 2, we formalize the definition, the security model, and the privacy-preserving model for PDPKS. In Sec. 3 we propose a PDPKS construction, and prove its security and privacy in Sec. 4. In Sec. 5 we discuss the application and implementation of the proposed PDPKS construction. The paper is concluded in Sec. 6. In Apendix B, we give another PDPKS construction and the outlines for the proofs of security and privacy. In Apendix C, we show that the IBS scheme in [13] does not have the MPK-pack-able property.

2 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 particular, we first formalize the comprising algorithms and the security model, which capture the special functionality that fresh public/verification keys could be derived from a long-term public key, and the security requirement that the derived (public/verification key, secret/signing key) pairs are insulated from each other so that one being compromised will not affect others. Then we formalize two models for privacy, both of which reflect practical privacy concerns. Specifically, the first model captures that an adversary, given the fresh secret/signing key corruption oracle and signing oracle, should not be able to link a fresh public/verification key to its original long-term public key out of a set of known long-term public keys; and the second captures that an adversary should not be able to tell whether two fresh public/verification keys are derived from the same long-term public key. We prove that the privacy in the second model is implied by that of the first, so that we can focus on the privacy in the first model.

Note that the concept of PDPKS is motivated by the security and privacy problems in cryptocurrency, where it is suggested that each public/verification key, as the coin address, is used only once. But in this paper we do not restrict the concept to one-time signature scheme, which requires that for each public key the signing oracle can be queried at most once. Our proposed PDPKS requires stronger security, namely, even if the users use the freshly derived key pairs multiple times, the system is still safe.

2.1 Algorithm Definition

A Key-Insulated and Privacy-Preserving Signature Scheme with Publicly Derived Public Key (PDPKS) consists of following algorithms:

- $\mathsf{Setup}(\lambda) \to \mathsf{PP}$. The algorithm takes as input a security parameter λ, runs in polynomial time in λ, and outputs system public parameters PP.

  The system public parameters PP *are common parameters used by all participants in the system, including the underlying groups, hash functions, etc.*

– $\mathsf{KeyGen}(\mathsf{PP}) \rightarrow (\mathsf{PK}, \mathsf{SK})$. The algorithm takes as input the system public parameters PP, and outputs a (public key, secret key) pair $(\mathsf{PK}, \mathsf{SK})$.

Each participant runs KeyGen *algorithm to generate his long-term (public key, secret key) pair.*

– $\mathsf{VrfyKeyDerive}(\mathsf{PK}, \mathsf{PP}) \rightarrow \mathsf{DVK}$. The algorithm takes as input a public key PK and the system public parameters PP, and outputs a derived verification key DVK.[10]

Anyone can run this algorithm to generate a fresh public/verification key from a long-term public key.

– $\mathsf{VrfyKeyCheck}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP}) \rightarrow 1/0$. The algorithm takes as input a derived verification key DVK, a (public key, secret key) pair $(\mathsf{PK}, \mathsf{SK})$, and the system public parameters PP, and outputs a bit $b \in \{0, 1\}$, with $b = 1$ meaning that DVK is a valid derived verification key generated from PK and $b = 0$ otherwise.

The owner of a long-term (public key, secret key) pair can use this algorithm to check whether a verification key is derived from his public key. In a cryptocurrency, a payee can use this algorithm to check whether he is the intended receiver of a coin on the verification key. Note that this algorithm is actually a subroutine of the following SignKeyDerive *algorithm. It is put here as a standalone algorithm to capture the application scenario that, in a cryptocurrency, when a payer issues a transaction paying to a payee, the payee may first check whether he is the owner of the output coin's verification key to ensure he is paid well, but does not compute the corresponding signing key at this moment. The signing key may be computed just before it is used to sign a transaction to spend the coin.*

– $\mathsf{SignKeyDerive}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP}) \rightarrow \mathsf{DSK}$ or $\perp$. The algorithm takes as input a derived verification key DVK, a (public key, secret key) pair $(\mathsf{PK}, \mathsf{SK})$, and the system public parameters PP, and outputs a derived signing key DSK, or $\perp$ implying that DVK is not a valid verification key derived from PK.

The owner of a long-term (public key, secret key) pair 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 public key.

[10] 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.

– $\mathsf{Sign}(m, \mathsf{DVK}, \mathsf{DSK}, \mathsf{PP}) \to \sigma$. The algorithm takes as input a message m in message space $\mathcal{M}$, a derived (verification key, signing key) pair $(\mathsf{DVK}, \mathsf{DSK})$, and the system public parameters PP, and outputs a signature σ.
– $\mathsf{Verify}(m, \sigma, \mathsf{DVK}, \mathsf{PP}) \to 1/0$. The algorithm takes as input a (message, signature) pair (m, σ), a derived verification key DVK, and the system public parameters PP, and 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

$$\mathsf{PP} \leftarrow \mathsf{Setup}(\lambda), (\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\mathsf{PP}),$$
$$\mathsf{DVK} \leftarrow \mathsf{VrfyKeyDerive}(\mathsf{PK}, \mathsf{PP}), \quad \mathsf{DSK} \leftarrow \mathsf{SignKeyDerive}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP}),$$

it holds that

$$\mathsf{VrfyKeyCheck}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP}) = 1 \textit{ and}$$
$$\mathsf{Verify}(m, \mathsf{Sign}(m, \mathsf{DVK}, \mathsf{DSK}, \mathsf{PP}), \mathsf{DVK}, \mathsf{PP}) = 1.$$

2.2 Security Model

The security of a PDPKS scheme is defined as below.

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*** $\mathsf{Game}_{\mathsf{UEF}}$ *is negligible.*

– **Setup.** $\mathsf{PP} \leftarrow \mathsf{Setup}(\lambda)$ *is run and* PP *are given to* $\mathcal{A}$.
$(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\mathsf{PP})$ *is run and* PK *is given to* $\mathcal{A}$. *An empty set* $L_{dvk} = \emptyset$ *is initialized.* [11]
– **Probing Phase.** $\mathcal{A}$ *can adaptively query the following oracles:*
  • *Verification Key Adding Oracle* $\mathsf{ODVKAdd}(\cdot)$*:*
  Upon input a derived verification key DVK*, this oracle returns* $b \leftarrow \mathsf{VrfyKeyCheck}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP})$ *to* $\mathcal{A}$. *If* $b = 1$*, set* $L_{dvk} = L_{dvk} \cup \{\mathsf{DVK}\}$.
  This captures that $\mathcal{A}$ *can try and test whether the derived verification keys generated by him are accepted by the owner of* PK.

[11] This list is defined only for describing the game easier.

- *Signing Key Corruption Oracle* $\mathsf{ODSKCorrput}(\cdot)$*:*

  Upon input a derived verification key DVK *which is in* L_{dvk}*, this oracle returns* $\mathsf{DSK} \leftarrow \mathsf{SignKeyDerive}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP})$ *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* $\mathsf{OSign}(\cdot, \cdot)$*: Upon input a message* $m \in \mathcal{M}$ *and a derived verification key* $\mathsf{DVK} \in L_{dvk}$*, this oracle returns* $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{DVK}, \mathsf{DSK}, \mathsf{PP})$ *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.*

– **Output Phase.** $\mathcal{A}$ *outputs a message* $m^* \in \mathcal{M}$*, a derived verification key* $\mathsf{DVK}^* \in L_{dvk}$*, and a signature* σ^**.* $\mathcal{A}$ *succeeds in the game if* $\mathsf{Verify}(m^*, \sigma^*, \mathsf{DVK}^*, \mathsf{PP}) = 1$ *under the* **restriction** *that (1)* $\mathsf{ODSKCorrput}(\mathsf{DVK}^*)$ *is never queried, and (2)* $\mathsf{OSign}(m^*, \mathsf{DVK}^*)$ *is never queried.*

Remark: **Note that the adversary in the above model is allowed to generate derived verification keys and corrupt the corresponding signing keys on 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 public key are corrupted, the specific one is still safe. With such a security requirement, the security flaws in Monero's protocol and Bitcoin's deterministic wallet are avoided.**

2.3 Privacy Models

The public key privacy of a PDPKS scheme needs to consider two cases:

– **Case I**: Given a derived verification key, an adversary should not be able to tell which public key, out of a set of known public keys, is the one from which the verification key was derived.

– **Case II**: Given two derived verification keys, an adversary should not be able to tell whether they are generated from the same public key.

Below we define the two types of key privacy, and prove that we only need to consider Case I.

Definition 2. *A PDPKS scheme is public key unlinkable (PK-UNL), if for all PPT adversaries* $\mathcal{A}$*, the advantage of* $\mathcal{A}$ *in the following game* $\mathsf{Game}_{\mathsf{PKUNL}}$*, denoted by* $Adv_{\mathcal{A}}^{pkunl}$*, is negligible.*

– **Setup.** $\mathsf{PP} \leftarrow \mathsf{Setup}(\lambda)$ *is run and* PP *are given to* $\mathcal{A}$.
$(\mathsf{PK}_0, \mathsf{SK}_0) \leftarrow \mathsf{KeyGen}(\mathsf{PP})$ *and* $(\mathsf{PK}_1, \mathsf{SK}_1) \leftarrow \mathsf{KeyGen}(\mathsf{PP})$ *are run, and* $\mathsf{PK}_0, \mathsf{PK}_1$ *are given to* $\mathcal{A}$. *An empty set* $L_{dvk} = \emptyset$ *is initialized.* [12]

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

- *Verification Key Adding Oracle* $\mathsf{ODVKAdd}(\cdot, \cdot)$*:*
Upon input a derived verification key DVK *and a public key* $\mathsf{PK} \in \{\mathsf{PK}_0, \mathsf{PK}_1\}$*, this oracle returns* $b \leftarrow \mathsf{VrfyKeyCheck}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP})$ *to* $\mathcal{A}$*, where* SK *is the secret key corresponding to* PK. *If* $b = 1$*, set* $L_{dvk} = L_{dvk} \cup \{(\mathsf{DVK}, \mathsf{PK})\}$.
This captures that $\mathcal{A}$ *can try and test whether the derived verification keys generated by him are accepted by the owner of* PK.
- *Signing Key Corruption Oracle* $\mathsf{ODSKCorrput}(\cdot)$*:*
Upon input a derived verification key DVK *which is in* L_{dvk}*, this oracle returns* $\mathsf{DSK} \leftarrow \mathsf{SignKeyDerive}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP})$ *to* $\mathcal{A}$*, where* PK *is the public key that* DVK *is derived from, and* SK *is the secret key corresponding to* PK.
This captures that $\mathcal{A}$ *can obtain the derived signing keys for some existing valid derived verification keys of its choice.*
- *Signing Oracle* $\mathsf{OSign}(\cdot, \cdot)$*: Upon input a message* $m \in \mathcal{M}$ *and a derived verification key* DVK *in* L_{dvk}*, this oracle returns* $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{DVK}, \mathsf{DSK}, \mathsf{PP})$ *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.*

– **Challenge.** *A random bit* $b \in \{0, 1\}$ *is chosen,* $\mathsf{DVK}^* \leftarrow \mathsf{VrfyKeyDerive}(\mathsf{PK}_b)$ *is given to* $\mathcal{A}$. *Set* $L_{dvk} = L_{dvk} \cup \{(\mathsf{DVK}^*, \mathsf{PK}_b)\}$.

– **Phase 2**. *Same as* **Phase 1**, *except that*
(1) $\mathsf{ODVKAdd}(\mathsf{DVK}^*, \mathsf{PK}_i)$ *(for* $i \in \{0, 1\}$*) cannot be queried; and (2)* $\mathsf{ODSKCorrput}(\mathsf{DVK}^*)$ *cannot be queried.*

– **Guess**. $\mathcal{A}$ *outputs a bit* $b' \in \{0, 1\}$ *as its guess to* b.
$\mathcal{A}$ *succeeds in the the game if* $b = b'$. *The advantage of* $\mathcal{A}$ *is* $Adv_{\mathcal{A}}^{pkunl} = |\Pr[b' = b] - \frac{1}{2}|$.

Remark: **Note that the adversary in the above model is allowed to query** $\mathsf{OSign}(\cdot, \mathsf{DVK}^*)$**. This captures the privacy-preserving requirement in cryptocurrency that even after**

[12] The list is defined only for describing the game easier.

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 long-term public key.

Definition 3. *A PDPKS scheme is public key* ***strongly*** *unlinkable (PK-S-UNL), if for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following game* $\mathsf{Game}_{\mathsf{PKSUNL}}$*, denoted by* $Adv_{\mathcal{A}}^{pksunl}$*, is negligible.*

$\mathsf{Game}_{\mathsf{PKSUNL}}$*: Same as* $\mathsf{Game}_{\mathsf{PKUNL}}$*, except that in* **Phase 2**, *the restriction "(2)* $\mathsf{ODSKCorrput}(\mathsf{DVK}^*)$ *cannot be queried" is removed.*

Remark: In the game $\mathsf{Game}_{\mathsf{PKSUNL}}$, without the restriction "(2) $\mathsf{ODSKCorrput}(\mathsf{DVK}^*)$ cannot be queried", it is implied that, even given the derived signing key corresponding to the challenge derived verification key, an adversary cannot tell which public key the verification key is derived from. This implies stronger privacy.

Below we define the key privacy for the **Case II**.

Definition 4. *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* $\mathsf{Game}_{\mathsf{DVKUNL}}$*, denoted by* $Adv_{\mathcal{A}}^{dvkunl}$*, is negligible.*

- **Setup.** *Same as that of* $\mathsf{Game}_{\mathsf{PKUNL}}$.
- **Phase 1.** *Same as that of* $\mathsf{Game}_{\mathsf{PKUNL}}$.
- **Challenge.** *A random bit $b \in \{0,1\}$ is chosen, and a random bit $c \in \{0,1\}$ is chosen. Compute* $\mathsf{DVK}_0^* \leftarrow \mathsf{VrfyKeyDerive}(\mathsf{PK}_c, \mathsf{PP})$. *If $b = 0$, compute* $\mathsf{DVK}_1^* \leftarrow \mathsf{VrfyKeyDerive}(\mathsf{PK}_c, \mathsf{PP})$, *otherwise compute* $\mathsf{DVK}_1^* \leftarrow \mathsf{VrfyKeyDerive}(\mathsf{PK}_{1-c}, \mathsf{PP})$. $(\mathsf{DVK}_0^*, \mathsf{DVK}_1^*)$ *is given to $\mathcal{A}$. Set* $L_{dvk} = L_{dvk} \cup \{(\mathsf{DVK}_0^*, \mathsf{PK}_c), (\mathsf{DVK}_1^*, \mathsf{PK}^*)\}$, *where* $\mathsf{PK}^* = \mathsf{PK}_c$ *if $b = 0$,* $\mathsf{PK}^* = \mathsf{PK}_{1-c}$ *otherwise.*
- **Phase 2**. *Same as* **Phase 1**, *except that*
  (1) $\mathsf{ODVKAdd}(\mathsf{DVK}_j^*, \mathsf{PK}_i)$ *(for $j, i \in \{0,1\}$) cannot be queried; and (2)* $\mathsf{ODSKCorrput}(\mathsf{DVK}_j^*)$ *(for $j \in \{0,1\}$) cannot be queried.*
- **Guess**. *$\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ as its guess to b, i.e., guess whether* DVK_0^* *and* DVK_1^* *are derived from the same public key.*
  $\mathcal{A}$ succeeds in the the game if $b = b'$. The advantage of $\mathcal{A}$ is $Adv_{\mathcal{A}}^{dvkunl} = |\Pr[b' = b] - \frac{1}{2}|$.

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

Definition 5. *A PDPKS scheme is derived verification key **strongly** unlinkable (DVK-S-UNL), if for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following game $\mathsf{Game}_{\mathsf{DVKSUNL}}$, denoted by $Adv_{\mathcal{A}}^{dvksunl}$, is negligible.*
$\mathsf{Game}_{\mathsf{DVKSUNL}}$*: Same as* $\mathsf{Game}_{\mathsf{DVKUNL}}$*, except that in* **Phase 2***, the restriction "(2)* $\mathsf{ODSKCorrput}(\mathsf{DVK}_j^*)$ *(for $j \in \{0,1\}$) cannot be queried" is removed.*

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

Theorem 1. *If a PDPKS scheme is public key unlinkable (resp. public key strongly unlinkable), then it is also derived verification key unlinkable (resp. derived verification key strongly unlinkable).*

Proof. The proof details are referred to Appendix A.

With the above Theorem 1, for the privacy in PDPKS scheme, we only need to consider the public key (strong) unlinkability.

Remark: It seems that the reverse side of Theorem 1, i.e., *"If a PDPKS scheme is derived verification key unlinkable (resp. derived verification key strongly unlinkable), then it is also public key unlinkable (resp. public key strongly unlinkable)"*, also holds and can be be proved trivially. In particular, if there is an algorithm $\mathcal{A}$ that can win the $\mathsf{Game}_{\mathsf{PKUNL}}$, i.e. link a derived verification key to its original public key, then an algorithm $\mathcal{B}$ can be constructed to win the $\mathsf{Game}_{\mathsf{DVKUNL}}$, by running the algorithm $\mathcal{A}$ two times, on the two challenged derived verification keys DVK_0^* and DVK_1^* respectively. We do not investigate the details formally here, since we already have the Theorem 1 and public key unlinkability is more natural for indistinguishability definitions.

3 Our Construction

In this section, we first present some preliminaries, including the bilinear groups and the assumptions, then we propose a PDPKS scheme, which is obtained by applying our approach introduced in Sec. 1 to the IBS scheme by Barreto et al. [3].

3.1 Preliminaries

3.1.1 Bilinear Map Groups [10]

Let λ be a security parameter and p be a λ-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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfying the following peoperties:

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 $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ such that $\psi(Q) = P$.

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

3.1.2 Assumptions

The security of our PDPKS construction relies on the q-Strong Diffie-Hellman (q-SDH) Assumption [8], while the privacy-preserving relies on the Computational Diffie-Hellman (CDH) Assumption [41] on bilinear groups.

Definition 6 (q-Strong Diffie-Hellman Assumption). *[8,3] The q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follows: given a $q+2$-tuple $(P, Q, \beta Q, \beta^2 Q, \ldots, \beta^q Q)$ as input, output a pair $(c, \frac{1}{c+\beta} P)$ with $c \in \mathbb{Z}_p^*$. An algorithm $\mathcal{A}$ has advantage ϵ in solving q-SDH in $(\mathbb{G}_1, \mathbb{G}_2)$ if*

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

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

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

Definition 7 (Computational Diffie-Hellman Assumption). *[41] 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* ϵ *in solving CDH in* $\mathbb{G}_2$ *if*

$$\Pr\left[\mathcal{A}(Q, aQ, bQ) = abQ\right] \geq \epsilon$$

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

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

3.2 Construction

- $\mathsf{Setup}(\lambda) \to \mathsf{PP}$. Upon input a security parameter λ, 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 $H_1 : \mathbb{G}_2 \times \mathbb{G}_2 \to \mathbb{Z}_p^*$, $H_2 : \{0,1\}^* \times \mathbb{G}_T \to \mathbb{Z}_p^*$. The algorithm outputs public parameters

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

  and the message space is $\mathcal{M} = \{0,1\}^*$.
- $\mathsf{KeyGen}(\mathsf{PP}) \to (\mathsf{PK}, \mathsf{SK})$. The algorithm chooses random $\alpha, \beta \in \mathbb{Z}_p^*$, then outputs a public key PK and corresponding secret key SK as

  $$\begin{aligned}\mathsf{PK} :=& (Q_{pub,1}, Q_{pub,2}) = (\alpha Q, \beta Q) \in \mathbb{G}_2 \times \mathbb{G}_2,\\ \mathsf{SK} :=& (\alpha, \beta) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*.\end{aligned}$$

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

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

- $\mathsf{VrfyKeyCheck}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP}) \to 1/0$. Upon input $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$, $\mathsf{PK} = (Q_{pub,1}, Q_{pub,2}) \in \mathbb{G}_2 \times \mathbb{G}_2$, $\mathsf{SK} = (\alpha, \beta) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and the system public parameters PP, the algorithm checks whether $Q_{vk} \stackrel{?}{=} H_1(Q_r, \alpha Q_r)Q + Q_{pub,2}$. If it holds, the algorithm outputs 1, otherwise outputs 0.

– $\mathsf{SignKeyDerive}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP}) \rightarrow \mathsf{DSK}$ or $\perp$. Upon input $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$, $\mathsf{PK} = (Q_{pub,1},\ Q_{pub,2}) \in \mathbb{G}_2 \times \mathbb{G}_2$, $\mathsf{SK} = (\alpha, \beta) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, the algorithm checks whether $Q_{vk} \stackrel{?}{=} H_1(Q_r, \alpha Q_r)Q + Q_{pub,2}$. If it hods, the algorithm outputs a derived signing key

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

otherwise, outputs $\perp$.

– $\mathsf{Sign}(m, \mathsf{DVK}, \mathsf{DSK}, \mathsf{PP}) \rightarrow \sigma$. Upon input a message $m \in \mathcal{M}$, a derived verification key $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$, a signing key $\mathsf{DSK} = P_{sk} \in \mathbb{G}_1$, and the system public parameters 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.

– $\mathsf{Verify}(m, \sigma, \mathsf{DVK}, \mathsf{PP}) \rightarrow 1/0$. Upon 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 $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$, and the system public parameters PP, the algorithm checks whether $h \stackrel{?}{=} H_2(m, e(P_\sigma, Q_{vk})g^{-h})$ 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) $\mathsf{VrfyKeyCheck}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP}) = 1$, since $\alpha Q_r = \alpha r Q = r Q_{pub,1}$, and
(2) $\mathsf{Verify}(m, \mathsf{Sign}(m, \mathsf{DVK}, \mathsf{DSK}, \mathsf{PP}), \mathsf{DVK}, \mathsf{PP}) = 1$, since

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

Note that

$$\begin{aligned} & e(P_{sk}, Q_{vk}) \\ =& e(\frac{1}{H_1(Q_r, \alpha Q_r) + \beta} P, H_1(rQ, rQ_{pub,1})Q + Q_{pub,2}) \\ =& e(\frac{1}{H_1(rQ, \alpha rQ) + \beta} P, H_1(rQ, r\alpha Q)Q + \beta Q) \\ =& e(P, Q) = g. \end{aligned}$$

4 Proofs of Security and Privacy

In this section, we prove our PDPKS construction above is existentially unforgeable under an adaptive chosen-message attack (i.e. is secure) (w.r.t. Def. 1) and is public key strongly unlinkable (w.r.t. Def. 3). For the proof of security, in Sec. 4.1 we reduce the security of our PDPKS construction to the security of the IBS construction by Barreto et al. [3]. For the proof of privacy, in Sec. 4.2 we reduce the public key strong unlinkability of our PDPKS construction to the hardness of CDH problem.

4.1 Proof of Security

Below, we first review the definition and security model of IBS, as well as the IBS construction and security conclusion in [3], then prove the security of our PDPKS construction by giving a reduction from our PDPKS construction to the IBS construction in [3].

4.1.1 Review of Identity-Based Signature in [3]

Definition of Identity-Based Signature Scheme

An IBS scheme consists of following four algorithms:

- $\mathsf{Setup}(\lambda) \rightarrow (\mathsf{PP}, \mathsf{MSK})$. The algorithm takes as input a security parameter λ, runs in polynomial time in λ, and outputs system public parameters PP and a system master secret key MSK.
- $\mathsf{KeyExtract}(\mathsf{ID}, \mathsf{PP}, \mathsf{MSK}) \rightarrow \mathsf{SK}_{\mathsf{ID}}$. The algorithm takes as input an arbitrary identity $\mathsf{ID} \in \{0,1\}^*$, the system public parameters PP, and the master secret key MSK, and outputs a private key $\mathsf{SK}_{\mathsf{ID}}$ for the identity ID.
- $\mathsf{Sign}(m, \mathsf{ID}, \mathsf{PP}, \mathsf{SK}_{\mathsf{ID}}) \rightarrow \sigma$. The algorithm takes as input a message m in the message space $\mathcal{M}$, an identity $\mathsf{ID} \in \{0,1\}^*$, the system public parameters PP, and a private key $\mathsf{SK}_{\mathsf{ID}}$ corresponding to the identity ID, and outputs a signature σ for the message m and the identity ID.
- $\mathsf{Verify}(m, \sigma, \mathsf{ID}, \mathsf{PP}) \rightarrow 1/0$. The algorithm takes as input a (message, signature) pair (m, σ), an identity $\mathsf{ID} \in \{0,1\}^*$, and the system public parameters PP, and outputs a bit $b \in \{0,1\}$, with $b = 1$ meaning valid and $b = 0$ meaning invalid.

Security Model of IBS

Definition 8. *An IBS scheme is existentially unforgeable under adaptive chosen message and identity attacks if no PPT adversary has a non-negligible advantage in the following game* $\mathsf{Game}_{\mathsf{IBS,UEF}}$*:*

- **Setup**. $(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}()$ *is run and* PP *are given to the adversary* $\mathcal{A}$*.*
- **Probing Phase**. *The adversary can adaptively query the following oracles:*
  - *Key Extract Oracle* $\mathsf{OKeyExtract}(\cdot)$*: Upon input an arbitrary identity* $\mathsf{ID} \in \{0,1\}^*$*,* $\mathsf{OKeyExtract}(\mathsf{ID})$ *returns the corresponding private key* $\mathsf{SK}_{\mathsf{ID}}$ *to* $\mathcal{A}$*.*
  - *Signing Oracle* $\mathsf{OSign}(\cdot,\cdot)$*: Upon input a message* $m \in \mathcal{M}$ *and an identity* $\mathsf{ID} \in \{0,1\}^*$*,* $\mathsf{OSign}(m, \mathsf{ID})$ *returns* $\mathsf{Sign}(m, \mathsf{ID}, \mathsf{PP}, \mathsf{SK}_{\mathsf{ID}})$ *to* $\mathcal{A}$*, where* $\mathsf{SK}_{\mathsf{ID}}$ *is a private key for* ID*.*
- **Output Phase**. $\mathcal{A}$ *outputs a message* $m^* \in \mathcal{M}$*, an identity* ID^**, and a signature* σ^**.* $\mathcal{A}$ *succeeds in the the game if* $\mathsf{Verify}(m^*, \sigma^*, \mathsf{ID}^*, \mathsf{PP}) = 1$ *under the* **restrictions** *that (1)* $\mathsf{OKeyExtract}(\mathsf{ID}^*)$ *is never queried, and (2)* $\mathsf{OSign}(m^*, \mathsf{ID}^*)$ *is never queried.*

Construction of the IBS in [3]

Below is the IBS construction in [3]. [13]

- $\mathsf{Setup}(\lambda) \rightarrow (\mathsf{PP}, \mathsf{MSK})$. Upon input a security parameter λ, 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 $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, $H_2 : \{0,1\}^* \times \mathbb{G}_T \rightarrow \mathbb{Z}_p^*$. The algorithm selects random $\beta \in \mathbb{Z}_p^*$ and computes $Q_{pub} = \beta Q \in \mathbb{G}_2$, then outputs public parameters PP and master secret key MSK as $\mathsf{PP} := (p, (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \psi), P, Q, Q_{pub}, g, H_1, H_2),\ \mathsf{MSK} := \beta$. The message space is $\mathcal{M} = \{0,1\}^*$.
- $\mathsf{KeyExtract}(\mathsf{ID}, \mathsf{PP}, \mathsf{MSK}) \rightarrow \mathsf{SK}_{\mathsf{ID}}$. Upon input an arbitrary identity $\mathsf{ID} \in \{0,1\}^*$, the system public parameters PP, and the master secret key MSK, the algorithm outputs a private key $\mathsf{SK}_{\mathsf{ID}}$ for the identity ID as $\mathsf{SK}_{\mathsf{ID}} = \frac{1}{H_1(\mathsf{ID})+\beta} P \in \mathbb{G}_1$.
- $\mathsf{Sign}(m, \mathsf{ID}, \mathsf{PP}, \mathsf{SK}_{\mathsf{ID}}) \rightarrow \sigma$. Upon input a message $m \in \{0,1\}^*$, an identity $\mathsf{ID} \in \{0,1\}^*$, the system public parameters PP, and a private key $\mathsf{SK}_{\mathsf{ID}}$ for the identity ID, the algorithm
  1. picks a random $x \in \mathbb{Z}_p^*$ and computes $X = g^x$,
  2. sets $h = H_2(m, X) \in \mathbb{Z}_p^*$,

[13] Note that we slightly changed the variable names in the IBS construction, to better suit our PDPKS construction in later proof.

3. computes $P_\sigma = (x + h)\mathsf{SK}_{\mathsf{ID}} \in \mathbb{G}_1$,

and outputs $\sigma = (h, P_\sigma) \in \mathbb{Z}_p^* \times \mathbb{G}_1$ as a signature for message m and identity ID.

– $\mathsf{Verify}(m, \sigma, \mathsf{ID}, \mathsf{PP}) \rightarrow 1/0$. Upon input a message $m \in \{0,1\}^*$, a signature $\sigma = (h, P_\sigma) \in \mathbb{Z}_p^* \times \mathbb{G}_1$, an identity $\mathsf{ID} \in \{0,1\}^*$, and the system public parameters PP, the algorithm outputs $b = 1$ if and only if $h = H_2(m, e(P_\sigma, H_1(\mathsf{ID})Q + Q_{pub})g^{-h})$.

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

$$\begin{aligned} \mathsf{CMPK} &:= (p, (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \psi), P, Q, g, H_1, H_2), \\ \mathsf{IMPK} &:= (Q_{pub}), \\ F(\mathsf{PP}, \mathsf{ID}) &:= H_1(\mathsf{ID})Q + Q_{pub}. \end{aligned}$$

Security of the IBS in [3]

The security of the IBS construction in [3] is established by the following lemma.

Lemma 1. *[3, Theorem 1] Let us assume that there exists an adaptively chosen message and identity attacker $\mathcal{A}$ making q_{h_i} queries to random oracles $H_i (i = 1, 2)$ and q_s queries to the signing oracle. Assume that, within a time t, $\mathcal{A}$ produces a forgery with probability $\epsilon \geq 10(q_s+1)(q_s+q_{h_2})/2^\lambda$. Then, there exists an algorithm $\mathcal{B}$ that is able to solve the q-SDH Problem for $q = q_{h_1}$ in an expected time*

$$t' \leq 120686 q_{h_1} q_{h_2} (t + O(q_s \tau_p))/(\epsilon(1 - q/2^\lambda)) + O(q^2 \tau_{mult})$$

where τ_{mult} and τ_p respectively denote the cost of a scalar multiplication in $\mathbb{G}_2$ and the required time for pairing evaluation.

4.1.2 Security Proof of our PDPKS Construction

Now we prove the security of our PDPKS construction, by a reduction to the IBS construction in [3], as shown in the following Lemma 2.

Lemma 2. *Assume that there exists an adaptively chosen message attacker $\mathcal{A}$ that 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 in* $\mathsf{Game}_{\mathsf{UEF}}$ *for our PDPKS construction. Assume that, within time t, $\mathcal{A}$ produces a forgery with probability ϵ. Then, there exists an algorithm $\mathcal{B}$ that is able to produce within time $\bar{t} = t + O(q_a \tau_{mult})$ a forgery with probability $\bar{\epsilon} = \epsilon$ in* $\mathsf{Game}_{\mathsf{IBS,UEF}}$ *for the IBS construction in [3], where $\mathcal{B}$ makes $\bar{q}_{h_i}$ queries to random oracles $H_i(i = 1, 2)$ and $\bar{q}_s$ queries to the signing oracle, with $\bar{q}_{h_1} \leq q_{h_1} + q_a, \bar{q}_{h_2} \leq q_{h_2}, \bar{q}_s \leq q_s$.*

Proof. Below, $\mathcal{B}$ acts as an adversary in $\mathsf{Game}_{\mathsf{IBS,UEF}}$ to interact with a challenger $\mathcal{C}$ which simulates the IBS scheme Π_{ibs} to $\mathcal{B}$ in the random oracle model, and at the same time, $\mathcal{B}$ simulates our PDPKS scheme Π_{pdpks} to $\mathcal{A}$, which is an adversary for $\mathsf{Game}_{\mathsf{UEF}}$ in the random oracle model. $\mathcal{B}$ tries to attack π_{ibs}, by making use of $\mathcal{A}'s$ attacking ability to our Π_{pdpks}.

Setup (for $\mathsf{Game}_{\mathsf{IBS,UEF}}$). *$\mathcal{B}$ is given $\Pi_{ibs}.\mathsf{PP} = (p, (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \psi), P, Q, Q_{pub}, g, \tilde{H}_1, H_2)$.*
Then $\mathcal{B}$ simulates $\mathsf{Game}_{\mathsf{UEF}}$**.Setup** *to $\mathcal{A}$ as follows.*

$\mathcal{B}$ sets $\Pi_{pdpks}.\mathsf{PP} = (p, (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \psi), P, Q, g, H_1, H_2)$, and gives $\Pi_{pdpks}.\mathsf{PP}$ to $\mathcal{A}$, where $H_1 : \mathbb{G}_2 \times \mathbb{G}_2 \to \mathbb{Z}_p^*$ is defined as: for any $(preimg_1, preimg_2) \in \mathbb{G}_2 \times \mathbb{G}_2, H_1(preimg_1, preimg_2) := \tilde{H}_1(preimg_1 \| preimg_2)$ where '$\|$' denotes concatenation.

$\mathcal{B}$ chooses random $\alpha \in \mathbb{Z}_p^*$, sets $Q_{pub,1} = \alpha Q$ and $Q_{pub,2} = Q_{pub}$, then gives $\Pi_{pdpks}.\mathsf{PK} := (Q_{pub,1}, Q_{pub,2})$ to $\mathcal{A}$.

$\mathcal{B}$ initializes an empty list $L_{H_1} = \emptyset$, each element of which will be a (preimage 1, preimage 2, hash value) tuple $(preimg_1, preimg_2, hval)$.

$\mathcal{B}$ initializes an empty list $L_{dvk} = \emptyset$, each element of which will be a (derived verification key, derived signing key, corrupted, preimage 1, preimage 2) tuple $(\mathsf{DVK}, \mathsf{DSK}, corrupted, preimg_1, preimg_2)$, satisfying $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$, $preimg_1 = Q_r$, $preimg_2 = \alpha Q_r$, and $Q_{vk} = H_1(preimg_1, preimg_2)Q + Q_{pub,2}$.

Probing Phase (for $\mathsf{Game}_{\mathsf{IBS,UEF}}$). *According to the queries that $\mathcal{A}$ makes adaptively in* $\mathsf{Game}_{\mathsf{UEF}}$**.Probing Phase**, *$\mathcal{B}$ makes adaptive queries to the challenger $\mathcal{C}$ in* $\mathsf{Game}_{\mathsf{IBS,UEF}}$ *as follows.*

- When $\mathcal{A}$ makes a H_1 query for input $(preimg_1, preimg_2) \in \mathbb{G}_2 \times \mathbb{G}_2$ to $\mathcal{B}$: $\mathcal{B}$ searches L_{H_1} to find a tuple $ht \in L_{H_1}$ such that $ht.preimg_1 = preimg_1 \; AND \; ht.preimg_2 = preimg_2$:
  - if such a tuple exists, $\mathcal{B}$ returns $ht.hval$ to $\mathcal{A}$.

- otherwise, $\mathcal{B}$ makes a $\tilde{H}_1$ query for input $preimg_1 \| preimg_2$ to $\mathcal{C}$ and obtains a hash value $hval$, then $\mathcal{B}$ adds $(preimg_1, preimg_2, hval)$ to L_{H_1}, and returns $hval$ to $\mathcal{A}$.

– When $\mathcal{A}$ makes an $\mathsf{ODVKAdd}(\cdot)$ query for input $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$ to $\mathcal{B}$:

$\mathcal{B}$ searches L_{dvk} to find a tuple $dvk \in L_{dvk}$ such that $dvk.\mathsf{DVK} = \mathsf{DVK}$. If such a tuple exists, $\mathcal{B}$ return 1 to $\mathcal{A}$; Otherwise,

1. $\mathcal{B}$ searches L_{H_1} to find a tuple $ht \in L_{H_1}$ such that $ht.preimg_1 = Q_r$ AND $ht.preimg_2 = \alpha Q_r$. If such a tuple does not exist, $\mathcal{B}$ makes a $\tilde{H}_1$ query for input $Q_r \| \alpha Q_r$ to $\mathcal{C}$ and obtains a value $hval = \tilde{H}_1(Q_r \| \alpha Q_r)$, then sets $ht = (Q_r, \alpha Q_r, hval)$ and adds ht to L_{H_1}.
2. $\mathcal{B}$ then checks if $ht.hval Q + Q_{pub,2} \stackrel{?}{=} Q_{vk}$. If the equation holds, $\mathcal{B}$ adds $(\mathsf{DVK}, null, 0, Q_r, \alpha Q_r)$ to L_{dvk} and returns 1 to $\mathcal{A}$; otherwise, $\mathcal{B}$ returns 0 to $\mathcal{A}$.

– When $\mathcal{A}$ makes an $\mathsf{ODSKCorrput}(\cdot)$ query for input $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$ to $\mathcal{B}$:

Note that $\mathcal{A}$ can only query $\mathsf{ODSKCorrput}(\cdot)$ for input DVK such that DVK exists in L_{dvk}, $\mathcal{B}$ searches L_{dvk} to find a tuple $dvk \in L_{dvk}$ such that $dvk.\mathsf{DVK} = \mathsf{DVK}$. If such a tuple does not exist, return $\perp$ to $\mathcal{A}$, otherwise

1. $\mathcal{B}$ sets $\mathsf{ID} = dvk.preimg_1 \| dvk.preimg_2$ and makes a query $\mathsf{OKeyExtract}(\cdot)$ for input ID to $\mathcal{C}$, and obtains a private key $\mathsf{SK}_{\mathsf{ID}} \in \mathbb{G}_1$ for ID.
2. $\mathcal{B}$ sets $\mathsf{DSK} = \mathsf{SK}_{\mathsf{ID}}$ and returns DSK to $\mathcal{A}$. Note that $\mathsf{SK}_{\mathsf{ID}} = \frac{1}{\tilde{H}_1(dvk.preimg_1 || dvk.preimg_2)+\beta} P = \frac{1}{H_1(dvk.preimg_1, dvk.preimg_2)+\beta} P$, i.e. from the view of $\mathcal{A}$, it obtains a valid derived signing key corresponding to $\mathsf{DVK} = (Q_r, Q_{vk})$, since

$$
\begin{aligned}
Q_r &= dvk.\mathsf{DVK}.Q_r = dvk.preimg_1, \\
Q_{vk} &= dvk.\mathsf{DVK}.Q_{vk} \\
&\quad = H_1(dvk.preimg_1, dvk.preimg_2) Q + Q_{pub,2}, \\
dvk.preimg_2 &= \alpha \cdot dvk.preimg_1.
\end{aligned}
$$

3. $\mathcal{B}$ updates the tuple dvk (in L_{dvk}) by setting $dvk.\mathsf{DSK} = \mathsf{SK}_{\mathsf{ID}}$, $dvk.corrupted = 1$.

– When $\mathcal{A}$ makes a signing query $\mathsf{OSign}(\cdot, \cdot)$ for input message $m \in \mathcal{M}$ and derived verification key $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$ to $\mathcal{B}$:

Note that $\mathcal{A}$ can only query $\mathsf{OSign}(\cdot, \cdot)$ for input DVK such that DVK exists in L_{dvk}, $\mathcal{B}$ searches L_{dvk} to find a tuple $dvk \in L_{dvk}$ such that $dvk.\mathsf{DVK} = \mathsf{DVK}$. If such a tuple does not exist, return $\perp$ to $\mathcal{A}$, otherwise

1. $\mathcal{B}$ sets $\mathsf{ID} = dvk.preimg_1 \| dvk.preimg_2$ and makes a query $\mathsf{OSign}(\cdot,\cdot)$ for input m and ID to $\mathcal{C}$, and obtains a signature $\sigma = (h, P_\sigma) \in \mathbb{Z}_p^* \times \mathbb{G}_1$ such that $h = H_2(m, e(P_\sigma, \tilde{H}_1(\mathsf{ID})Q + Q_{pub})g^{-h})$.
2. $\mathcal{B}$ forwards $\sigma = (h, P_\sigma)$ to $\mathcal{A}$. Note that (h, P_σ) satisfies $h = H_2(m, e(P_\sigma, \mathsf{DVK}.Q_{vk})g^{-h})$, since

$$\begin{aligned}
&\tilde{H}_1(\mathsf{ID})Q + Q_{pub} \\
&= \tilde{H}_1(dvk.preimg_1 \| dvk.preimg_2)Q + Q_{pub} \\
&= H_1(dvk.preimg_1, dvk.preimg_2)Q + Q_{pub,2} \\
&= dvk.\mathsf{DVK}.Q_{vk} = \mathsf{DVK}.Q_{vk},
\end{aligned}$$

   i.e. from the view of $\mathcal{A}$, σ is a valid signature for message m and derived verification key DVK.

Output Phase (for $\mathsf{Game}_{\mathsf{IBS,UEF}}$). In the $\mathsf{Game}_{\mathsf{UEF}}$**.Output Phase**, $\mathcal{A}$ outputs a message $m^* \in \mathcal{M}$, a derived verification key $\mathsf{DVK}^* = (Q_r^*, Q_{vk}^*) \in \mathbb{G}_2 \times \mathbb{G}_2$ such that there is a tuple $dvk \in L_{dvk}$ with $dvk.\mathsf{DVK} = \mathsf{DVK}^*$, and a signature $\sigma^* = (h^*, P_\sigma^*) \in \mathbb{Z}_p^* \times \mathbb{G}_1$. $\mathcal{B}$ sets $\mathsf{ID}^* = dvk.preimg_1 \| dvk.preimg_2$, and forwards $(m^*, \mathsf{ID}^*, \sigma^*)$ to $\mathcal{C}$. Note that

- $\Pi_{pdpks}.\mathsf{Verify}(m^*, \sigma^*, \mathsf{DVK}^*, \mathsf{PDPKS.PP}) = 1$ means $h^* = H_2(m^*, e(P_\sigma^*, Q_{vk}^*)g^{-h^*})$, and this implies $h^* = H_2(m^*, e(P_\sigma^*, \tilde{H}_1(\mathsf{ID}^*)Q + Q_{pub})g^{-h^*})$, since

$$\begin{aligned}
Q_{vk}^* &= dvk.\mathsf{DVK}.Q_{vk} \\
&= H_1(dvk.preimg_1, dvk.preimg_2)Q + Q_{pub,2} \\
&= \tilde{H}_1(\mathsf{ID}^*)Q + Q_{pub},
\end{aligned}$$

  i.e., $\Pi_{ibs}.\mathsf{Verify}(m^*, \sigma^*, \mathsf{ID}^*, \mathsf{IBS.PP}) = 1$.
- That $\mathcal{A}$ never made query $\mathsf{ODSKCorrput}(\mathsf{DVK}^*)$ implies that $\mathcal{B}$ never made query $\mathsf{OKeyExtract}(\mathsf{ID}^*)$ to $\mathcal{C}$.
- That $\mathcal{A}$ never made query $\mathsf{OSign}(m^*, \mathsf{DVK}^*)$ implies that $\mathcal{B}$ never made query $\mathsf{OSign}(m^*, \mathsf{ID}^*)$ to $\mathcal{C}$.

This implies that if $\mathcal{A}$ wins $\mathsf{Game}_{\mathsf{UEF}}$ agains Π_{pdpks}, then $\mathcal{B}$ wins $\mathsf{Game}_{\mathsf{IBS,UEF}}$ against Π_{ibs}.

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. This follows Lemma 1 and Lemma 2 immediately.

4.2 Proof of Privacy

Now we prove that our PDPKS construction in Sec. 3 is public key strongly unlinkable (w.r.t. Def. 3).

Theorem 3. *The PDPKS scheme is public key strongly 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_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 $\mathsf{Game}_{\mathsf{PKSUNL}}$ with advantage ϵ, then there exists an algorithm $\mathcal{B}$ that runs within time $\bar{t} = t + O((q_{h_1} + q_s)\tau_{mult}) + O((q_{h_1} + q_a)\tau_p) + O(q_s \tau_{exp})$, where τ_{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. Below we show that, if there exists a PPT adversary $\mathcal{A}$ that can win $\mathsf{Game}_{\mathsf{PKSUNL}}$ 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.

Setup. $\mathcal{B}$ is given an instance of CDH problem on bilinear map groups, i.e. bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \psi)$ of prime order p, generator $Q \in \mathbb{G}_2$, and a tuple $(A = aQ, B = bQ) \in \mathbb{G}_2 \times \mathbb{G}_2$ for unknown $a, b \in \mathbb{Z}_p^*$, and the target of $\mathcal{B}$ is to compute an element $C \in \mathbb{G}_2$ such that $C = abQ$.

$\mathcal{B}$ sets $\mathsf{PP} := (p, (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \psi), P, Q, g, H_1, H_2)$ and gives PP to $\mathcal{A}$, where $P = \psi(Q) \in \mathbb{G}_1$, $g = e(P, Q)$, and H_1 and H_2 are hash functions modeled as random oracles.

$\mathcal{B}$ chooses random $\alpha_0', \beta_0, \alpha_1', \beta_1 \in \mathbb{Z}_p^*$, sets $Q_{pub,1}^{(0)} = \alpha_0' A, Q_{pub,2}^{(0)} = \beta_0 Q$, $Q_{pub,1}^{(1)} = \alpha_1' A, Q_{pub,2}^{(1)} = \beta_1 Q$, and gives $\mathsf{PK}_0 := (Q_{pub,1}^{(0)}, Q_{pub,2}^{(0)}), \mathsf{PK}_1 := (Q_{pub,1}^{(1)}, Q_{pub,2}^{(1)})$ to $\mathcal{A}$. Note that the secret keys corresponding to PK_0 and PK_1 are $\mathsf{SK}_0 := (\alpha_0' a, \beta_0)$ and $\mathsf{SK}_1 := (\alpha_1' a, \beta_1)$ respectively, where $\mathcal{B}$ does not know the value of a.

$\mathcal{B}$ initializes an empty list $L_{H_1} = \emptyset$, each element of which will be a (preimage 1, preimage 2, hash value, group element) tuple $(preimg_1, preimg_2, hval, hvalQ)$.

$\mathcal{B}$ initializes an empty list $L_{dvk} = \emptyset$, each element of which will be a (derived verification key, derived signing key, corrupted, preimage 1, preimage 2, public key index) tuple ($\mathsf{DVK}, \mathsf{DSK}, corrupted, preimg_1, preimg_2, i$), satisfying (1) $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$, (2) $i \in \{0, 1\}$, (3) $preimg_1 = Q_r$ and $preimg_2$ satisfies $e(P, preimg_2) = e(\psi(Q_{pub,1}^{(i)}), preimg_1)$, i.e. assuming $preimg_1 = Q_r = rQ$ for some $r \in \mathbb{Z}_p^*$, then $preimg_2$ satisfies $preimg_2 = (\alpha_i' a)preimg_1 = \alpha_i' arQ = rQ_{pub,1}^{(i)}$, and (4) $Q_{vk} = H_1(preimg_1, preimg_2)Q + Q_{pub,2}^{(i)}$.

Phase 1.

- When $\mathcal{A}$ makes a H_1 query for input $(preimg_1, preimg_2) \in \mathbb{G}_2 \times \mathbb{G}_2$ to $\mathcal{B}$:

  $\mathcal{B}$ searches L_{H_1} to find a tuple $ht \in L_{H_1}$ such that $ht.preimg_1 = preimg_1$ AND $ht.preimg_2 = preimg_2$:

  - if such a tuple exists, $\mathcal{B}$ returns $ht.hval$ to $\mathcal{A}$.
  - otherwise, $\mathcal{B}$ chooses a random $hval \in \mathbb{Z}_p^*$, adds $(preimg_1, preimg_2, hval, havlQ)$ to L_{H_1}, and returns $hval$ to $\mathcal{A}$.
- When $\mathcal{A}$ makes a query $\mathsf{ODVKAdd}(\cdot, \cdot)$ for input $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$ and $\mathsf{PK}_i (i \in \{0, 1\})$ to $\mathcal{B}$:

  $\mathcal{B}$ searches L_{dvk} to find a tuple $dvk \in L_{dvk}$ such that $dvk.\mathsf{DVK} = \mathsf{DVK}$ AND $dvk.i = i$. If such a tuple exists, $\mathcal{B}$ return 1 to $\mathcal{A}$. Otherwise, $\mathcal{B}$ searches L_{H_1} to find a tuple $ht \in L_{H_1}$ such that $ht.hvalQ = Q_{vk} - Q_{pub,2}^{(i)}$. Note the validity requirement for a derived verification key and that $\mathcal{A}$ can obtain the hash values for H_1 only by making H_1 queries, if such a tuple does not exist, $\mathcal{B}$ returns 0 to $\mathcal{A}$, [14] otherwise

  - if $ht.preimg_1 = Q_r$ AND $e(P, ht.preimg_2) = e(\psi(Q_{pub,1}^{(i)}), ht.preimg_1)$ holds, $\mathcal{B}$ returns 1 to $\mathcal{A}$ and adds $(\mathsf{DVK}, \frac{1}{ht.hval+\beta_i} P, 0, preimg_1, preimg_2, i)$ to L_{dvk}.

    Note that the equation $e(P, ht.preimg_2) = e(\psi(Q_{pub,1}^{(i)}), ht.preimg_1)$ implies that $ht.preimg_2 = (\alpha_i' a)ht.preimg_1$. Assuming $Q_r = rQ$ for some $r \in \mathbb{Z}_p^*$, note that $ht.hvalQ = Q_{vk} - Q_{pub,2}^{(i)}$, we have that (Q_r, Q_{vk}) satisfies $Q_{vk} = ht.hvalQ + Q_{pub,2}^{(i)} = H_1(ht.preimg_1, ht.preimg_2)Q + Q_{pub,2}^{(i)} = H_1(Q_r, \alpha_i' aQ_r)Q + Q_{pub,2}^{(i)} = H_1(rQ, rQ_{pub,1}^{(i)})Q + Q_{pub,2}^{(i)}$.
  - otherwise, $\mathcal{B}$ returns 0 to $\mathcal{A}$.
- When $\mathcal{A}$ makes a query $\mathsf{ODSKCorrput}(\cdot)$ for input $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$ to $\mathcal{B}$:

[14] Without making a H_1 query that produces such a tuple, the chance that $\mathsf{DVK} = (Q_r, Q_{vk})$ is a valid derived verification key is negligible.

Note that $\mathcal{A}$ can only query $\mathsf{ODSKCorrput}(\cdot)$ for input DVK such that DVK exists in L_{dvk}, $\mathcal{B}$ searches L_{dvk} to find a tuple $dvk \in L_{dvk}$ such that $dvk.\mathsf{DVK} = \mathsf{DVK}$. If such a tuple does not exist, return $\perp$ to $\mathcal{A}$, otherwise $\mathcal{B}$ returns $dvk.\mathsf{DSK}$ to $\mathcal{A}$ and sets $dvk.corrupted = 1$.

– When $\mathcal{A}$ makes a query $\mathsf{OSign}(\cdot,\cdot)$ for input message $m \in \mathcal{M}$ and derived verification key $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$ to $\mathcal{B}$:

Note that $\mathcal{A}$ can only query $\mathsf{OSign}(\cdot,\cdot)$ for input DVK such that DVK exists in L_{dvk}, $\mathcal{B}$ searches L_{dvk} to find a tuple $dvk \in L_{dvk}$ such that $dvk.\mathsf{DVK} = \mathsf{DVK}$. If such a tuple does not exist, return $\perp$ to $\mathcal{A}$, otherwise $\mathcal{B}$ returns $(h, P_\sigma) \leftarrow \mathsf{Sign}(m, \mathsf{DVK}, dvk.\mathsf{DSK}, \mathsf{PP})$ to $\mathcal{A}$.

Challenge. A random bit $i^* \in \{0,1\}$ is chosen. $\mathcal{B}$ generates the challenge derived verification key $\mathsf{DVK}^* = (Q_r^*, Q_{vk}^*)$ from PK_{i^*} as follows:

1. Set $Q_r^* = B$.
2. Note that $B = bQ$ and Q_{vk}^* should be $Q_{vk}^* = H_1(B, bQ_{pub,1}^{(i^*)})Q + Q_{pub,2}^{(i^*)} = H_1(B, b\alpha'_{i^*}aQ)Q + Q_{pub,2}^{(i^*)}$, where a and b are unknown to $\mathcal{B}$. $\mathcal{B}$ chooses a random $hval^* \in \mathbb{Z}_p^*$, and adds $(B, \top, hval^*, hval^*Q)$ to L_{H_1}, where $\top$ is a special symbol to denote the value of $\alpha'_{i^*}abQ$ that is unknown by $\mathcal{B}$. $\mathcal{B}$ sets $Q_{vk}^* = hval^*Q + Q_{pub,2}^{(i^*)}$ and gives $\mathsf{DVK}^* = (Q_r^*, Q_{vk}^*)$ to $\mathcal{A}$.
3. $\mathcal{B}$ sets $\mathsf{DSK}^* = \frac{1}{hval^* + \beta_{i^*}} P$ and adds $(\mathsf{DVK}^*, \mathsf{DSK}^*, 0, B, \top, i^*)$ to L_{dvk}.

Phase 2. Similar to **Phase 1**,

– When $\mathcal{A}$ makes a H_1 query for input $(preimg_1, preimg_2) \in \mathbb{G}_2 \times \mathbb{G}_2$ to $\mathcal{B}$:

$\mathcal{B}$ acts in the same way as in that of **Phase 1** except that if $preimg_1 = B$ AND $e(P, preimg_2) = e(\alpha'_i\psi(A), preimg_1)$ for $i = 0$ or 1 (denote this event by $\mathbf{E}$), which implies that $preimg_2 = \alpha'_i abQ = b\alpha'_i A = bQ_{pub,1}^{(i)}$, $\mathcal{B}$ outputs $\frac{1}{\alpha'_i} preimg_2$ as the solution for the CDH problem and aborts the game.

$\mathcal{B}$ acts in the same way as in that of **Phase 1**.

– When $\mathcal{A}$ makes a query $\mathsf{ODVKAdd}(\cdot,\cdot)$ for input $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$ and $\mathsf{PK}_i (i \in \{0,1\})$ to $\mathcal{B}$:

If $Q_r \neq Q_r^*$, $\mathcal{B}$ acts in the same way as in that of **Phase 1**.

If $Q_r = Q_r^*$, note that $\mathcal{A}$ is only allowed to query $\mathsf{DVK} \neq \mathsf{DVK}^*$, we only need to consider $Q_{vk} \neq Q_{vk}^*$. If $Q_r = Q_r^*$ AND $Q_{vk} \neq Q_{vk}^*$, $\mathcal{B}$ directly returns 0 to $\mathcal{A}$, since

- If $\mathsf{PK}_i = \mathsf{PK}_{i^*}$, then DVK is invalid for sure and hence should be rejected.

- If $\mathsf{PK}_i = \mathsf{PK}_{1-i^*}$, since there has been no corresponding random oracle query made to H_1 yet (as otherwise $\mathcal{B}$ will solve the CDH problem and abort the game if this happens), $\mathcal{B}$ returns 0 to $\mathcal{A}$ as in **Phase 1**.

– When $\mathcal{A}$ makes a query $\mathsf{ODSKCorrput}(\cdot)$ for input $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$ to $\mathcal{B}$:
$\mathcal{B}$ acts in the same way as in that of **Phase 1**. Note that for public key strong unlinkability, the adversary $\mathcal{A}$ is allowed to make $\mathsf{ODSKCorrput}(\cdot)$ query on input the challenge derived verification key DVK^*. As shown above, $\mathcal{B}$ knows the values of $hval^*$ and β_{i^*}, so that $\mathcal{B}$ can generate the derived signing key corresponding to DVK^*, namely, $\mathsf{DSK}^* = \frac{1}{hval^* + \beta_{i^*}} P$.

– When $\mathcal{A}$ makes a query $\mathsf{OSign}(\cdot, \cdot)$ for input message $m \in \mathcal{M}$ and derived verification key $\mathsf{DVK} = (Q_r, Q_{vk}) \in \mathbb{G}_2 \times \mathbb{G}_2$ to $\mathcal{B}$:
$\mathcal{B}$ acts in the same way as in that of **Phase 1**. Note that as shown above, $\mathcal{B}$ can generate the derived signing key corresponding to DVK^*, so that even when the adversary $\mathcal{A}$ makes a query $\mathsf{OSign}(\cdot, \cdot)$ on input the challenge derived verification key DVK^*, $\mathcal{B}$ can answer the query by running the sign algorithm using the derived signing key.

Guess. $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$ as its guess to i^*. Note that it implies event $\mathbf{E}$ does not occur in this case, and $\mathcal{B}$ outputs $\perp$ and aborts the game (i.e., $\mathcal{B}$ fails to solve the CDH problem).

Analysis. Let $\mathbf{G_0}$ and $\mathbf{G_1}$ denote the original $\mathsf{Game}_{\mathsf{PKSUNL}}$ game and the above simulated game by $\mathcal{B}$, respectively. Let $\mathbf{E}'$ denote the event that $\mathcal{A}$ makes a valid verification key adding query without first making the corresponding H_1 query. Then we have

$$\Pr[\mathbf{E}'] \leq q_a/p$$

where q_a denotes the number of verification key adding queries.

If event $\mathbf{E}$ and $\mathbf{E}'$ don't occur, then $\mathbf{G_0}$ and $\mathbf{G_1}$ are identical. So we have

$$\Pr[\mathcal{A} \text{ wins in } \mathbf{G_0} | \neg(\mathbf{E} \vee \mathbf{E}')] = \Pr[\mathcal{A} \text{ wins in } \mathbf{G_1} | \neg(\mathbf{E} \vee \mathbf{E}')]$$

which gives

$$\Pr[(\mathbf{E} \vee \mathbf{E}')] \geq |\Pr[\mathcal{A} \text{ wins in } \mathbf{G_0}] - \Pr[\mathcal{A} \text{ wins in } \mathbf{G_1}]| \,.$$

Since H_1 is modeled as a random oracle, and in game $\mathbf{G_1}$ the adversary $\mathcal{A}$ does not obtain any information about $H_1(Q_r^* = g^b, bQ_{pub,1}^{(0)})$ or $H_1(Q_r^* = g^b, bQ_{pub,1}^{(1)})$, then $(\mathsf{DVK}^*, \mathsf{DSK}^*)$ does not reveal any information about i^*. Also, the adversary is forbidden to query DVK^* in any $\mathsf{ODVKAdd}$ query,

so the adversary has no advantage in $\mathbf{G_1}$, which means $\Pr[\mathcal{A}$ wins in $\mathbf{G_1}] = 1/2$. Therefore, if the adversary has advantage ϵ over random guess in winning the original game, i.e., $\Pr[\mathcal{A}$ wins in $\mathbf{G_0}] = \epsilon + 1/2$, then

$$\Pr[\mathbf{E}] + \Pr[\mathbf{E}'] \geq \Pr[(\mathbf{E} \vee \mathbf{E}')] \geq \epsilon$$

and we have

$$\Pr[\mathbf{E}] \geq \epsilon - q_a/p$$

which means $\mathcal{B}$ can solve the CDH problem with probability at least $\epsilon - q_a/p$.

5 Application and Implementation

As shown previously, Monero's core cryptographic protocol and Bitcoin's deterministic wallet algorithm suffer a security vulnerability that may endanger users' coin-spending keys. The proposed PDPKS scheme provides a secure and convenient tool to fix this problem, without weakening their functionality and privacy features.

In addition, note that the cryptocurrencies with strong privacy often suffer low efficiency or even undesired features, for example, Monero's signature size is linear in the size of ring which the actually spent coin is hidden in [15], ZCash suffers long proof generation time and the requirement of a trusted setup. For some scenarios, it is desirable to achieve moderate privacy without security or efficiency concerns. As Bitcoin suggests, *"a new key pair should be used for each transaction to keep them from being linked to a common owner"*[28], "each coin with a fresh/unique public key" is the bottom line for preserving privacy, since otherwise the coins with the same public key will be linked, so do the corresponding transactions. On the other side, it is a simple but effective way to enhance privacy. Actually, the concept of "stealth addresses" [40] has described the functionality and privacy requirements of "each coin with a fresh/unique public key". The proposed PDPKS scheme provides a secure and convenient tool for implementing stealth addresses in cryptocurrencies by simply using our PDPKS scheme as the underlying digital signature scheme.

On the implementation, note that our construction is using a type-2 pairing [21] and does not need to hash to $\mathbb{G}_2$, so it can be implemented based on any pairing friendly curve [21]. We suggest to use the Barreto-Naehrig (BN) curve [4], which has been well studied and regarded

[15] More specifically, using a ring with n members means the payer hides itself in n potential payers, i.e. to get stronger privacy, the payer has to use a larger ring.

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 suggest to adopt the parameter recommended in the recent work by Barbulescu and Duquesne [2, 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, 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 [30], which are however now invalidated by Kim and Barbulescu's recent progress on number field sieve algorithm for discrete logarithms in F_{p^n} [25]. That is why we suggest to use the above parameter recommended by Barbulescu and Duquesne [2, Section 6.1], which has taken into account the attacking algorithm in [25].

On the verification key and signature size, with the parameter suggested above, the verification key (i.e. the coin-receiving address), say $(R, S) \in \mathbb{G}_2 \times \mathbb{G}_2$, is 1848-bits, and the signature, say $(h, P_\sigma) \in \mathbb{Z}_p^* \times \mathbb{G}_1$, is 924-bits. These are larger than that of ECDSA implemented on elliptic curve "secp256k1" [39] for 128-bits security, which is used by Bitcoin, with public key size 264-bits and signature size 520-bits. [16] But for cryptocurrencies, this is a reasonable and acceptable cost for achieving enhanced privacy with solid security and convenient functionality. On the computation time for deriving fresh verification key, signing, and verification, verification is the most expensive, since it needs one paring computation. According to the experimental results by Khandaker et al. [24, Section 4], for the parameter suggested above, on a usual computation environment (Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz, 4GB Memory), one pairing computation needs less than 8 ms. This is fast enough for a signature scheme to be applied in cryptocurrencies.

6 Conclusion

In this paper, we identified a security vulnerability in the privacy-preserving key derivation algorithm of Monero. To provide a practical and solid solution for this problem, 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-preserving model. We proposed a PDPKS construction, and proved its security and privacy in the

[16] This comparison may be unfair, as the evaluation of the PDPKS has considered the latest results in cryptanalysis, while that of ECDSA does not.

random oracle model. On the functionality, anyone can derive an arbitrary number of fresh public verification keys from a user's long-term public key, without interactions with the key owner, while only the key owner can generate the corresponding signing keys from his long-term secret key. On the privacy, the derived verification keys and corresponding signatures do not leak any information that can be linked to the original long-term public key. On the security, the derived keys are independent/insulated from each other, namely, for any specific derived public verification key, even if an adversary corrupts all other derived signing keys, the adversary cannot forge a valid signature with respect to it. With these functionality, security, and privacy-preserving features, PDPKS could be a convenient and secure cryptographic tool for building privacy-preserving cryptocurrencies. Particularly, the proposed PDPKS construction can be used to fix the identified security vulnerability in Monero, and also provides a robust solution for implementing the so-called stealth addresses for cryptocurrencies. It can also be used to fix a similar vulnerability in the Deterministic Wallet algorithm of Bitcoin.

References

1. Backes, M., Hanzlik, L., Kluczniak, K., Schneider, J.: Signatures with flexible public key: A unified approach to privacy-preserving signatures (full version). IACR Cryptology ePrint Archive (2018), `http://eprint.iacr.org/2018/191`
2. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. IACR Cryptology ePrint Archive (2017), `http://eprint.iacr.org/2017/334`
3. Barreto, P.S.L.M., Libert, B., McCullagh, N., Quisquater, J.: Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. In: ASIACRYPT 2005. pp. 515–532 (2005)
4. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: SAC 2005. pp. 319–331 (2005)
5. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: ASIACRYPT 2001. pp. 566–582 (2001)
6. Bellare, M., Namprempre, C., Neven, G.: Security proofs for identity-based identification and signature schemes. J. Cryptology 22(1), 1–61 (2009)
7. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy, SP 2014. pp. 459–474 (2014)
8. Boneh, D., Boyen, X.: Short signatures without random oracles. In: EUROCRYPT 2004. pp. 56–73 (2004)
9. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. In: CRYPTO 2001. pp. 213–229 (2001)
10. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. J. Cryptology 17(4), 297–319 (2004)

11. Boyen, X., Waters, B.: Anonymous hierarchical identity-based encryption (without random oracles). In: CRYPTO 2006. pp. 290–307 (2006)
12. Buterin, V.: Deterministic wallets, their advantages and their understated flaws. Bitcoin Magazine (November 2013), http://bitcoinmagazine.com/8396/deterministic-wallets-advantages-flaw/
13. Cha, J.C., Cheon, J.H.: An identity-based signature from gap diffie-hellman groups. In: PKC 2003. pp. 18–30 (2003)
14. Chaum, D.: Blind signatures for untraceable payments. In: CRYPTO '82. pp. 199–203 (1982)
15. Chaum, D., van Heyst, E.: Group signatures. In: EUROCRYPT '91. pp. 257–265 (1991)
16. CoinMarketCap: Top 100 cryptocurrencies by market capitalization, https://coinmarketcap.com. Accessed 4 October 2018
17. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (1976)
18. Dodis, Y., Katz, J., Xu, S., Yung, M.: Key-insulated public key cryptosystems. In: EUROCRYPT 2002. pp. 65–82 (2002)
19. Dodis, Y., Katz, J., Xu, S., Yung, M.: Strong key-insulated signature schemes. In: PKC 2003. pp. 130–144 (2003)
20. Electrum.org: Electrum lightweight bitcoin wallet (November 2011), https://electrum.org
21. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics 156(16), 3113–3121 (2008)
22. Gutoski, G., Stebila, D.: Hierarchical deterministic bitcoin wallets that tolerate key leakage. In: FC 2015. pp. 497–504 (2015)
23. Hess, F.: Efficient identity based signature schemes based on pairings. In: SAC 2002. pp. 310–324 (2002)
24. Khandaker, M.A., Nanjo, Y., Ghammam, L., Duquesne, S., Nogami, Y., Kodera, Y.: Efficient optimal ate pairing at 128-bit security level. IACR Cryptology ePrint Archive (2017), http://eprint.iacr.org/2017/1174
25. Kim, T., Barbulescu, R.: Extended tower number field sieve: A new complexity for the medium prime case. In: CRYPTO 2016, Part I. pp. 543–571 (2016)
26. Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In: ACISP 2004. pp. 325–335 (2004)
27. Liu, Z., Wong, D.S., Nguyen, K., Yang, G., Wang, H.: Abelian Coin (ABE) - A Quantum-Resistant Cryptocurrency Balancing Privacy and Accountability. Abelian Foundation, available at https://www.abelianfoundation.org/wp-content/uploads/2018/08/Abelian-Whitepaper-CB20180615.pdf.
28. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008), http://bitcoin.org/bitcoin.pdf
29. Narayanan, A., Bonneau, J., Felten, E.W., Miller, A., Goldfeder, S.: Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction. Princeton University Press (2016)
30. National Institute of Standards and Technology (NIST): Recommendation for key management, part 1: General (revised) (July 2012), https://csrc.nist.gov/publications/detail/sp/800-57-part-1/revised/archive/2007-03-01
31. NIST: Fips pub 186-4, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf. Accessed 2 June 2018

32. Noether, S., Mackenzie, A.: Ring confidential transactions. Ledger, vol. 1, pp. 1-18 (2016)
33. Okamoto, T., Ohta, K.: Universal electronic cash. In: CRYPTO '91. pp. 324–337 (1991)
34. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO '91. pp. 129–140 (1991)
35. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2), 120–126 (1978)
36. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: ASIACRYPT 2001. pp. 552–565 (2001)
37. van Saberhagen, N.: Cryptonote v 2.0 (2013), `https://cryptonote.org/whitepaper.pdf`
38. Shamir, A.: Identity-based cryptosystems and signature schemes. In: CRYPTO '84. pp. 47–53 (1984)
39. Standards for Efficient Cryptography Group: Sec 2: Recommended elliptic curve domain parameters (2010), `http://www.secg.org/sec2-v2.pdf`
40. Todd, P.: Stealth addresses, `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-January/004020.html`
41. Waters, B.: Efficient identity-based encryption without random oracles. In: EUROCRYPT 2005. pp. 114–127 (2005)
42. Wuille, P.: Bip32: Hierarchical deterministic wallets, `https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki`. Accessed 4 October 2018

A Proof of Theorem 1

Proof. Let Π be a PDPKS scheme, and Π is public key unlinkable. Below we prove that Π is derived verification key unlinkable.

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

Setup. $\mathcal{B}$ is given PP, PK_0 and PK_1, then $\mathcal{B}$ forwards PP, PK_0 and PK_1 to $\mathcal{A}$.

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

Challenge. $\mathcal{B}$ receives DVK^*, which is derived from PK_b.

$\mathcal{B}$ chooses a random $c \in \{0,1\}$, and computes $\mathsf{DVK}_0^* \leftarrow \mathsf{VrfyKeyDerive}(\mathsf{PK}_c, \mathsf{PP})$.

$\mathcal{B}$ sets $\mathsf{DVK}_1^* = \mathsf{DVK}^*$, and returns $(\mathsf{DVK}_0^*, \mathsf{DVK}_1^*)$ to $\mathcal{A}$.

$\mathcal{B}$ sets $L_{dvk} = L_{dvk} \cup \{(\mathsf{DVK}_0^*, \mathsf{PK}_c), (\mathsf{DVK}_1^*, \top)\}$, where $\top$ is a special symbol to denote the public key PK_b where b is unknown to $\mathcal{B}$.

Phase 2. Same as **Phase 1**. Note that $\mathcal{A}$ is not allowed to query $\mathsf{ODVKAdd}(\mathsf{DVK}_j^*, \mathsf{PK}_i)$ (for $j, i \in \{0,1\}$) or $\mathsf{ODSKCorrput}(\mathsf{DVK}_j^*)$ (for $j \in \{0,1\}$), when $\mathcal{B}$ forwards $\mathcal{A}$'s queries to $\mathcal{C}$ and forwards the results to $\mathcal{A}$, from $\mathcal{C}$'s point of view, $\mathcal{B}$ does not make queries $\mathsf{ODVKAdd}(\mathsf{DVK}^*, \mathsf{PK}_i)$ (for $i \in \{0,1\}$) or $\mathsf{ODSKCorrput}(\mathsf{DVK}^*)$.

Guess. $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, $\mathcal{B}$ sets $b'' = b' \oplus c$ and returns b'' to $\mathcal{C}$. Note that

- $b' = 0$ implies $\mathcal{A}$ is guessing that DVK_0^* and DVK_1^* are derived from the same public key, and this implies that DVK^* is also derived from PK_c. Thus, if $c = 0$, $\mathcal{B}$ sets $b'' = 0$, otherwise $\mathcal{B}$ sets $b'' = 1$. This means $\mathcal{B}$ sets $b'' = b' \oplus c$.
- $b' = 1$ implies $\mathcal{A}$ is guessing that DVK_0^* and DVK_1^* are derived from different public keys, and this implies that DVK^* is derived from PK_{1-c}. Thus, if $c = 0$, $\mathcal{B}$ sets $b'' = 1$, otherwise $\mathcal{B}$ sets $b'' = 0$. This means $\mathcal{B}$ sets $b'' = b' \oplus c$.

The advantage of $\mathcal{A}$ in $\mathsf{Game}_{\mathsf{DVKUNL}}$ is

$$\begin{aligned} Adv_{\mathcal{A}}^{dvkunl} &= |\Pr[b' = 0 | c = b] + \Pr[b' = 1 | c = 1 - b] - 1/2| \\ &= |\Pr[b' = c \oplus b] - 1/2| = |\Pr[b = c \oplus b'] - 1/2|. \end{aligned}$$

Note that the advantage of $\mathcal{B}$ in $\mathsf{Game}_{\mathsf{PKUNL}}$ is

$$Adv_{\mathcal{B}}^{pkunl} = |\Pr[b'' = b] - 1/2| = |\Pr[b' \oplus c = b] - 1/2|,$$

we have $Adv_{\mathcal{B}}^{pkunl} = Adv_{\mathcal{A}}^{dvkunl}$.

The proof for public key strong unlinkability implying derived verification key strong unlinkability is similar, except that **Phase 2** changes as below.

Phase 2. At the begin of **Phase 2**, $\mathcal{B}$ makes query $\mathsf{ODVKAdd}(\mathsf{DVK}_0^*, \mathsf{PK}_c)$ to $\mathcal{C}$. Note that DVK_0^* is honestly derived from PK_c, $\mathcal{C}$ will return 1 to $\mathcal{B}$, implying DVK_0^* is valid.

Then, when $\mathcal{A}$ makes query to oracles $\mathsf{ODVKAdd}(\cdot,\cdot)$, $\mathsf{ODSKCorrput}(\cdot)$, $\mathsf{OSign}(\cdot,\cdot)$, $\mathcal{B}$ makes the same query to $\mathcal{C}$, and forwards the results to $\mathcal{A}$.

Note that $\mathcal{A}$ is not allowed to query $\mathsf{ODVKAdd}(\mathsf{DVK}_j^*, \mathsf{PK}_i)$ (for $j, i \in \{0,1\}$), when $\mathcal{B}$ forwards $\mathcal{A}$'s $\mathsf{ODVKAdd}()$ queries to $\mathcal{C}$, from $\mathcal{C}$'s point of view, $\mathcal{B}$ does not make queries $\mathsf{ODVKAdd}(\mathsf{DVK}^*, \mathsf{PK}_i)$ (for $i \in \{0,1\}$).

For derived verification key strong unlinkability, $\mathcal{A}$ is allowed to make queries $\mathsf{ODSKCorrput}(\mathsf{DVK}_j^*)$ (for $j \in \{0,1\}$). When $\mathcal{B}$ forwards $\mathcal{A}$'s $\mathsf{ODSKCorrput}(\mathsf{DVK}_0^*)$ query to $\mathcal{C}$, from $\mathcal{C}$'s point of view, it is a valid query since DVK_0^* has been checked valid by $\mathcal{C}$ at the begin of **Phase 2**. When $\mathcal{B}$ forwards $\mathcal{A}$'s $\mathsf{ODSKCorrput}(\mathsf{DVK}_1^*)$ query to $\mathcal{C}$, from $\mathcal{C}$'s point of view, it is a valid query since $\mathcal{B}$ is actually making $\mathsf{ODSKCorrput}()$ query on the challenge derived verification key DVK^*, which is allowed in the model for public key strong unlinkability.

B Another PDPKS Construction

In this section, we give another PDPKS construction as well as the proofs for its security and privacy.

B.1 Construction

This PDPKS construction and the underlying CDH assumption are on the bilinear map groups where $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$, $P = Q$, and ψ is the identity map.

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

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

  and the message space is $\mathcal{M} = \{0,1\}^*$.
- $\mathsf{KeyGen}(\mathsf{PP}) \to (\mathsf{PK}, \mathsf{SK})$. The algorithm chooses random $\alpha, \beta \in \mathbb{Z}_p^*$, then outputs a public key PK and corresponding secret key SK as

$$\mathsf{PK} := (A, B) = (\alpha P, \beta P) \in \mathbb{G} \times \mathbb{G},$$
$$\mathsf{SK} := (\alpha, \beta) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*.$$

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

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

– $\mathsf{VrfyKeyCheck}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP}) \to 1/0$. Upon input $\mathsf{DVK} = (R, T_{vk}) \in \mathbb{G} \times \mathbb{G}_T$, $\mathsf{PK} = (A, B) \in \mathbb{G} \times \mathbb{G}$, $\mathsf{SK} = (\alpha, \beta) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and the system public parameters PP, the algorithm checks whether $T_{vk} \stackrel{?}{=} e(H_1(R, \alpha R), -B)$. If it hods, the algorithm outputs 1, otherwise outputs 0.

– $\mathsf{SignKeyDerive}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP}) \to \mathsf{DSK}$ or $\perp$. Upon input $\mathsf{DVK} = (R, T_{vk}) \in \mathbb{G} \times \mathbb{G}_T$, $\mathsf{PK} = (A, B) \in \mathbb{G} \times \mathbb{G}$, $\mathsf{SK} = (\alpha, \beta) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and the system public parameters PP, the algorithm checks whether $T_{vk} \stackrel{?}{=} e(H_1(R, \alpha R), -B)$. If it hods, the algorithm outputs a derived signing key

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

otherwise, outputs $\perp$.

– $\mathsf{Sign}(m, \mathsf{DVK}, \mathsf{DSK}, \mathsf{PP}) \to \sigma$. Upon input a message $m \in \mathcal{M}$, a derived verification key $\mathsf{DVK} = (R, T_{vk}) \in \mathbb{G} \times \mathbb{G}_T$, a signing key $\mathsf{DSK} = S_{sk} \in \mathbb{G}$, and the system public parameters PP, 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 $P_\sigma = hS_{sk} + xP_1 \in \mathbb{G}$,

and outputs $\sigma = (h, P_\sigma)$ as a signature for m.

– $\mathsf{Verify}(m, \sigma, \mathsf{DVK}, \mathsf{PP}) \to 1/0$. Upon 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 $\mathsf{DVK} = (R, T_{vk}) \in \mathbb{G} \times \mathbb{G}_T$, 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) $\mathsf{VrfyKeyCheck}(\mathsf{DVK}, \mathsf{PK}, \mathsf{SK}, \mathsf{PP}) = 1$, since $\alpha R = \alpha r P = rA$, and

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

$$\begin{aligned} e(P_\sigma, P) \cdot (T_{vk})^h &= e(hS_{sk} + xP_1, P) \cdot e(H_1(rP, rA), -B)^h \\ &= e(h\beta H_1(R, \alpha R), P) \cdot e(xP_1, P) \cdot e(H_1(rP, rA), -B)^h \\ &= e(H_1(R, \alpha R), \beta P)^h \cdot e(P_1, P)^x \cdot e(H_1(rP, rA), -B)^h \\ &= X. \end{aligned}$$

B.2 Proof of Security

We prove the security of the above PDPKS construction by a reduction to the security of IBS construction in [23, Section 2]. First, in Appendix B.2.1 we review the IBS construction and its security conclusion, then in Appendix B.2.2 we give the reduction from the security of our PDPKS construction to the security of the IBS construction.

B.2.1 Review of IBS in [23, Section 2]

Construction of the IBS in [23, Section 2]

Below is the IBS construction in [23, Section 2]. [17]

- $\mathsf{Setup}(\lambda) \to (\mathsf{PP}, \mathsf{MSK})$. Upon input a security parameter λ, 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 $H_1 : \{0,1\}^* \to \mathbb{G}^*$, $H_2 : \{0,1\}^* \times \mathbb{G}_T \to \mathbb{Z}_p^*$, where $\mathbb{G}^* = \mathbb{G} \setminus \{0\}$. 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 $\mathsf{PP} := (p, (\mathbb{G}, \mathbb{G}_T, e), P, B, H_1, H_2),\ \mathsf{MSK} := \beta$.

  The message space is $\mathcal{M} = \{0,1\}^*$.
- $\mathsf{KeyExtract}(\mathsf{ID}, \mathsf{PP}, \mathsf{MSK}) \to \mathsf{SK}_{\mathsf{ID}}$. Upon input an arbitrary identity $\mathsf{ID} \in \{0,1\}^*$, the system public parameters PP, and the master secret key MSK, the algorithm outputs a private key $\mathsf{SK}_{\mathsf{ID}}$ for the identity ID as $\mathsf{SK}_{\mathsf{ID}} = \beta H_1(\mathsf{ID}) \in \mathbb{G}$.
- $\mathsf{Sign}(m, \mathsf{ID}, \mathsf{PP}, \mathsf{SK}_{\mathsf{ID}}) \to \sigma$. Upon input a message $m \in \{0,1\}^*$, an identity $\mathsf{ID} \in \{0,1\}^*$, the system public parameters PP, and a private key $\mathsf{SK}_{\mathsf{ID}}$ 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 $P_\sigma = h\mathsf{SK}_{\mathsf{ID}} + xP_1 \in \mathbb{G}$,

  and outputs $\sigma = (h, P_\sigma)$ as a signature for message m and identity ID.
- $\mathsf{Verify}(m, \sigma, \mathsf{ID}, \mathsf{PP}) \to 1/0$. Upon input a message $m \in \{0,1\}^*$, a signature $\sigma = (h, P_\sigma) \in \mathbb{Z}_p^* \times \mathbb{G}$, an identity $\mathsf{ID} \in \{0,1\}^*$, and the system public parameters PP, the algorithm outputs $b = 1$ if and only if $h = H_2(m, e(P_\sigma, P) \cdot e(H_1(\mathsf{ID}), -B)^h)$.

[17] Note that we slightly changed the variable names in the IBS construction, to better suit our PDPKS construction in later proof.

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

$$\begin{aligned}\mathsf{CMPK} :=& (p, (\mathbb{G}, \mathbb{G}_T, e), P, H_1, H_2),\\ \mathsf{IMPK} :=& (B),\\ F(\mathsf{PP}, \mathsf{ID}) :=& e(H_1(\mathsf{ID}), -B).\end{aligned}$$

Security of the IBS in [23, Section 2]

The security of the IBS construction in [23, Section 2] is established by the following lemma.

Lemma 3. *[23, Theorem 1] In the random oracle model, suppose that an adaptive adversary $\mathcal{A}$ which makes at most $n_1 \geq 1$ queries of an identity hash and extraction oracle, at most $n_2 \geq 1$ queries of a message hash and signing oracle and which succeeds within time $t_{\mathcal{A}}$ of making an existential forgery with probability $\epsilon_{\mathcal{A}} \geq \frac{an_1n_2^2}{p}$ for some constant $a \in \mathbb{Z}^{\geq 1}$. Then there is another probabilistic algorithm $\mathcal{C}$ and a constant $c \in \mathbb{Z}^{\geq 1}$ such that $\mathcal{C}$ solves the CDH problem in expected time $t_{\mathcal{C}} \leq \frac{cn_1n_2t_{\mathcal{A}}}{\epsilon_{\mathcal{A}}}$.*

B.2.2 Security Proof of our PDPKS Construction

Now we prove the security of our PDPKS construction in Appendix B.1, by a reduction to the IBS construction in [23, Section 2], as shown in the following Lemma 4.

Lemma 4. *Assume that there exists an adaptively chosen message attacker $\mathcal{A}$ that 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 in $\mathsf{Game}_{\mathsf{UEF}}$ for our PDPKS construction. Assume that, within a time t, $\mathcal{A}$ produces a forgery with probability ϵ. Then, there exists an algorithm $\mathcal{B}$ that is able to produce within time $\bar{t} = t + O(q_a\tau_{mult}) + O(q_a\tau_p)$ a forgery with probability $\bar{\epsilon} = \epsilon$ in $\mathsf{Game}_{\mathsf{IBS},\mathsf{UEF}}$ for the IBS construction in [23, Section 2], where $\mathcal{B}$ makes $\bar{q}_{h_i}$ queries to random oracles $H_i (i = 1, 2)$ and $\bar{q}_s$ queries to the signing oracle, with $\bar{q}_{h_1} \leq q_{h_1} + q_a, \bar{q}_{h_2} \leq q_{h_2}, \bar{q}_s \leq q_s$.*

Proof. The reduction is similar to that in Lemma 2, namely, using tuple (rP, rA) as the identity for the IBS construction. We omit the details here.

Theorem 4. *The PDPKS scheme in Appendix B.1 is secure under the CDH assumption in the random oracle model.*

Proof. This follows Lemma 3 and Lemma 4 immediately.

B.3 Proof of Privacy

Now we prove that our PDPKS construction in Appendix B.1 is public key strongly unlinkable (w.r.t. Def. 3).

Theorem 5. *The PDPKS scheme in Appendix B.1 is public key strongly 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_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 $\mathsf{Game}_{\mathsf{PKSUNL}}$ with advantage ϵ, then there exists an algorithm $\mathcal{B}$ that runs within time $\bar{t} = t + O((q_{h_1} + q_s)\tau_{mult}) + O((q_a q_{h_1} + q_s)\tau_p)$, and solves the CDH problem with probability at least $\epsilon - q_a/p$.*

Proof. Similar to the proof of Theorem 3, if there exists a PPT adversary $\mathcal{A}$ that can win $\mathsf{Game}_{\mathsf{PKSUNL}}$ 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 $\mathsf{PK}_0 := (A^{(0)}, B^{(0)}), \mathsf{PK}_1 := (A^{(1)}, B^{(1)})$ to $\mathcal{A}$. Note that the secret keys corresponding to PK_0 and PK_1 are $\mathsf{SK}_0 := (\alpha'_0 a, \beta_0)$ and $\mathsf{SK}_1 := (\alpha'_1 a, \beta_1)$ respectively, where $\mathcal{B}$ does not know the value of a.

Note that $\mathcal{B}$ knows the values of β_0 and β_1, so that it is able to answer $\mathcal{A}$'s queries to the $\mathsf{ODVKAdd}(\cdot, \cdot)$, $\mathsf{ODSKCorrput}(\cdot)$, $\mathsf{OSign}(\cdot, \cdot)$ 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 $\mathsf{DVK}^* = (R^*, T^*_{vk})$ from PK_{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 $\mathsf{DVK}^* = (R^*, T^*_{vk})$ to $\mathcal{A}$.

3. $\mathcal{B}$ sets $\mathsf{DSK}^* = (\beta_{i^*} hval^*)P$ and adds $(\mathsf{DVK}^*, \mathsf{DSK}^*, 0, \tilde{B}, \top, i^*)$ to L_{dvk}.

The rest of the proof and analysis can follow those of Theorem 3 and we omit the details here.

C An IBS without the MPK-pack-able Property

Below we review the IBS construction in [13], and show that it does not have the MPK-pack-able Property. [18]

The IBS Scheme in [13]

- $\mathsf{Setup}(\lambda) \to (\mathsf{PP}, \mathsf{MSK})$. Upon input a security parameter λ, 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 $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \{0,1\}^* \times \mathbb{G} \to \mathbb{Z}_p$. 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 $\mathsf{PP} := (p, (\mathbb{G}, \mathbb{G}_T, e), P, B, H_1, H_2)$, $\mathsf{MSK} := \beta$.

  The message space is $\mathcal{M} = \{0,1\}^*$.
- $\mathsf{KeyExtract}(\mathsf{ID}, \mathsf{PP}, \mathsf{MSK}) \to \mathsf{SK}_{\mathsf{ID}}$. Upon input an arbitrary identity $\mathsf{ID} \in \{0,1\}^*$, the system public parameters PP, and the master secret key MSK, the algorithm outputs a private key $\mathsf{SK}_{\mathsf{ID}}$ for the identity ID as $\mathsf{SK}_{\mathsf{ID}} = \beta H_1(\mathsf{ID}) \in \mathbb{G}$.
- $\mathsf{Sign}(m, \mathsf{ID}, \mathsf{PP}, \mathsf{SK}_{\mathsf{ID}}) \to \sigma$. Upon input a message $m \in \{0,1\}^*$, an identity $\mathsf{ID} \in \{0,1\}^*$, the system public parameters PP, and a private key $\mathsf{SK}_{\mathsf{ID}}$ for the identity ID, the algorithm
  1. picks a random $r \in \mathbb{Z}_p$, and computes $U = rH_1(\mathsf{ID}) \in \mathbb{G}$,
  2. sets $h = H_2(m, U) \in \mathbb{Z}_p$,
  3. computes $V = (r + h)\mathsf{SK}_{\mathsf{ID}} \in \mathbb{G}$,

  and outputs $\sigma = (U, V)$ as a signature for message m and identity ID.
- $\mathsf{Verify}(m, \sigma, \mathsf{ID}, \mathsf{PP}) \to 1/0$. Upon input a message $m \in \{0,1\}^*$, a signature $\sigma = (U, V) \in \mathbb{G} \times \mathbb{G}$, an identity $\mathsf{ID} \in \{0,1\}^*$, and the system public parameters PP, the algorithm outputs $b = 1$ if and only if $e(P, V) = e(B, U + H_2(m, U)H_1(\mathsf{ID}))$.

[18] Note that we slightly changed the variable names in the IBS construction.

The above IBS scheme does not have the MPK-pack-able property. Note that in the above IBS scheme, we have

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

In the verification algorithm, the used values are $P, V, B, U, m, \mathsf{ID}$. For the left side of the equation, as V is a part of the signature, neither V or $e(P, V)$ could be used to define the function F. As P is in CMPK, it is unnecessary to contain P as a component of F's output. For the right side of the equation, as U is a part of the signature, the only possible definitions of F are: (1) $F(\mathsf{PP}, \mathsf{ID}) = B$, or (2) $F(\mathsf{PP}, \mathsf{ID}) = e(B, H_1(\mathsf{ID}))$, or (3) $F(\mathsf{PP}, \mathsf{ID}) = H_1(\mathsf{ID})$.

- For case (1), i.e. $F(\mathsf{PP}, \mathsf{ID}) = B$: The output of F leaks the value of B which identifies IMPK.
- For case (2), i.e. $F(\mathsf{PP}, \mathsf{ID}) = e(B, H_1(\mathsf{ID}))$: To verify the signature, $e(B, U)$ has to be computed where B is used. This implies that there is no Verify_F such that $\mathsf{Verify}_F(\mathsf{CMPK}, F(\mathsf{PP}, \mathsf{ID}), M, \sigma) = \mathsf{IBS.Verify}(\mathsf{PP}, \mathsf{ID}, M, \sigma)$.
- For case (3), i.e. $F(\mathsf{PP}, \mathsf{ID}) = H_1(\mathsf{ID})$: The same to Case (2).

Thus, it concludes that the above IBS scheme does not have the MPK-pack-able property.