Certificateless Public Key Signature Schemes from Standard Algorithms

(Expanded Version)

Zhaohui Cheng[1] and Liqun Chen[2]

[1] Olym Info. Sec. Tech. Ltd.
chengzh@myibc.net
[2] Surrey University
liqun.chen@surrey.ac.uk

Abstract

Certificateless public key cryptography (CL-PKC) is designed to have succinct public key management without using certificates at the same time avoid the key-escrow attribute in the identity-based cryptography. However, it appears difficult to construct CL-PKC schemes from standard algorithms. Security mechanisms employing self-certified key (also known as implicit certificate) can achieve same goals. But there still lacks rigorous security definitions for implicit-certificate-based mechanisms and such type of schemes were not analyzed formally and often found vulnerable to attacks later.

In this work, we first unify the security notions of these two types of mechanisms within an extended CL-PKC formulation. We then present a general key-pair generation algorithm for CL-PKC schemes and use it with the key prefixing technique to construct certificateless public key signature (CL-PKS) schemes from standard algorithms. The security of the schemes is analyzed within the new model, and it shows that the applied technique helps defeat known-attacks against existing constructions.

The resulting schemes could be quickly deployed based on the existing standard algorithm implementations. They are particularly useful in the Internet of Things (IoT) to provide security services such as entity authentication, data integrity and non-repudiation because of their low computation cost, bandwidth consumption and storage requirement.

Contents

1 Introduction

In a public key cryptography system, a security mechanism to unequivocally demonstrate the relationship between the public key and the identity of the key's owner is indispensable. In the public key infrastructure (PKI) system, the authority issues a certificate to bind a user's identity with his public key. While the solution is well-established and universal, the PKI system can be very complicated and faces many challenges in practice, such as the efficiency and scalability of the system. The identity-based cryptography (IBC) offers an attractive alternative. In an IBC system, a user treats his identity as his public key, or more accurately everyone can derive a user's public key from his identity string through a pre-defined function with a set of system parameters. Hence, in such a system, the public key authenticity problem becomes trivial, and certificates are no longer necessary. However, the key generation center (KGC) can generate the private key corresponding to any identity in an IBC system. This key-escrow function sometimes causes concerns about users' privacy. Moreover, the compromise of the KGC resulting in leaking the master secret could be a disastrous event.

In 2003, Al-Riyami and Paterson introduced a new paradigm: the certificateless public key cryptography (CL-PKC) [1]. CL-PKC is designed to have succinct public key management without certificates at the same time remove the key-escrow property embedded in the IBC. In CL-PKC, a user has a public key, and his private key is determined by two pieces of secrets: one secret associated with the user's identity is extracted from the KGC, and the other is generated by the user himself. Moreover, one secret is not computable from the other, so the KGC cannot compute the user's private key. Hence, CL-PKC is key-escrow free. The approach against the public key replacement attack in CL-PKC is not to directly prove the authenticity of a public key with a certificate. Instead, CL-PKC guarantees that even if a malicious user successfully replaces a target's public key with his own choice, he still cannot generate a valid signature or decrypt a ciphertext generated with the false public key and the target's identity. This effect will undoubtedly reduce the interest of launching the attack.

CL-PKC includes Certificateless Public Key Encryption (CL-PKE) and Certificateless Public Key Signatures (CL-PKS). In this paper, we focus on signature schemes. As shown in Table 1, there are many schemes in this category. All of the existing unbroken CL-PKS schemes use custom signature algorithms while those using standard algorithms, such as ECDSA or Schnorr, have been broken or is vulnerable to certain attack. In this work, we introduce several CL-PKS schemes, which use standard signature schemes, and analyze their security.

1.1 Related Work

In the literature, there are many publications on CL-PKC either presenting concrete schemes or researching the formal models of related security notions. A short and incomplete list includes [1, 2, 6, 8, 18, 28, 29, 30, 27, 34, 35, 36, 37, 48, 51, 52]. In practice, many products have implemented standard cryptographic schemes. If the CL-PKC constructions can reuse these existing infrastructures, it will certainly help facilitate the adoption of CL-PKC-based security solutions. However, it appears difficult to construct secure CL-PKC schemes upon standard algorithms under current

Table 1: List of Some CL-PKS Schemes

Scheme	Security status	From standard alg.
AP[1]	broken[31]	No
CPHL[19]	proof*	No
HMSWW[30]	proof*	No
ZWXF[52]	proof*	No
ZZZ[53]	proof*	No
HRL[27]	no proof*,$	No
HCZ[29]	broken[49]	No
JHLC[34]	proof*	No
LXWHH[37]	proof*	No
YSCC[51]	broken[34]	Schnorr
PH/OMC+ECDSA[42, 14]#	broken[14]	ECDSA
Arazi/ECQV+ECDSA[5, 17]#	attack[14]	ECDSA
CL-PKS1#	partial proof	ECDSA
CL-PKS2#	proof	Enhanced ECDSA
CL-PKS3#	proof	Schnorr

* By Remark 1, these schemes do not satisfy the CL-PKS security notion in Definition 2 in this work.

$ The scheme does not strictly follow the Al-Riyami-Paterson formulation.

The schemes fit in with the extended CL-PKS formulation in this work.

CL-PKC formulation. For example, there are schemes such as [6, 27, 28, 29, 34, 35, 37, 48, 51] that do not require pairing, but none of the unbroken CL-PKS algorithms is constructed upon standard algorithms such as ECDSA [32], SM2 [21].

For similar security purpose, another line of work named "self-certified keys" or "implicit certificate" [23, 42, 5, 43, 15] had been developed before the birth of CL-PKC. In 1991, Girault brought forth the notion of "self-certified" public key [23] and constructed two schemes with RSA/Rabin signature and El-Gamal signature. The schemes were used with an identification protocol and a key exchange protocol. In 1997, Petersen and Horster presented self-certified keys based on the Schnorr signature and proposed to use such keys with standard algorithms [42]. In 1998, Arazi proposed a modified Schnorr signature to generate "certification" of keys [5]. Arazi also described how to use the the generated "certificate" of keys with the standardized algorithms. In 2000, Pintsov and Vanstone [43] proposed an "implicit certificate" scheme, called the Optimal Mail Certificate (OMC) scheme, which is similar to the Petersen-Horster scheme (we refer to both schemes as OMC hereafter). The OMC scheme was combined with the Pintsov-Vanstone signature to form a partial message recovery signature. In 2001, Brown, Gallant and Vanstone [15] described a modification of the OMC scheme, which is similar to Arazi's key generation algorithm. This scheme later became known as the elliptic curve Qu-Vanstone (ECQV) implicit certificate scheme [17]. ECQV has found its applications in the Internet of Things (IoT). For example, it becomes part of the cryptographic suite building blocks in the ZigBee smart energy standard [54]. Brecht et al. have proposed using ECQV with ECDSA in the vehicle to everything (V2X) applications [16].

As specified in [17], an implicit certificate is comprised of a user's identity, the public key reconstruction data and some extension fields which, together with KGC's public key, are used to generate the user's public key. The public key reconstruction data is same as user's public value in [5, 42]. Hence, an implicit certificate is a form of representing user's identity and self-certified key. Hereafter, we refer to both types of mechanisms as implicit certificate for convenience.

After many years of development, there still lacks a systematic treatment of the security notions of the implicit certificate and the security mechanisms using it. In [15], Brown et al. presented an implicit certificate security model, which however does not fully address the impact of a malicious KGC. Moreover, a direct composition of a sound implicit certificate with a standard mechanism such as a secure signature does not always result in a scheme to achieve the intended security properties. For example, OMC+ECDSA is completely broken and there is an attack against ECQV+ECDSA [14]. Hence, only a security definition of implicit certificate schemes is inadequate, and it's important to formulate security notions for the full implicit-certificate-based security mechanisms and so be able to formally analyze the security of this type of mechanisms.

It's worth mentioning that Groves developed an elliptic curve-based certificateless signature named ECCSI [24] "by drawing on ideas set out by Arazi." ECCSI does not allow a user to generate his secret. Hence it is more like an identity-based signature (IBS) because it still maintains the key-escrow attribute.

1.2 Our Contributions

In this work, we first extend the Al-Riyami-Paterson formulation of CL-PKC and define a unified model to cover both implicit-certificate-based mechanisms and CL-PKC schemes following Al-Riyami-Paterson's definition. We define the security notions for both signature and encryption primitives. Under the new model, we can use implicit certificate schemes to construct CL-PKC schemes and systematically analyze their security. Second, we present a certificateless key generation algorithm (CL-KGA) based on the Petersen-Horster scheme [42], and we formally analyze its security. The scheme could have better performance than ECQV if point scalar pre-computation is available. Third, we apply the key prefixing technique [39] to combine the proposed CL-KGA with several standard algorithms to construct CL-PKS schemes, and we analyze their security in our new security model. Because of the possibility of pre-computing point scalar operation in practical implementations, the new schemes could be more efficient than the related ones using ECQV.

1.3 Paper Organization

The paper is organized as follows. In Section 2, we redefine the formulation of CL-PKC and security notions of signature and encryption. Then, we present a CL-KGA and formally analyze its security in Section 3. We apply the key prefixing technique [39] to construct CL-PKS schemes by combining the presented CL-KGA and standard algorithms including ECDSA, Schnorr, etc. in Section 4 and analyze their security in Section 5. The performance of the proposed schemes is compared with the related ones in the literature and an implementation on an ARM chip is reported in Section 6. Finally, we draw a conclusion.

2 CL-PKC Definition

2.1 CL-PKC Formulation

In this section, we revisit the Al-Riyami-Paterson definition of CL-PKC and redefine the formulation of CL-PKS and CL-PKE. Because this type of cryptographic scheme shares a common key generation process (we call it CL-KGA), we define this process first and then describe signature and encryption functions.

Given a security parameter k, a CL-KGA uses following five functions to generate public/private keypairs. The first three functions are probabilistic and the others are deterministic. Function **CL.Setup** and **CL.Extract-Partial-Key** are typically executed by a KGC, which keeps $M_{\mathfrak{sk}}$ confidential.

- $(M_{\mathfrak{pk}}, M_{\mathfrak{sk}}) \leftarrow$ **CL.Setup**(1^k). The output is a master public/secret keypair.
- $(U_A, x_A) \leftarrow$ **CL.Set-User-Key**$(M_{\mathfrak{pk}}, \mathtt{ID}_A)$. $\mathtt{ID}_A \in \{0,1\}^*$ refers to an identity string of entity A; the output is a pair of public/secret values.
- $(W_A, d_A) \leftarrow$ **CL.Extract-Partial-Key**$(M_{\mathfrak{pk}}, M_{\mathfrak{sk}}, \mathtt{ID}_A, U_A)$. The output is a pair of partial public/private keys.
- $s_A \leftarrow$ **CL.Set-Private-Key**$(M_{\mathfrak{pk}}, \mathtt{ID}_A, U_A, x_A, W_A, d_A)$. The output is the private key of entity A.
- $P_A \leftarrow$ **CL.Set-Public-Key**$(M_{\mathfrak{pk}}, \mathtt{ID}_A, U_A, W_A)$. The output is the *claimed* public key of entity A.

The above key generation process is substantially different from the Al-Riyami-Paterson definition [1, 2], in which, two public key values U_A and W_A are not addressed. We replace their **CL.Set-Secret-Value** by **CL.Set-User-Key** to make U_A "visible". We also modify their **CL.Extract-Partial-Key** by specifically adding U_A as input and outputting W_A. Finally, in our definition, these two values are explicitly inputted to **CL.Set-Private-Key** and **CL.Set-Public-Key**, and x_A is excluded from the input to **CL.Set-Public-Key**.

Apparently, **CL.Set-User-Key** can compute any value, which needs x_A and is necessary to generate P_A, and include it in U_A. Hence, any key generation scheme following the Al-Riyami-Paterson definition can be covered by our definition. On the other hand, some schemes such as the ones presented in this work achieve the same goals of CL-PKC but cannot fit with the Al-Riyami-Paterson definition. Specifically, the schemes presented in this work require that **CL.Extract-Partial-Key** makes use of U_A. In [2], Al-Riyami and Paterson elaborated a method to construct Certificate-Based Encryption (CBE) [22] from CL-PKE. It requires to execute **CL.Set-Public-Key** immediately after **CL.Set-Private-Key** and uses P_A as part of $\mathtt{ID}_A$ to invoke **CL.Extract-Partial-Key**. This method essentially sets $U_A = P_A$ and calls **CL.Extract-Partial-Key**$(M_{\mathfrak{pk}}, M_{\mathfrak{sk}}, \mathtt{ID}_A \| U_A, \varnothing)$ with an empty variable $\varnothing$ under our definition. We think this circumventive method, which forces inefficient constructions on many occasions, is unnatural. An example of CL-KGA closely following the Al-Riyami-Paterson formulation is given in Appendix 8.2 for comparison. It shows that there is significant difference between the Al-Riyami-Paterson formulation and our new one.

By removing x_A from the input to **CL.Set-Public-Key**, the KGC can compute P_A after executing **CL.Extract-Partial-Key**. This modification is important to facilitate the security definitions below. We note that in principle **CL.Set-Public-Key** is unnecessary in the new formulation because **CL.Extract-Partial-Key** could return P_A instead of W_A, and we keep this function so that the new formulation is compatible with the existing work in the literature.

Once having generated the keypair, the user should be able to execute **CL.Verify-Key** to check the correctness of it.

- $\{$**valid** or **invalid**$\}\leftarrow$ **CL.Verify-Key**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, s_A). The deterministic function outputs whether $(\mathtt{ID}_A, P_A, s_A)$ is valid with regard to $M_{\mathfrak{pk}}$.

In CL-PKC schemes, another value derived from the identity and the master public key together with P_A is used as the *real* public key. This derivation process is typically specified in the encryption or signature verification function. Here, we explicitly define this process as the **CL.Calculate-Public-Key** function. We think this generalization could present a more distinct view of CL-PKC constructions.

- $O_A \leftarrow$ **CL.Calculate-Public-Key**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A). The deterministic function outputs the *real* public key O_A of entity A.

So both P_A and O_A are treated as the public keys of entity $\mathtt{ID}_A$. P_A (called the public key reconstruction data in the implicit certificate work [14]) is distributed in some way such as through an active directory or as part of a signature or message exchanged in a key establishment protocol, and O_A is computed from $M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, and P_A. O_A is the one used as the real public key of $\mathtt{ID}_A$ in the **CL.Encrypt** or **CL.Verify** or a session key computation function.

If **CL.Verify-Key**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, s_A) returns **valid**, the keypair (O_A, s_A), when used in cryptographic schemes such as encryption or signature, should satisfy the soundness requirement of those types of mechanisms.

Now we are ready to define the CL-PKS and CL-PKE. A CL-PKS scheme is specified by following two functions with the key generation scheme above.

- $\sigma \leftarrow$ **CL.Sign**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, s_A, m). The probabilistic function signs a message m and outputs a signature σ.
- $\{$**valid** or **invalid**$\}\leftarrow$ **CL.Verify**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, m, σ). The deterministic function outputs whether σ is a valid signature of m with respect to $(M_{\mathfrak{pk}}, \mathtt{ID}_A, P_A)$.

A CL-PKE scheme is specified by following two functions together with the key generation scheme above.

- $C \leftarrow$ **CL.Encrypt**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, m). The probabilistic function encrypts a message m with ($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A) and outputs a ciphertext C.
- $\{m$ or $\perp\}\leftarrow$ **CL.Decrypt**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, s_A, C). The deterministic function outputs a plaintext m or a failure symbol $\perp$.

As explained above, our CL-PKC formulation covers constructions following the Al-Riyami-Paterson definition. As shown in the following sections, implicit-certificate-based mechanisms are also embraced by this definition. For example, Appendix 8.1 shows that ECQV fits well in the above formulation as a CL-KGA. It has been demonstrated in [2] that Gentry's CBE can be constructed with the Al-Riyami-Paterson CL-PKE. Our generalized definition obviously works for CBE as well.

2.2 Security Definition

Al-Riyami and Paterson defined the security notion of indistinguishability under the adaptive chosen-ciphertext attack (IND-CCA) of CL-PKE [1]. A serial of work [30, 52] refined the security notion of existential unforgeability against the adaptive chosen-message attack (EUF-CMA) of CL-PKS. The formal security model of certificateless key agreement (CL-KA) can be found in such as [36]. All of these security notions are defined with two games. Game 1 is conducted between a challenger $\mathcal{C}$ and a Type-I adversary $\mathcal{A}_I$ who does not know the master secret key but can replace a user's public key with its choice. This type of adversary simulates those who may impersonate a party by providing others with a false public key. Game 2 is conducted between a challenger $\mathcal{C}$ and a Type-II adversary $\mathcal{A}_{II}$ who knows the master secret key (so every entity's partial private key). This type of adversary simulates a malicious KGC adversary who eavesdrops the communications between its subscribers or may even switch public keys among them. We refer to [1, 52, 36] for further details.

Here, we first introduce a formal security model of CL-KGA which has not been defined in the literature and can also serve as a model for implicit certificate mechanisms.[1] In CL-PKC, a KGC and its users could be opponents to each other, but they work together to generate a keypair for an identity $\mathtt{ID}$ if both behave honestly. Hence, they are in a different security world from the classic signature. On the other hand, we show that one still can make use of the security definition of signature mechanism to address the security requirements of a CL-KGA.

Following Al-Riyami and Paterson's approach, the two games of CL-KGA are depicted in Table 2. In these games, an adversary can access an oracle $\mathcal{O}_{\mathtt{CL}}$ to issue queries adaptively before outputting a keypair ($\mathtt{ID}_*$, P_*, s_*) for test. In both games, query **CL.Get-Public-Key**, **CL.Get-Private-Key** and **CL.Get-User-Key** can be asked. And in Game 1, query **CL.Extract-Partial-Key** can also be asked.

- Query **CL.Extract-Partial-Key**($M_{\mathfrak{pk}}$, $M_{\mathfrak{sk}}$, $\mathtt{ID}_A$, U_A). The oracle follows the function definition to generate W_A and d_A and calls function **CL.Set-Public-Key**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, U_A, W_A) to get P_A. It returns W_A and d_A after recording $(\mathtt{ID}_A, P_A)$ in a set $\mathbb{Q}$. The oracle can build the set $\mathbb{Q}$ because **CL.Set-Public-Key** doesn't need x_A in our CL-KGA formulation.

- Query **CL.Get-Public-Key**($\mathtt{ID}_A$, *bNewKey*). If *bNewKey* is true, the oracle follows function **CL.Set-User-Key**, **CL.Extract-Partial-Key**, **CL.Set-Private-Key**, and **CL.Set-Public-Key** sequentially to generate keys, and it returns P_A after recording all the internal keys as ($\mathtt{ID}_A$, P_A, x_A, s_A) in a set $\mathbb{L}$ and putting P_A in a set $\mathbb{P}$. Otherwise, the oracle returns P_A from the latest record indexed by $\mathtt{ID}_A$ in $\mathbb{L}$.

- Query **CL.Get-Private-Key(**$\mathtt{ID}_A, P_A$**)**. The oracle returns s_A from the record indexed by $(\mathtt{ID}_A, P_A)$ in $\mathbb{L}$ after putting $(\mathtt{ID}_A, P_A)$ in a set $\mathbb{S}_1$

- Query **CL.Get-User-Key(**$\mathtt{ID}_A, P_A$**)**. The oracle returns x_A from the record indexed by $(\mathtt{ID}_A, P_A)$ in $\mathbb{L}$ after putting $(\mathtt{ID}_A, P_A)$ in a set $\mathbb{S}_2$.

[1]In [15], a security model of the implicit certificate mechanism is defined. The model is more like for a key agreement and does not consider the Type-II adversary.

Table 2: The CL-KGA Games

Game 1: Type-I Adversary
1. $(M_{\mathfrak{pk}}, M_{\mathfrak{sk}}) \leftarrow$ **CL.Setup**(1^k). 2. $(\mathtt{ID}_*, P_*, s_*) \leftarrow \mathcal{A}_I^{\mathcal{O}^1_{\mathtt{CL}}}(M_{\mathfrak{pk}})$. 3. succeed if $(\mathtt{ID}_*, P_*) \notin \mathbb{S}_1 \cup \mathbb{Q}$ and **valid**$\leftarrow$**CL.Verify-Key**$(M_{\mathfrak{pk}}, \mathtt{ID}_*, P_*, s_*)$.
Game 2: Type-II Adversary
1. $(M_{\mathfrak{pk}}, M_{\mathfrak{sk}}) \leftarrow$ **CL.Setup**(1^k). 2. $(\mathtt{ID}_*, P_*, s_*) \leftarrow \mathcal{A}_{II}^{\mathcal{O}^2_{\mathtt{CL}}}(M_{\mathfrak{pk}}, M_{\mathfrak{sk}})$. 3. succeed if $P_* \in \mathbb{P}$, $(\mathtt{ID}_*, P_*) \notin \mathbb{S}_1 \cup \mathbb{S}_2$ and **valid**$\leftarrow$**CL.Verify-Key**$(M_{\mathfrak{pk}}, \mathtt{ID}_*, P_*, s_*)$.

In these two games, if no record is found when searching $\mathbb{L}$, the oracle returns an error. To exclude the cases that the adversary can win trivially, **CL.Get-Private-Key(**$\mathtt{ID}_*, P_*$**)** is disallowed in both games, i.e., $(\mathtt{ID}_*, P_*) \notin \mathbb{S}_1$. In Game 1, $(\mathtt{ID}_*, P_*)$ is not allowed in the final test if **CL.Extract-Partial-Key**$(M_{\mathfrak{pk}}$, $M_{\mathfrak{sk}}$, $\mathtt{ID}_*, U_*)$ has been queried for some U_*, and W_* from the query output satisfies $P_* =$ **CL.Set-Public-Key**$(M_{\mathfrak{pk}}$, $\mathtt{ID}_*$, U_*, $W_*)$, i.e., $(\mathtt{ID}_*, P_*) \notin \mathbb{Q}$. In Game 2, **CL.Get-User-Key(**$\mathtt{ID}_*, P_*$**)** is forbidden, i.e., $(\mathtt{ID}_*, P_*) \notin \mathbb{S}_2$, and P_* has to be a public key generated through a query **CL.Get-Public-Key**$(\mathtt{ID}_A$, *true*$)$ for some $\mathtt{ID}_A$, i.e., $P_* \in \mathbb{P}$.

The two games above are defined in such way to respond to CL-PKE and CL-PKS' security requirements of a used CL-KGA. Intuitively, a secure CL-PKE requires that an adversary, who does not know one of x_A and d_A for a valid keypair $(\mathtt{ID}_A$, P_A, $s_A)$, should not be able to decrypt a ciphertext encrypted with $(\mathtt{ID}_A, P_A)$. Following the two-game definition, a Type-I adversary $\mathcal{A}_I$ succeeds in Game 1, if it generates a valid keypair $(\mathtt{ID}_*, P_*, s_*)$ from any $(\mathtt{ID}_*, U_*)$ and **CL.Extract-Partial-Key**$(M_{\mathfrak{pk}}$, $M_{\mathfrak{sk}}$, $\mathtt{ID}_*, U_*)$ has not been queried. A Type-II adversary $\mathcal{A}_{II}$ succeeds in Game 2 if it generates a valid keypair $(\mathtt{ID}_*, P_*, s_*)$ of which P_* is generated by the challenger through **CL.Set-Public-Key** and related functions, and its related secret values x_* and s_* are not disclosed to the adversary. A secure CL-PKE requires that its CL-KGA is safe against these two types of adversaries.

Similarly, a secure CL-PKS at least requires that an adversary, who does not know one of x_A and d_A, should not be able to generate a valid signature with a keypair $(\mathtt{ID}_A, P_A, s_A)$. For non-repudiation, a secure CL-PKS further requires that an adversary should not be able to generate a signature on a message with a pair of keys different from the one obtained through **CL.Extract-Partial-Key**. More formally, an adversary succeeds in Game 1 if it generates two valid keypairs $(\mathtt{ID}_*, P_*, s_*)$ and $(\mathtt{ID}_*, P'_*, s'_*)$ for any chosen $(\mathtt{ID}_*, U_*)$ and **CL.Extract-Partial-Key**$(M_{\mathfrak{pk}}$, $M_{\mathfrak{sk}}$, $\mathtt{ID}_*, U_*)$ has been queried *at most once*. A secure CL-PKS requires its CL-KGA is safe against this type of adversary. This requirement is similar to the strong EUF-CMA notion of a signature scheme [3]. As in a CL-PKE, a CL-PKS requires that its CL-KGA is also secure against Type-II adversaries.

Definition 1 *A CL-KGA is secure if the success probability of both $\mathcal{A}_I$ and $\mathcal{A}_{II}$ in the CL-KGA games is negligible.*

Table 3: The CL-PKS-EUF-CMA Games

Game 1: Type-I Adversary
1. $(M_{\mathfrak{pk}}, M_{\mathfrak{sk}}) \leftarrow$ **CL.Setup**(1^k).
2. $(\mathtt{ID}_*, P_*, m_*, \sigma_*) \leftarrow \mathcal{A}_I^{\mathcal{O}^1_{\mathtt{CL}}}(M_{\mathfrak{pk}})$.
3. succeed if $(\mathtt{ID}_*, P_*) \notin \mathbb{S}_1 \cup \mathbb{Q}$, $(\mathtt{ID}_*, P_*, m_*) \notin \mathbb{M}$ and **valid**$\leftarrow$**CL.Verify**$(M_{\mathfrak{pk}}, \mathtt{ID}_*, P_*, m_*, \sigma_*)$.
Game 2: Type-II Adversary
1. $(M_{\mathfrak{pk}}, M_{\mathfrak{sk}}) \leftarrow$ **CL.Setup**(1^k).
2. $(\mathtt{ID}_*, P_*, m_*, \sigma_*) \leftarrow \mathcal{A}_{II}^{\mathcal{O}^2_{\mathtt{CL}}}(M_{\mathfrak{pk}}, M_{\mathfrak{sk}})$.
3. succeed if $P_* \in \mathbb{P}$, $(\mathtt{ID}_*, P_*) \notin \mathbb{S}_1 \cup \mathbb{S}_2$, $(\mathtt{ID}_*, P_*, m_*) \notin \mathbb{M}$ and **valid**$\leftarrow$**CL.Verify**$(M_{\mathfrak{pk}}, \mathtt{ID}_*, P_*, m_*, \sigma_*)$.

For CL-PKS, we use the security games shown in Table 3 to define the security notion of EUF-CMA. As in the CL-KGA games, query **CL.Get-Public-Key**($\mathtt{ID}_A$, *bNewKey*), **CL.Get-Private-Key**($\mathtt{ID}_A$, P_A) and **CL.Get-User-Key** ($\mathtt{ID}_A$, P_A) can be issued in both games, and in Game 1, query **CL.Extract-Partial-Key**($M_{\mathfrak{pk}}$, $M_{\mathfrak{sk}}$, $\mathtt{ID}_A$, U_A) can also be asked. To enable signature queries, the following extra query is allowed in both games.

- Query **CL.Get-Sign**$(\mathtt{ID}_A, P_A, m)$. The oracle uses the private key s_A from the record indexed by ($\mathtt{ID}_A$, P_A) in $\mathbb{L}$ to sign the message m and returns the signature after recording ($\mathtt{ID}_A$, P_A, m) in a set $\mathbb{M}$. If no private key is found corresponding to P_A belonging to $\mathtt{ID}_A$, return an error.

In the security model of [30, 52], the adversary in Game 1 is allowed to issue another query **CL.Replace-Public-Key**$(\mathtt{ID}_A, P_A)$, which replaces user $\mathtt{ID}_A$'s public key with his choice P_A. This query simulates the attack to forge a signature for a targeted identity but with a faked public key. In this work, we don't use this query because CL-PKS may work as an IBS by sending the public key together with a signature [7]. In this case, a user doesn't publish his key separately. To be able to simulate more usage scenarios, instead we allow the adversary to provide a public key of his choice in **CL.Verify** in the final stage of both games. This arrangement implicitly empowers the adversary to cheat a signature verifier with a faked public key. Adversaries defined by this approach corresponds to the normal (instead of strong) adversaries in [30].

As in the CL-KGA games, same restrictions are applied to allowed queries to avoid trivial cases that the adversary can win. Moreover, **CL.Get-Sign**($\mathtt{ID}_*$, P_*, m_*) is disallowed in both games, which implies ($\mathtt{ID}_*$, P_*, m_*) $\notin \mathbb{M}$, because the some of proposed schemes in this work are not strong EUF-CMA-secure.

Definition 2 *A CL-PKS is secure if the success probability of both $\mathcal{A}_I$ and $\mathcal{A}_{II}$ in the CL-PKS-EUF-CMA games is negligible.*

Remark 1 Definition 2 defines a very strong security notion. It requires that without the help from the KGC, a user of identity $\mathtt{ID}_A$ cannot generate a pair of keys (P_A, s_A)

satisfying **valid**$\leftarrow$ **CL.Verify-Key**$(M_{\mathfrak{pk}},\ \mathtt{ID}_A,\ P_A,\ s_A)$. This attribute is essential for a CL-PKS to provide non-repudiation security service. Any CL-PKS scheme implementing function **CL.Extract-Partial-Key** without including U_A as part of the input will be broken by an adversary in Game 1 as follows: the adversary first issues the **CL.Extract-Partial-Key** query and generates a valid public key P_A and private key s_A following the specification (in this case $(\mathtt{ID}_A, P_A) \notin \mathbb{S}_1 \cup \mathbb{Q}$) and then it produces a signature on any message to win the game.

Here, we also define the security notion for CL-PKE with our new formulation, which may be of independent interest to analyze CL-PKE schemes. We use the standard two-stage games shown in Table 4 to define the IND-CCA security notion of as in [1]. Like the CL-KGA games, query **CL.Get-Public-Key**$(\mathtt{ID}_A,\ \textit{bNewKey})$, **CL.Get-Private-Key**$(\mathtt{ID}_A,\ P_A)$ and **CL.Get-User-Key**$(\mathtt{ID}_A,\ P_A)$ can be issued in both games, and in Game 1, query **CL.Extract-Partial-Key**$(M_{\mathfrak{pk}},\ M_{\mathfrak{sk}},\ \mathtt{ID}_A,\ U_A)$ can also be asked. To enable decryption queries, the following extra query is allowed in both games.

- Query **CL.Decrypt-Message**$(\mathtt{ID}_A, P_A, C)$. The oracle uses the private key s_A from the record indexed by $(\mathtt{ID}_A, P_A)$ in $\mathbb{L}$ to decrypt the ciphertext C and returns the output. If no private key is located with such index, then use the latest private key (if any) belonging to user $\mathtt{ID}_A$ to decrypt C and return the output. The challenger in stage two records $(\mathtt{ID}_A, P_A, C)$ in a set $\mathbb{D}$, which implies that both $\mathcal{A}_{I-2}$ and $\mathcal{A}_{II-2}$ cannot ask this query with $(\mathtt{ID}_*, P_*, C_*)$.

Table 4: The CL-PKE-IND-CCA Games

Game 1: Type-I Adversary
1. $(M_{\mathfrak{pk}}, M_{\mathfrak{sk}})\leftarrow$ **CL.Setup**(1^k).
2. $(\mathtt{ID}_*, P_*, m_1, m_2, \rho)\leftarrow\mathcal{A}_{I-1}^{\mathcal{O}^1_{\mathtt{CL}}}(M_{\mathfrak{pk}})$.
3. $C_* \leftarrow$ **CL.Enc**$(M_{\mathfrak{pk}}, \mathtt{ID}_*, P_*, m_b)$ with random $b \leftarrow \{0,1\}$.
4. $b'\leftarrow\mathcal{A}_{I-2}^{\mathcal{O}^1_{\mathtt{CL}}}(M_{\mathfrak{pk}}, \mathtt{ID}_*, P_*, m_1, m_2, C_*, \rho)$.
5. succeed if $b = b'$, $(\mathtt{ID}_*, P_*) \notin \mathbb{S}_1 \cup \mathbb{Q}$, and $(\mathtt{ID}_*, P_*, C_*) \notin \mathbb{D}$.
Game 2: Type-II Adversary
1. $(M_{\mathfrak{pk}}, M_{\mathfrak{sk}})\leftarrow$ **CL.Setup**(1^k).
2. $(\mathtt{ID}_*, P_*, m_1, m_2, \rho)\leftarrow\mathcal{A}_{II-1}^{\mathcal{O}^2_{\mathtt{CL}}}(M_{\mathfrak{pk}}, M_{\mathfrak{sk}})$.
3. $C_* \leftarrow$ **CL.Enc**$(M_{\mathfrak{pk}}, \mathtt{ID}_*, P_*, m_b)$ with random $b \leftarrow \{0,1\}$.
4. $b'\leftarrow\mathcal{A}_{II-2}^{\mathcal{O}^2_{\mathtt{CL}}}(M_{\mathfrak{pk}}, M_{\mathfrak{sk}}, \mathtt{ID}_*, P_*, m_1, m_2, C_*, \rho)$.
5. succeed if $b = b'$, $P_* \in \mathbb{P}$, $(\mathtt{ID}_*, P_*) \notin \mathbb{S}_1 \cup \mathbb{S}_2$, and $(\mathtt{ID}_*, P_*, C_*) \notin \mathbb{D}$.

Like the security definition of CL-PKS, the adversary here is not allowed to issue the **CL.Replace-Public-Key**$(\mathtt{ID}_A, P_A)$ query. Instead, at the end of stage one in both games, the challenger has to encrypt the message m_b with a public key P_*

chosen by the adversary. The challenger does not need to answer query **CL.Decrypt-Message**($\mathtt{ID}_A, P_A, C$) correctly without knowing related private key as in practice. Adversaries defined by this approach corresponds to the normal (instead of conceptional strong) adversaries in [1].

Definition 3 *A CL-PKE is secure if the advantage: 2(Pr[succeed]-1/2) of both $\mathcal{A}_I$ and $\mathcal{A}_{II}$ in the CL-PKE-IND-CCA games is negligible.*

2.3 Unification of Two Realms

In [14], the authors interpreted the reason that "the composition of two 'provably secure' schemes, namely original OMC and ECDSA, results in an insecure scheme" as "This situation may be viewed as a specific limitation of the security definition for implicit certificates given in" [15], "or ... as a broader limitation of provable security, or ... as a need to formulate all security definitions according to the recently defined universal composability."

Because both OMC and ECQV appear to be natural candidates to generate implicit certificates, we interpret this failure of universal composition as the limitation of implicit certificates in general. Hence, we should not purposely define a stronger implicit-certificate security notion, which maintains universal composability but excludes those natural constructions such as OMC and ECQV. Instead, we need to define proper security notions that the full implicit-certificate-based mechanisms should satisfy. Meanwhile, the Al-Riyami-Paterson CL-PKC formulation has rigorous security definitions [1, 52] but it appears difficult to construct cryptographic schemes from standard algorithms under the model.

The new CL-PKC formulation and security definitions in this work overcome this hurdle. The formulation above unifies these two types of security mechanisms, namely the one using implicit certificates and CL-PKC following Al-Riyami-Paterson's definition, under one umbrella, and brings forth the benefits of both realms, i.e., the efficiency of implicit-certificate-based schemes and the rigorous security analysis approach of CL-PKC.

3 Certificateless Key Generation

3.1 The CL-KGA Scheme

Here following the definition in Section 2, we present a certificateless key generation algorithm to generate public/private keypairs, which will be used in the CL-PKS schemes later. The algorithm can also be used to construct CL-PKE and CL-KA schemes. The scheme is built upon the standard elliptic curve Schnorr signature (specifically EC-FSDSA [32]). In the description, we use symbol $\in_R$ to denote the operation to randomly choose from a set, and x_G and y_G to signify the x-coordinate and y-coordinate of a point G respectively.

- **CL.Setup**(1^k)
    1. Select an elliptic curve $\boldsymbol{E} : Y^3 = X^2 + aX + b$ defined over a prime field $\mathbb{F}_p$. The curve has a cyclic point group $\mathbb{G}$ of prime order q.

2. Pick a generator $G \in \mathbb{G}$.
3. $s \in_R \mathbb{Z}_q^*$.
4. $P_{KGC} = [s]G$.
5. Pick two cryptographic hash functions: $H_1 : \{0,1\}^* \to \{0,1\}^n; H_2 : \{0,1\}^* \to \mathbb{Z}_q^*$ for some integer $n > 0$.
6. Output $M_{\mathfrak{pk}} = (a, b, p, q, G, P_{KGC}, H_1, H_2)$ and $M_{\mathfrak{sk}} = s$.

- **CL.Set-User-Key**$(M_{\mathfrak{pk}}, \mathtt{ID}_A)$
  1. $x_A \in_R \mathbb{Z}_q^*$.
  2. $U_A = [x_A]G$.
  3. Output (U_A, x_A).
- **CL.Extract-Partial-Key**$(M_{\mathfrak{pk}}, M_{\mathfrak{sk}}, \mathtt{ID}_A, U_A)$
  1. $Z = H_1(a\|b\|x_G\|y_G\|x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$.
  2. $w \in_R \mathbb{Z}_q^*$.
  3. $X = [w]G$.
  4. $W = U_A + X$.
  5. $\lambda = H_2(x_W\|y_W\|Z)$.
  6. $t = (w + \lambda \cdot s) \mod q$.
  7. Output $(W_A = W, d_A = t)$.
- **CL.Set-Private-Key**$(M_{\mathfrak{pk}}, \mathtt{ID}_A, U_A, x_A, W_A, d_A)$
  1. Output $s_A = (x_A + d_A) \mod q$.
- **CL.Set-Public-Key**$(M_{\mathfrak{pk}}, \mathtt{ID}_A, U_A, W_A)$
  1. Output $P_A = W_A$.
- **CL.Calculate-Public-Key**$(M_{\mathfrak{pk}}, \mathtt{ID}_A, P_A)$
  1. $Z = H_1(a\|b\|x_G\|y_G\|x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$.
  2. $\lambda = H_2(x_{P_A}\|y_{P_A}\|Z)$.
  3. $O_A = P_A + [\lambda]P_{KGC}$.
- **CL.Verify-Key**$(M_{\mathfrak{pk}}, \mathtt{ID}_A, P_A, s_A)$
  1. $Z = H_1(a\|b\|x_G\|y_G\|x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$.
  2. $\lambda = H_2(x_{P_A}\|y_{P_A}\|Z)$.
  3. $P'_A = [s_A]G - [\lambda]P_{KGC}$.
  4. Output **valid** if $P_A = P'_A$, and **invalid** otherwise.

It is easy to check that $O_A = [s_A]G$ and everyone can compute it from public values. However, function **CL.Verify-Key** makes use of s_A, so only the owner of the keypair can validate its correctness. It cannot be done by one just knowing O_A. The equation $P'_A = O_A - [\lambda]P_{KGC}$ and $P_A = P'_A$ does not mean a Schnorr signature. The hash-function H_1 in the description is unnecessary in theory, but useful for a neat implementation. **CL.Calculate-Public-Key**($M_{\mathfrak{pk}}$, $\texttt{ID}_A$, P_A) in this scheme could be more efficient than the corresponding operation of ECQV as shown in Appendix 8.1, because $[\lambda]P_{KGC}$ could be speeded up with pre-computation for the fixed P_{KGC}.

We note that one may switch the position of P_{KGC} and W in the operation to generate Z and λ and adjust the sequence of operations accordingly in **CL.Extract-Partial-Key**.

The security of the CL-KGA can be summarised by following two theorems.

Definition 4 *Let* $(\mathbb{G}, G, q)$ *be a group of prime order* q *and* G *is a generator. The discrete logarithm problem (DLP) is given a random* $P \in \mathbb{G}$ *to find* α *such that* $P = [\alpha]G$.

Theorem 1 *If there exists a Type-I adversary* $\mathcal{A}_I$ *that has a non-negligible probability of success in Game 1 against the CL-KGA, then the DLP in group* $\mathbb{G}$ *can be solved in polynomial time in the random oracle model.*

The reduction behaves very much like the reduction of Schnorr signature in [44]. The challenger simulates the KGC (the signer) to answer **CL.Extract-Partial-Key**($M_{\mathfrak{pk}}$, $M_{\mathfrak{sk}}$, $\texttt{ID}_A$, U_A) as follows: it randomly chooses $w, \lambda \in \mathbb{Z}_q^*$, and returns $(W = [w]G + U_A - [\lambda]P_{KGC}, t = w)$ if W is not point zero, otherwise resamples w. This response should be indistinguishable from the result generated with private key s: randomly choose $w, \lambda \in \mathbb{Z}_q^*$, and return $(W = [w]G + U_A, t = (w + \lambda \cdot s) \mod q)$. To answer query **CL.Get-Public-Key**($\texttt{ID}_i$, *true*), randomly select $x_i, d_i, \lambda_i \in \mathbb{Z}_q^*$, return $P_i = [x_i]G + [d_i]G - [\lambda_i]P_{KGC}$. To answer query **CL.Get-Private-Key**($\texttt{ID}_i, P_i$), return $x_i + d_i$. To answer **CL.Get-User-Key**($\texttt{ID}_i, P_i$), return x_i. We skip the details of the full reduction.

Theorem 2 *If there exists a Type-II adversary* $\mathcal{A}_{II}$ *that has a non-negligible probability of success in Game 2 against the CL-KGA, then the DLP in group* $\mathbb{G}$ *can be solved in polynomial time in the random oracle model.*

Proof. Suppose that $\mathcal{A}_{II}$ succeeds in Game 2 with a non-negligible probability $\epsilon(k)$ in time $t(k)$. Given a DLP ($\mathbb{G}$, G, $[\alpha]G$), we use $\mathcal{A}_{II}$ to construct an algorithm $\mathcal{C}$ to compute α. Suppose that in Game 2, **CL.Get-Public-Key** is queried $\mathcal{N}_{pub}$ times with *bNewKey* as true. The challenger $\mathcal{C}$ randomly selects an index $0 < \mathcal{I} \leq \mathcal{N}_{pub}$. $\mathcal{C}$ maintains a tuple $\mathcal{T}$ in the form of $\langle \texttt{ID}_i, P_i, U_i, x_i, d_i, s_i, w_i \rangle$, which is indexed by $(\texttt{ID}_i, P_i)$. $\mathcal{T}_c$ is a counter, which increases by one each time when a new entry is put in $\mathcal{T}$. $\mathcal{C}$ answers the queries as follows:

- **CL.Setup**(1^k). $\mathcal{C}$ follows the algorithm to compute $M_{\mathfrak{pk}}$ and $M_{\mathfrak{sk}}$, and passes the values to $\mathcal{A}_{II}$.

- **CL.Get-Public-Key**($\texttt{ID}_i$, *bNewKey*). If *bNewKey* is false and at least one entry in $\mathcal{T}$ includes $\texttt{ID}_i$, then $\mathcal{C}$ returns P_i in the latest entry of $\texttt{ID}_i$ in $\mathcal{T}$, otherwise responds differently in the following two cases:

1. If $\mathcal{T}_c = \mathcal{I}$, then $\mathcal{C}$ runs **CL.Extract-Partial-Key**($M_{\mathfrak{pk}}$, $M_{\mathfrak{sk}}$, $\mathtt{ID}_i$, $[\alpha]G$) to get (W_i, d_i) and the internal random value w_i^*, and puts $\langle \mathtt{ID}_i$, W_i, $[\alpha]G$, $\perp$, d_i, $\perp$, $w_i^* \rangle$ in $\mathcal{T}$; $\mathcal{C}$ returns W_i.
2. Else, $\mathcal{C}$ randomly selects $x_i \in \mathbb{Z}_q^*$ and runs **CL.Extract-Partial-Key**($M_{\mathfrak{pk}}$, $M_{\mathfrak{sk}}$, $\mathtt{ID}_i$, $[x_i]G$) to get (W_i, d_i) and the internal random value w_i, and puts $\langle \mathtt{ID}_i$, W_i, $[x_i]G$, x_i, $d_i, x_i + d_i, w_i \rangle$ in $\mathcal{T}$; $\mathcal{C}$ returns P_i.

- **CL.Get-Private-Key(**$\mathtt{ID}_i, P_i$**)**. If there is no entry indexed by $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$, return error. Otherwise, if s_i of the found entry is $\perp$, then terminate the game (**Event 1**), or return s_i.

- **CL.Get-User-Key(**$\mathtt{ID}_i, P_i$**)**. If there is no entry indexed by $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$, return error. Otherwise, if x_i of the found entry is $\perp$, then terminate the game (**Event 2**), or return x_i.

- **CL.Verify-Key**($M_{\mathfrak{pk}}, \mathtt{ID}_*, P_*, s_*$). $\mathcal{C}$ searches $\mathcal{T}$, and if the $\mathcal{I}$-th entry does not include P_*, then terminates the game (**Event 3**). Otherwise, $\mathcal{C}$ outputs $s_* - s\lambda_* - w_i^*$ as the solution to the DLP, where λ_* is computed according to the specification by querying H_1 and H_2.

- Query to random oracle H_1 or H_2: $\mathcal{C}$ just simulates these random oracles as standard ones.

If the $\mathcal{I}$-th entry includes P_* ($\neg$**Event 3**), then the game won't terminate early ($\neg$**Event 1**$\wedge\neg$**Event 2**) and $\mathcal{A}_{II}$ won't notice any difference between the simulation and the attacking environment. $\mathcal{C}$ solves the DLP with the probability of $\frac{\epsilon(k)}{\mathcal{N}_{pub}}$ and time $O(t(k))$. ■

3.2 On Some Choices of the Scheme

One may notice that **CL.Verify-Key** after step 2 is precisely a standard Schnorr verification function which verifies the signature (P_A, s_A) on the message Z.

Message Z here is a hash result of the concatenation of octet representation of $M_{\mathfrak{pk}}$ and $\mathtt{ID}_A$. We choose this design based on several considerations. The use of H_1 is unnecessary in theory, but useful for a neat implementation. Z can be only the concatenation of octet representation of $M_{\mathfrak{pk}}$ and $\mathtt{ID}_A$. This change would not affect much the security analysis of the CL-KGA. While from the practical point of view, the interface of a signature algorithm such as [47] typically only accepts a message digest instead of a full message. This type of interface not only forces a modular approach for the signing and verification process but also reduces memory consumption in a (hardware) implementation. Without restricting the length of $\mathtt{ID}_A$, which may include other information such as the time period of the generated key, it appears reasonable to introduce an extra hash operation.

The inclusion of $M_{\mathfrak{pk}}$ in the input to H_1 appears to help only a little on the security of the CL-KGA. On the key-related attack, Morita et al. showed that one has to recompute $P_{KGC} = [s]G$ in every signing action before including P_{KGC} in H_1 to defend certain attack [40]. On the aspect of security deduction in the multi-user setting [9], there won't be many KGCs, and a user usually will only register with a handful of them. On the other hand, Z computed in current mode may serve as a fixed-size globally unique identifier of a user with a KGC. Therefore λ, which is generated in the Schnorr signing process, may act as a fixed-size globally unique identifier of the $\langle$KGC, user, public-key$\rangle$ trio. Instead of using an independent procedure to compute these values to identify keys, integrating these values into the cryptographic schemes helps avoid possible management operational mistakes.

The downside of the design is that an extra hash operation is executed in the **CL.Calculate-Public-Key** function whenever O_A is required and additional storage is used to store those input values. Fortunately, for the **CL.Sign** function, saving only λ is enough if we ignore some advanced key-related attacks and λ is also necessary for the security of the presented CL-PKS schemes as we will see in Section 4. The **CL.Encrypt** and **CL.Verify** function can compute Z on the fly without the extra cost of persistent storage.

Overall the benefit brought by the current way of generating Z weighs against the little extra cost of a hash operation and minor implementation hassle. On the unique representation of the domain parameter $M_{\mathfrak{pk}}$, instead of only using P_{KGC}, a conservative approach of including those essential values is chosen to prevent possible loopholes including advanced attacks exploiting different curve parameters such as the invalid-curve attack [4] or the domain parameter shifting attack [50]. If all KGCs use a fixed curve and G, the value of a, b and G may be excluded from H_1.

3.3 Secure Key Provision

In the CL-KGA process, a user queries the KGC for his partial keys with his public key value U_A. Once (W_A, d_A) is generated, there should be a security protection mechanism to safely distribute these values to the user. One solution is to establish a secure channel between users and the KGC, which requires extra trust chain or pre-deployed secrets. Due to the high sensitivity of d_A and in pursuit of a more succinct key management system using CL-PKC, it would be desirable to have a better solution. Observing that U_A is provided by the user who should know the corresponding private value, the KGC can encrypt d_A with U_A through a standard public key encryption algorithm such as ECIES [33]. This approach also implicitly verifies that the user knows d_A, which is although not as critical as a process required for the same security purpose when a CA issues certificates.

4 CL-PKS

4.1 Generic Approach to Construct CL-PKS

Using CL-KGA, a user with identity $\texttt{ID}_A$ generates a pair of keys (P_A, s_A), and everyone can call function **CL.Calculate-Public-Key**($M_{\mathfrak{pk}}$, $\texttt{ID}_A$, P_A) to compute the real public key O_A. A standard signature scheme is defined by three functions $(\mathcal{G}, \Sigma, V)$ such that the key generation function $\mathcal{G}$ generates a keypair (O_A, s_A), the signing function Σ takes (O_A, s_A, m) as input and produces a signature σ, and the verification function V takes (O_A, m, σ) as input and tests whether σ is a valid signature of m with respect to O_A. An obvious way to construct a CL-PKS is to call a CL-KGA to generate keys and call Σ in **CL.Sign** and call **CL.Calculate-Public-Key** first to compute O_A and then call V to test a signature in **CL.Verify**. However, such crude construction with a CL-KGA that is secure by Definition 1 and a signature scheme that is EUF-CMA-secure even in the multi-user setting [39] does not always end up with a secure CL-PKS satisfying Definition 2.

Menezes and Smart investigated the security notions of digital signature in the multi-user setting [39]. They formulated two types of security notions for a signature scheme in this case. One security notion is formulated against weak-key substitution (WSK) attacks, which requires that an adversary, if outputs a pair of message and signature generated upon public key O_i that is also valid with respect to a different public key O_*, should know the private key corresponding to O_*. With this restriction, they proved that ECDSA is WSK-secure if users share the same domain parameters such as those in $M_{\mathfrak{pk}}$. In Section 3 we have proved that the CL-KGA, which bears high similarity with the OMC scheme, is secure by Definition 1. However, the simple combination of the CL-KGA with ECDSA following the suggested method does not produce a secure CL-PKS. In [14] Brown et al. detailed a security analysis which shows that the OMC with ECDSA is completely broken and the ECQV with ECDSA is not safe against an artificial forgery attack. These cases demonstrate that an EUF-CMA and WSK-secure DSA is not sufficient for universal composability. This happens because in the CL-PKS setting, an adversary may output a valid tuple $(\mathtt{ID}_*, P_*, m_*, \sigma_*)$ without knowing the private key. Moreover, m_* may not have been signed by any entity in the system and P_* may not belong to any entity either. Hence, it appears necessary that the used EUF-CMA-secure DSA is against the strong-key substitution (SKS) attacks [39], which does not require the adversary knows the private key corresponding to O_* after outputting $(\mathtt{ID}_*, P_*, m_*, \sigma_*)$ for test, where $O_* \leftarrow$**CL.Calculate-Public-Key**($M_{\mathfrak{pk}}$, $\mathtt{ID}_*$, P_*) and **valid**$\leftarrow$**CL.Verify**$(M_{\mathfrak{pk}}, \mathtt{ID}_*, P_*, m_*, \sigma_*)$.

Here, we show a simple technique to enhance the security of composed schemes. The intermediate value λ in the CL-KGA, which is generated in the Schnorr signing process, is called the *assignment* in the general framework defined in ISO/IEC 14888-3 [32] for signatures schemes based on discrete logarithm with randomized witness. If the signing function of the digital signature algorithm (DSA) is signing on $(\lambda \| m)$ instead of m, the two algorithms, the CL-KGA and DSA, are linked together to safeguard the security of resulting CL-PKS. Intuitively, with including λ as the prefix of the message to be signed, the signer is forced to commit to a public key P_A and hence the corresponding *real* public key O_A before generating a signature. This mechanism takes away the freedom of a forger to generate a signature before finding a public key P_A satisfying the verification equation. The security of a standard DSA such as ECDSA guarantees that without knowing the private key, it is unlikely to generate a valid signature with respect to a given public key O_A. Meanwhile, the security of the CL-KGA assures that without the help of the KGC, the adversary cannot compute the private key s_A corresponding to a given public key O_A.

This simple technique works like applying with the so-called "key prefixing" technique [9, 39] by signing on a message together with the signer's public key and its identity indirectly. The technique has been used in [24] to construct an identity-based signature (IBS) scheme ECCSI. We apply this technique to construct four CL-PKS schemes. We will show later that the technique indeed plays an essential role to defeat all the known attacks against the resulting CL-PKS. We note that using $Z \| x_{P_A} \| y_{P_A}$ instead of λ as the key prefixing achieves the same effect.

4.2 CL-PKS1 from ECDSA

First, we present a scheme (CL-PKS1) using the CL-KGA and the standard algorithm ECDSA. The scheme uses another hash function $H_3 : \{0,1\}^* \rightarrow \{0,1\}^n$. In practice, both H_1 and H_3 are instantiated by a secure hash function like SHA256. H_2 is also constructed from the same hash function by excluding the zero output modulo q.

Table 5: CL-PKS1

CL.Sign($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, s_A, m)	**CL.Verify**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, m, σ)
1. $Z = H_1(a\|b\|x_G\|y_G\| x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$.	1. $Z = H_1(a\|b\|x_G\|y_G\| x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$.
2. $\lambda = H_2(x_{P_A}\|y_{P_A}\|Z)$.	2. $\lambda = H_2(x_{P_A}\|y_{P_A}\|Z)$.
3. $h = H_3(\lambda\|m)$.	3. $O_A = P_A + [\lambda]P_{KGC}$.
4. $r \in_R \mathbb{Z}_q^*$.	4. $h = H_3(\lambda\|m)$.
5. $Q = [r]G$.	5. $v_1 = v^{-1} \cdot h \mod q$.
6. $u = x_Q \mod q$.	6. $v_2 = v^{-1} \cdot u \mod q$.
7. $v = r^{-1} \cdot (u \cdot s_A + h) \mod q$.	7. $Q' = [v_1]G + [v_2]O_A$.
8. Output $\sigma = (u, v)$.	8. $u' = x_{Q'} \mod q$.
	9. Output **valid** if $u = u'$, and **invalid** otherwise.

The presented **CL.Sign** function from step 3 exactly follows ECDSA to sign with private key s_A on message $(\lambda\|m)$. The first two steps can be treated as a message preparation process, which re-generates the *assignment* computed in the Schnorr signing process invoked by **CL.Extract-Partial-Key**. These two steps can further be saved if λ is pre-computed and stored. **CL.Verify** function invokes two functions sequentially. It first activates **CL.Calculate-Public-Key** to calculate the signer's supposed real public key O_A and then calls the verification function of ECDSA to verify signature σ on message $(\lambda\|m)$ with regard to O_A. We note that signing on $(\lambda\|m)$ instead of m does not require any modification to the implementation of ECDSA either in software or hardware.

4.3 Effect of Using the Assignment as the Key Prefixing

4.3.1 Revisiting Existing Attacks

In [14], it's been shown that both the OMC and ECQV are insecure with ECDSA with direct composition. The analysis below shows that after applying with the key prefixing technique of signing on $(\lambda\|m)$, both CL-PKS1 and ECQV with ECDSA are secure against the known attacks and CL-PKS1 has better security than the ECQV with the vanilla ECDSA scheme.

Using the notation of this paper, we revisit the attack analysis of [14] applying to CL-PKS1 and ECQV with ECDSA. To guarantee that **CL.Verify**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, m, σ) outputs **valid** in CL-PKS1, equation (1) should be satisfied.

$$[v]Q - [u]P_A = [h]G + [\lambda][u]P_{KGC}. \quad (1)$$

We express equation (1) using row and column vectors:

$$\begin{bmatrix}-u & v\end{bmatrix}\begin{bmatrix}P_A\\ Q\end{bmatrix} = \begin{bmatrix}\lambda u & h\end{bmatrix}\begin{bmatrix}P_{KGC}\\ G\end{bmatrix}. \tag{2}$$

The relation of P_A and Q to P_{KGC} and G can be expressed as

$$\begin{bmatrix}P_A\\ Q\end{bmatrix} = \begin{bmatrix}d & e\\ f & g\end{bmatrix}\begin{bmatrix}P_{KGC}\\ G\end{bmatrix}. \tag{3}$$

After replacing $[P_A, Q]$ and eliminating $[P_{KGC}, G]$ and transposing the resulting matrix, we have

$$\begin{bmatrix}d & f\\ e & g\end{bmatrix}\begin{bmatrix}-u\\ v\end{bmatrix} = \begin{bmatrix}\lambda u\\ h\end{bmatrix}. \tag{4}$$

By multiplying the inverse matrix (we first assume that the matrix is invertible), we get

$$\begin{bmatrix}-u\\ v\end{bmatrix} = \frac{1}{dg-ef}\begin{bmatrix}gu\lambda - fh\\ -eu\lambda + dh\end{bmatrix}. \tag{5}$$

As in [14], we consider λ as a non-linear function of d and e: $\lambda = L(d,e)$, since $\lambda = H_2(x_{P_A}\|y_{P_A}\|Z)$ and $P_A = [d]P_{KGC} + [e]G$. Similarly, we have u as a non-linear function of f and g: $u = U(f,g)$, since $u = \tilde{x}_Q = x_Q \mod q$ and $Q = [f]P_{KGC} + [g]G$. Different from [14], we have an extra non-linear function $h = H(L(d,e))$, since $h = H_3(\lambda\|m)$. This produces five equations:

$$-u = \frac{gu\lambda - fh}{dg-ef}, \tag{6}$$

$$v = \frac{-eu\lambda + dh}{dg-ef}, \tag{7}$$

$$\lambda = L(d,e), \tag{8}$$

$$u = U(f,g), \tag{9}$$

$$h = H(L(d,e)). \tag{10}$$

Substituting λ, u and h in equation (6) and (7) with equation (8), (9) and (10) respectively, we get

$$-U(f,g) = \frac{g}{dg-ef}U(f,g)L(d,e) - \frac{f}{dg-ef}H(L(d,e)), \tag{11}$$

$$v = -\frac{e}{dg-ef}U(f,g)L(d,e) + \frac{d}{dg-ef}H(L(d,e)). \tag{12}$$

If we choose $g = 0$ as the attacks in [14], the adversary needs to resolve the following equations:

$$-U(f,0) = \frac{1}{e}H(L(d,e)), \tag{13}$$

$$v = -\frac{H(L(d,e))(L(d,e)+d)}{ef}. \tag{14}$$

We slight abuse the notation by using a point instead of coordinates of a point in hash functions. The above equations can be converted to

$$P_A = [d]P_{KGC} + [e]G, \tag{15}$$

$$\tilde{x}_{[f]P_{KGC}} = -\frac{1}{e}H_3(H_2(P_A\|Z)\|m), \tag{16}$$

$$v = \frac{\tilde{x}_{[f]P_{KGC}} \cdot (H_2(P_A\|Z) + d)}{f}. \tag{17}$$

By using the attacks in [14] on OMC with ECDSA to forge a valid signature, the attacker simply chooses an identity $\mathtt{ID}_A$, a message m and any $0 < d, f < q$ and computes $u = \tilde{x}_{[f]P_{KGC}}$ and $e = -\frac{H_3(m)}{\tilde{x}_{[f]P_{KGC}}}$ first, and further computes v according to equation (7). However, in CL-PKS1, e appears on both sides of equation (18) and the relation is non-linear because of involving hash functions.

$$e = -\frac{H_3(H_2(([d]P_{KGC} + [e]G)\|Z)\|m)}{\tilde{x}_{[f]P_{KGC}}}. \tag{18}$$

Recall that $Z = H_1(a\|b\|x_G\|y_G\|x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$. Hence, given any $0 < f < q$, if the hash functions are collision resistant, it would be difficult to find d and e or some $\mathtt{ID}_A$ or m satisfying equation (18). Conversely, given any *proper* d and e or some $\mathtt{ID}_A$ or m, the hash functions simulated as random oracles would generate a random $j = -\frac{1}{e}H_3(H_2(P_A\|Z)\|m)$ corresponding to the x-coordinate modulo q of a point in a set J whose cardinality is small for practically used curves [38]. The problem becomes given $(G, [s]G, J)$ finding f such that $[f][s]G \in J$ for a random s and a random small set J. This problem appears hard based on the DL assumption.

It's not difficult to verify that the above analysis also works on the combination of ECQV with ECDSA. In ECQV plus ECDSA, if $(\lambda\|m)$ is signed, equation (11) becomes

$$-U(f,g)L(d,e) = \frac{g}{dg - ef}U(f,g) - \frac{f}{dg - ef}H(L(d,e)). \tag{19}$$

When $g = 0$, the equation of e becomes

$$e = -\frac{H_3(H_2(([d]P_{KGC} + [e]G)\|Z)\|m)}{H_2(([d]P_{KGC} + [e]G)\|Z) \cdot \tilde{x}_{[f]P_{KGC}}}. \tag{20}$$

The Kravitz attack [14] requires finding (Z, P_A, m) satisfying $H_3(H_2(P_A\|Z)\|m) = H_2(P_A\|Z)$. This task becomes difficult if the used hash functions are collision resistant.

Now, let's consider that the matrix in equation (4) is non-invertible ($dg = ef$), and in this case the attacker against CL-PKS1 has to find (Z, P_A, m, $\tilde{x}_Q$, v) satisfying the following simultaneous equations:

$$P_A = [d][s]G + [e]G,$$
$$Q = [f][s]G + [g]G,$$
$$dg = ef,$$
$$v = \frac{H_2(P_A\|Z)s\tilde{x}_Q + ds\tilde{x}_Q + e\tilde{x}_Q + H_3(H_2(P_A\|Z)\|m)}{fs + g}. \tag{21}$$

Note that a valid signature requires $H_3(H_2(P_A\|Z)\|m) \neq 0$, $u = \tilde{x}_Q \neq 0$ and $v \neq 0$. Let's investigate the possible four cases depending on the value of f and e.

1. Case 1: $f = 0, d = 0$, then $g \neq 0$ and $e \neq 0$, and

$$v = \frac{H_2(P_A \| Z)s\tilde{x}_Q + e\tilde{x}_Q + H_3(H_2(P_A \| Z) \| m)}{g}.$$

Now, $P_A = [e]G$. Hence, if P_A is fixed, so is e, and there is only a negligible probability that $H_2(P_A \| Z) = -e/s$.

2. Case 2: $f = 0, e = 0$, then $g \neq 0$ and $d \neq 0$, and

$$v = \frac{H_2(P_A \| Z)s\tilde{x}_Q + ds\tilde{x}_Q + H_3(H_2(P_A \| Z) \| m)}{g}.$$

Now, $P_A = [d][s]G$. Hence, if P_A is fixed, so is d, and there is only a negligible probability that $H_2(P_A \| Z) = -d$.

3. Case 3: $e = 0, g = 0$, then $f \neq 0, d \neq 0$, and

$$v = \frac{H_2(P_A \| Z)\tilde{x}_Q + d\tilde{x}_Q}{f} + \frac{H_3(H_2(P_A \| Z) \| m)}{fs}.$$

A valid signature requires $H_3(H_2(P_A \| Z) \| m) \neq 0$.

4. Case 4: $ef = dg \neq 0$. H_2 as a random oracle forces the attacker to fix P_A before computing v. Let $c = ds + e \neq 0$ as some constant. From $dg = ef$, we have $fs + g = gc/e$, so the attacker after querying H_2 and H_3 computes

$$v = e\frac{H_2(P_A \| Z)s\tilde{x}_Q + c\tilde{x}_Q + H_3(H_2(P_A \| Z) \| m)}{gc}.$$

Again, there is only a negligible probability that $H_2(P_A \| Z) = -c/s$.

In all four cases, there appears to be no simple trick to compute v without knowing s.

Overall, we can see that the key prefixing method by signing on $(\lambda \| m)$ indeed plays an essential role to help CL-PKS1 defeat existing attacks against OMC or ECQV.

4.3.2 Further Security Result

We see that CL-PKS1 can defend known attacks against a direct composition of CL-KGA and ECDSA. In fact, with including λ in H_3, we can further establish following result.

Lemma 1 *In the random oracle model, if there exists an efficient algorithm to solve equation (11), then there exists an efficient algorithm to solve equation (19).*

Proof. Suppose that an algorithm $\mathcal{A}$ finds a solution to equation (11) with probability $\epsilon(k)$ in running time $t(k)$. Suppose $\mathcal{A}$ makes $\mathcal{N}_{H_3}$ queries to H_3. Let ζ be a random integer such that $0 < \zeta \leq \mathcal{N}_{H_3}$. We construct an algorithm $\mathcal{B}$ by re-running $\mathcal{A}$. However, this time for the ζ-th query to $H_3(\lambda_\zeta \| m_\zeta)$, the oracle returns $h_\zeta \lambda_\zeta$, where h_ζ is the output of the same query in the last run, and all other random oracle queries return same values as last time. With $1/\mathcal{N}_{H_3}$ probability, $\mathcal{B}$ will find a solution to the following equation

$$-U(f,g)L^{-1}(d,e) = \frac{g}{dg-ef}U(f,g) - \frac{f}{dg-ef}H(L(d,e)). \tag{22}$$

Suppose $\mathcal{B}$ makes $\mathcal{N}_{H_2}$ queries to H_2. Let γ be a random integer such that $0 < \gamma \leq \mathcal{N}_{H_2}$. We construct an algorithm $\mathcal{D}$ by re-running $\mathcal{B}$. This time the oracle returns $1/\lambda_\gamma$ for the γ-th query to $H_2(P_\gamma \| Z_\gamma)$ and returns $H_3(\frac{1}{\lambda_\gamma}\|m_j) = h_j$, where in the last run $\lambda_\gamma = H_2(P_\gamma\|Z_\gamma)$ and $h_j = H_3(\lambda_\gamma\|m_j)$ for each j. Overall, if such algorithm $\mathcal{A}$ exists, then there exists an algorithm to solve equation (19) with probability $O(\frac{\epsilon(k)}{\mathcal{N}_{H_2}\cdot\mathcal{N}_{H_3}})$ and time $O(t(k))$. ■

In [14], Brown et al. proved in Theorem 1 that in the *combined* random oracle (for the hash function) and generic group model (for the elliptic curve group) [45], there does not exist an efficiently algorithm, which can find a solution (other than Kravitz's) to equation (23)

$$-U(f,g)L(d,e) = \frac{g}{dg-ef}U(f,g) - \frac{f}{dg-ef}H', \tag{23}$$

where H' is a hash function that only depends on m. Obviously any solution to equation (19) can be converted to a solution to equation (23) by using $\lambda\|m$ as the message input to H'. Following from Lemma 1, we conclude that there is no efficient algorithm to solve equation (11) in the same model. This result implies that CL-PKS1 has the security at least equivalent to (in fact better than) the vanilla ECQV with ECDSA scheme against attackers who forge a signature by solving equation (11) and (23) respectively. Note that in the generic group model, the DLP is hard [45].

4.4 More CL-PKS from ECDSA-II and Schnorr-DSA

Because ECDSA lacks a security reduction based on a standard complexity assumption, several modifications to ECDSA such as [38] were proposed to address this issue. All modifications include u as an input to H_3. However, the way to generate u is different in each proposal. We use a variant of ECDSA by setting $u = x_Q$ (called ECDSA-II in [38]). For most of the elliptic curves defined over prime fields used in practice, this modification will not change the size of the representation of u. On the other hand, this variant can be proved secure in the random oracle with the *Improved Forking Lemma* [11] as in [38]. We use this modified ECDSA to construct CL-PKS2.

We note that without including λ, even with u as an input to H_3, such variant still suffers from the attacks shown in Section 4.3. This again demonstrates the effectiveness of the key prefixing technique. Another scheme with a standard reduction is Schnorr DSA: EC-FSDSA [32]. A certificateless variant of EC-FSDSA is shown in Table 7.

5 Security Analysis

Now, we analyze the security of the schemes. Apart from the analysis against the existing attacks in Section 4.3, we present two formal security results of CL-PKS1 for

Table 6: CL-PKS2

CL.Sign($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, s_A, m)	**CL.Verify**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, m, σ)
1. $Z = H_1(a\|b\|x_G\|y_G\| x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$. 2. $\lambda = H_2(x_{P_A}\|y_{P_A}\|Z)$. 3. $r \in_R \mathbb{Z}_q^*$. 4. $Q = [r]G$. 5. $u = x_Q$. 6. $h = H_3(u\|\lambda\|m)$. 7. $v = r^{-1} \cdot (u \cdot s_A + h) \mod q$. 8. Output $\sigma = (u, v)$.	1. $Z = H_1(a\|b\|x_G\|y_G\| x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$. 2. $\lambda = H_2(x_{P_A}\|y_{P_A}\|Z)$. 3. $O_A = P_A + [\lambda]P_{KGC}$. 4. $h = H_3(u\|\lambda\|m)$. 5. $v_1 = v^{-1} \cdot h \mod q$. 6. $v_2 = v^{-1} \cdot u \mod q$. 7. $Q' = [v_1]G + [v_2]O_A$. 8. $u' = x_{Q'}$. 9. Output **valid** if $u = u'$, and **invalid** otherwise.

Table 7: CL-PKS3

CL.Sign($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, s_A, m)	**CL.Verify**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, m, σ)
1. $Z = H_1(a\|b\|x_G\|y_G\| x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$. 2. $\lambda = H_2(x_{P_A}\|y_{P_A}\|Z)$. 3. $r \in_R \mathbb{Z}_q^*$. 4. $Q = [r]G$. 5. $h = H_3(x_Q\|y_Q\|\lambda\|m)$. 6. $v = (r + h \cdot s_A) \mod q$. 7. Output $\sigma = (Q, v)$.	1. $Z = H_1(a\|b\|x_G\|y_G\| x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$. 2. $\lambda = H_2(x_{P_A}\|y_{P_A}\|Z)$. 3. $O_A = P_A + [\lambda]P_{KGC}$. 4. $h = H_3(x_Q\|y_Q\|\lambda\|m)$. 5. $Q' = [v]G - [h]O_A$. 6. Output **valid** if $Q = Q'$, and **invalid** otherwise.

building confidence in the scheme. The analysis of CL-PKS1 with a few changes is also applicable to ECQV with ECDSA if the technique of signing on $(\lambda\|m)$ is used. We fully analyze CL-PKS2's security. The analysis of CL-PKS3 can be done similarly to CL-PKS2.

Because the CL-PKS1 scheme is the composition of the CL-KGA and ECDSA, the security of the scheme won't be better than either of the components. For ECDSA, the known security result is either based on the collision resistance of the used hash function in the generic group model [12] or based on so-called the semi-logarithm problem (SLP) in the random oracle model [13, 20]. As we have already adopted the random oracle model to analyze the security of the CL-KGA, here we continue to analyze the security of the CL-PKS schemes in the same model.

To address the technique shortcoming of the proof, we put a restriction on the **CL.Get-Sign(**$\mathtt{ID}_A$**,** P_A**,** m**)** query. If $\mathtt{ID}_* = \mathtt{ID}_A$ and $P_* = P_A$, then each message m can be queried *at most once*. This "one-per-message unforgeability" security notion [20] is weaker than the EUF-CMA. However, it is so far the provable one for ECDSA in the random oracle. We label these two types of adversaries as Type-I$^-$ and Type-II$^-$ adversary. We note that for CL-PKS2 (CL-PKS3), this restriction is unnecessary because of including u (Q) in H_3.

Definition 5 *Let* $(\mathbb{G}, G, q)$ *be a group of prime order* q *and* G *is a generator. The semi-logarithm problem is given a random* $P \in \mathbb{G}$ *to find* (u, v) *such that* $u = \mathcal{F}([v^{-1}](G + [u]P)$*, where* $\mathcal{F}(X)$ *returns x-coordinate of point* X*.*

For Type-I adversaries, there are two possible attacking cases. Case 1: $\mathcal{A}_{Ia}$ generates a signature which is valid with a targeted $\texttt{ID}_*$ and $\texttt{ID}_*$'s public key. Case 2: $\mathcal{A}_{Ib}$ generates a signature which is valid with a targeted $\texttt{ID}_*$ but a public key different from $\texttt{ID}_*$'s. Note that in this case, the owner of $\texttt{ID}_*$ may have no public key. The security analysis results of these two CL-PKS schemes are as follows.

Lemma 2 *If there exists an adversary* $\mathcal{A}_{Ia}^{-}$ *that has a non-negligible probability of success in Game 1 against CL-PKS1 in the random oracle model, then the SLP in group* $\mathbb{G}$ *can be solved in polynomial time.*

Proof. Suppose that $\mathcal{A}_{Ia}^{-}$ succeeds in Game 1 with a non-negligible probability $\epsilon(k)$ in time $t(k)$. Given a SLP ($\mathbb{G}$, G, $[\alpha]G$), we use $\mathcal{A}_{Ia}^{-}$ to construct an algorithm $\mathcal{C}$ to find a solution. Suppose that in Game 1, H_1 and H_2 are queried $\mathcal{N}_{H_1}$ and $\mathcal{N}_{H_2}$ times respectively and $\mathcal{N}_{Key}$ keys are generated in the game through **CL.Get-Public-Key**, and the targeted $\texttt{ID}_*$ has generated $\mathcal{N}_{TKey}$ keys and $\mathcal{N}_{TH_3}$ queries on H_3 with the targeted $\texttt{ID}_*$ and P_* are called and $\mathcal{N}_E$ **CL.Extract-Partial-Key** queries are asked. The challenger $\mathcal{C}$ randomly selects three indices $0 < \mathcal{I} \leq \mathcal{N}_{H_1}, 0 < \mathcal{J} \leq \mathcal{N}_{TKey}, 0 < \mathcal{K} \leq \mathcal{N}_{TH_3}$. $\mathcal{C}$ maintains a tuple $\mathcal{T}$ in the form of $\langle \texttt{ID}_i, P_i, \lambda_i, U_i, x_i, d_i, s_i \rangle$, which is indexed by $(\texttt{ID}_i, P_i)$. For the presentation purpose, we use PI to denote the system parameter string $a\|b\|x_G\|y_G\|x_{P_{KGC}}\|y_{P_{KGC}}$. $\mathcal{C}$ answers the queries as follows:

- **CL.Setup**(1^k). $\mathcal{C}$ sets $P_{KGC} = [\alpha]G$, and passes $M_{\mathfrak{pk}}$ to $\mathcal{A}_{Ia}^{-}$. $\mathcal{C}$ randomly chooses three values $Z_*, h_* \in \{0,1\}^n$ and $\lambda_* \in \mathbb{Z}_q^*$.
- $H_1(PI\|\texttt{ID}_A)$. $\mathcal{C}$ maintains a list $\mathcal{H}_1$ in the form of $\langle I_i, Z_i \rangle$. If the input is on the list, then the hash value is returned. If this is the $\mathcal{I}$-th distinctive query, then $\mathcal{C}$ puts $(PI\|\texttt{ID}_A, Z_*)$ on the list, and returns Z_*. Otherwise, it randomly samples $Z_i \in \{0,1\}^n$ (if $Z_i = Z_*$, terminate the game (**Event 1**)), and returns Z_i after putting the pair into $\mathcal{H}_1$.
- $H_2(P_A\|Z)$. Similarly, $\mathcal{C}$ has a list $\mathcal{H}_2$ in the form of $\langle I_i, \lambda_i \rangle$. If the input is on the list, then the hash value is returned. Otherwise, it randomly samples $\lambda_i \in \mathbb{Z}_q^*$. If $\lambda_i \neq \lambda_*$, return λ_i after putting the pair into $\mathcal{H}_2$, else terminate the game (**Event 1**).
- $H_3(\lambda\|m)$. $\mathcal{C}$ maintains a list $\mathcal{H}_3$ in the form of $\langle I_i, h_i, u_i, v_i \rangle$. If the input is on the list, then the hash value h_i is returned. Otherwise, $\mathcal{C}$ behaves differently in the following cases:
  1. If $\lambda \neq \lambda_*$, then randomly choose $h_i \in \{0,1\}^n$, return h_i after putting $(\lambda\|m, h_i, \perp, \perp)$ into $\mathcal{H}_3$.
  2. Else, if this is the $\mathcal{K}$-th query, then after putting ($\lambda\|m$, h_*, $\perp$, $\perp$) in the list, return h_*. Otherwise, randomly sample $(a_i, b_i) \in \mathbb{Z}_q^{*2}$ and compute $u_i = \mathcal{F}([a_i]G + [b_i][h_*][\alpha]G)$, $v_i = u_i/b_i$, and $h_i = a_i u_i/b_i$. $\mathcal{C}$ returns h_i after putting $(\lambda\|m, h_i, u_i, v_i)$ into $\mathcal{H}_3$.

- **CL.Extract-Partial-Key**($M_{\mathtt{pk}}$, $M_{\mathtt{sk}}$, $\mathtt{ID}_i$, U_i). $\mathcal{C}$ randomly selects $d_i, \lambda_i \in \mathbb{Z}_q^*$, and computes $P_i = [d_i]G + U_i - [\lambda_i][\alpha]G$. $\mathcal{C}$ puts $(P_i \| Z_i, \lambda_i)$ in $\mathcal{H}_2$ with $Z_i = H_1(PI \| \mathtt{ID}_i)$ and returns (P_i, d_i) after putting $(\mathtt{ID}_i, P_i)$ in a set $\mathbb{Q}$. If $\mathcal{H}_2$ has an entry indexed by $(P_i \| Z_i)$ that has different value from λ_i, terminate the game (**Event 1**).

- **CL.Get-Public-Key**($\mathtt{ID}_i$,*bNewKey*). If *bNewKey* is false and at least one entry in $\mathcal{T}$ includes $\mathtt{ID}_i$, then $\mathcal{C}$ returns P_i in the latest entry of $\mathtt{ID}_i$ in $\mathcal{T}$. Otherwise, let $Z_i = H_1(PI \| \mathtt{ID}_i)$, and $\mathcal{C}$ responds differently in the following cases:

  1. If $Z_i = Z_*$, and this is the $\mathcal{J}$-th public key generation on $\mathtt{ID}_i$, then compute $P_i = [h_* - \lambda_*][\alpha]G$, put $(P_i \| Z_i, \lambda_*)$ in $\mathcal{H}_2$ and randomly select $x_i \in \mathbb{Z}_q^*$ and put ($\mathtt{ID}_i$, P_i, λ_*, $[x_i]G$, x_i, $\perp$, $\perp$) in $\mathcal{T}$; $\mathcal{C}$ returns P_i. If $\mathcal{H}_2$ has an entry indexed by $(P_i \| Z_i)$ that has different value from λ_*, terminate the game (**Event 1**).
  2. Else, randomly select $x_i, d_i, \lambda_i \in \mathbb{Z}_q^*$ (if $\lambda_i = \lambda_*$, terminate the game (**Event 1**)), compute $P_i = [x_i]G + [d_i]G - [\lambda_i][\alpha]G$, put $(P_i \| Z_i, \lambda_i)$ in $\mathcal{H}_2$ and ($\mathtt{ID}_i$, P_i, λ_i, $[x_i]G$, x_i, d_i, $x_i + d_i$) in $\mathcal{T}$; $\mathcal{C}$ returns P_i.

- **CL.Get-Private-Key(**$\mathtt{ID}_i, P_i$**)**. If there is no entry indexed by $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$, return error. Otherwise, if s_i of the found entry is $\perp$, then terminate the game (**Event 2**), or return s_i.

- **CL.Get-User-Key(**$\mathtt{ID}_i, P_i$**)**. If there is no entry indexed by $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$, return error. Otherwise, return x_i from the found entry.

- **CL.Get-Sign**($\mathtt{ID}_i, P_i, m_i$). If there is no entry indexed by $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$, return error. Otherwise, use λ_i from the found entry to query $H_3(\lambda_i \| m_i)$ and respond as follows:

  1. If s_i from the found entry is not $\perp$, then use s_i as the private key and P_i as the public key to sign the message and return signature.
  2. Else (i.e., $\lambda_i = \lambda_*$), use $\lambda_i \| m_i$ to search the list $\mathcal{H}_3$.
     - If u_i is $\perp$ on the found entry, then terminate the game (**Event 3**).
     - Else, return (u_i, v_i) as the signature.

- **CL.Verify**($M_{\mathtt{pk}}, \mathtt{ID}_*, P_*, m_*, \sigma_*$). If $Z_* \neq H_1(PI \| \mathtt{ID}_*)$ or $\lambda_* \neq H_2(P_* \| Z_*)$ or $h_* \neq H_3(\lambda_* \| m_*)$ or $(\mathtt{ID}_*, P_*) \in \mathbb{Q}$, then terminate the game (**Event 4**). Otherwise, parse σ_* as (u_*, v_*) and output $(u_*, v_*/h_*)$.

First, we claim that if the game is not terminated prematurely, then the simulation is indistinguishable from the environment and the final output is the solution of the SLP. The output of H_1, H_2 and H_3 are all sampled randomly. **CL.Extract-Partial-Key** returns the correct response as $Z_i = H_1(PI \| \mathtt{ID}_i), \lambda_i = H_2(P_i \| Z_i)$, $O_i = P_i + [\lambda_i]P_{KGC} = [d_i]G + U_i$. The keypair $(\mathtt{ID}_i, P_i, s_i)$ for an identity $\mathtt{ID}_i$ is also generated randomly. For any $(\mathtt{ID}_i, P_i, s_i)$ with $\mathtt{ID}_i \neq \mathtt{ID}_*$ or $P_i \neq P_*$, we have $Z_i = H_1(PI \| \mathtt{ID}_i)$, $\lambda_i = H_2(P_i \| Z_i)$, $O_i = P_i + [\lambda_i]P_{KGC} = [x_i + d_i]G$ and $s_i = x_i + d_i$ with x_i, d_i, s_i from the entry indexed with $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$. Hence, the keypair is valid and the signature generated by **CL.Get-Sign**($\mathtt{ID}_i, P_i, m_i$) is also valid. On the case that $\mathtt{ID}_i = \mathtt{ID}_*$ and $P_i = P_*$, (u_i, v_i) is returned as the signature. According to the reduction, $Z_* = H_1(PI\|\mathtt{ID}_*)$, $\lambda_* = H_2(P_*\|Z_*)$ and $P_* = [h_* - \lambda_*][\alpha]G$. Hence, $O_* = P_* + [\lambda_*][\alpha]G = [\alpha h_*]G$. According to **CL.Verify**, we have $v_i^1 = v_i^{-1}h_i = a_i$, $v_i^2 = v_i^{-1}u_i = b_i$, $Q_i' = [v_i^1]G + [v_i^2]O_* = [a_i]G + [b_i\alpha h_*]G$. Hence, $x_{Q_i'} = u_i$, which means the signature is valid. Furthermore, if (u_*, v_*) is a valid signature, $u_* = \mathcal{F}([v_*^{-1}h_*]G + [v_*^{-1}u_*][h_*\alpha]G) = \mathcal{F}([(v_*/h_*)^{-1}](G + [u_*][\alpha]G))$ and the SLP is solved successfully. **CL.Get-Private-Key** and **CL.Get-User-Key** return valid values which satisfy the requirements of the corresponding function definitions.

Second, we analyze the possibility of finding a solution. Let **Event 1** be that the hash collision happens on either H_1 or H_2. Let **Event 5** be that the adversary $\mathcal{A}_{Ia}^-$ indeed chooses the $\mathcal{I}$-th identity as the target, the $\mathcal{J}$-th public key of the target and the $\mathcal{K}$-th query of $H_2(\lambda_*\|m_*)$ to generate σ_*. If **Event 5** happens, then **Event 2, 3** and **4** won't happen. Overall, $\mathcal{C}$ solves the SLP with the probability at least $\frac{\epsilon(k)}{\mathcal{N}_{H_1}\cdot\mathcal{N}_{TKey}\cdot\mathcal{N}_{TH_3}} - \frac{\mathcal{N}_{H_2}+\mathcal{N}_{Key}+\mathcal{N}_E}{q} - \frac{\mathcal{N}_{H_1}}{2^n}$ and time $O(t(k))$. ■

Theorem 3 *If there exists an adversary $\mathcal{A}_{II}^-$ that has a non-negligible probability of success in Game 2 against CL-PKS1 in the random oracle model, then the SLP in group $\mathbb{G}$ can be solved in polynomial time.*

Proof. Suppose that $\mathcal{A}_{II}^-$ succeeds in Game 2 with a non-negligible probability $\epsilon(k)$ in time $t(k)$. Given a SLP $(\mathbb{G}, G, [\alpha]G)$, we use $\mathcal{A}_{II}^-$ to construct an algorithm $\mathcal{C}$ to find a solution. Suppose that in Game 2, H_1 and H_2 are queried $\mathcal{N}_{H_1}$ and $\mathcal{N}_{H_2}$ times respectively, and $\mathcal{N}_{Key}$ keys are generated through **CL.Get-Public-Key** and $\mathcal{N}_{TH_3}$ queries with the targeted $\mathtt{ID}_*$ and P_* are called. The challenger $\mathcal{C}$ randomly selects three indices $0 < \mathcal{I} \leq \mathcal{N}_{H_1}, 0 < \mathcal{J} \leq \mathcal{N}_{Key}, 0 < \mathcal{K} \leq \mathcal{N}_{TH_3}$. $\mathcal{C}$ maintains a tuple $\mathcal{T}$ in the form of $\langle \mathtt{ID}_i, P_i, \lambda_i, U_i, x_i, d_i, s_i\rangle$, which is indexed by $(\mathtt{ID}_i, P_i)$. $\mathcal{C}$ answers the queries as follows:

- **CL.Setup**(1^k). $\mathcal{C}$ follows the algorithm to compute $M_{\mathfrak{pk}}$ and $M_{\mathfrak{sk}}$, and passes the values to $\mathcal{A}_{II}$. In particular, $\mathcal{C}$ chooses a random $s \in \mathbb{Z}_q^*$ as $M_{\mathfrak{sk}}$ and sets $P_{KGC} = [s]G$. $\mathcal{C}$ randomly chooses four values $Z_*, h_* \in \{0,1\}^n$ and $\lambda_*, \lambda_{\mathcal{J}} \in \mathbb{Z}_q^*$.

- $H_1(PI\|\mathtt{ID}_A)$. $\mathcal{C}$ maintains a list $\mathcal{H}_1$ in the form of $\langle I_i, Z_i\rangle$. If the input is on the list, then the hash value is returned. If this is the $\mathcal{I}$-th distinctive query, then $\mathcal{C}$ puts $(PI\|\mathtt{ID}_A, Z_*)$ on the list, and returns Z_*. Otherwise, it randomly samples $Z_i \in \{0,1\}^n$ (if $Z_i = Z_*$, terminate the game (**Event 1**)), and returns Z_i after putting the pair into $\mathcal{H}_1$.

- $H_2(P_A\|Z)$. Similarly, $\mathcal{C}$ has a list $\mathcal{H}_2$ in the form of $\langle I_i, \lambda_i\rangle$. If the input is on the list, then the hash value is returned. Otherwise, it randomly samples $\lambda_i \in \mathbb{Z}_q^*$. If $\lambda_i \neq \lambda_*$ and $\lambda_i \neq \lambda_{\mathcal{J}}$, return λ_i after putting the pair into $\mathcal{H}_2$, else terminate the game (**Event 1**).

- $H_3(\lambda\|m)$. $\mathcal{C}$ maintains a list $\mathcal{H}_3$ in the form of $\langle I_i, h_i, u_i, v_i\rangle$. If the input is on the list, then the hash value h_i is returned. Otherwise, $\mathcal{C}$ behaves differently in the following cases:

  1. If $\lambda \neq \lambda_{\mathcal{J}}$ and $\lambda \neq \lambda_*$, randomly choose $h_i \in \{0,1\}^n$, return h_i after putting $(\lambda\|m, h_i, \perp, \perp)$ into $\mathcal{H}_3$.

2. Else (i.e., $\lambda = \lambda_*$ or $\lambda = \lambda_{\mathcal{J}}$), if the $\mathcal{J}$-th public key has not been generated, terminate the game (**Event 1**). $\mathcal{C}$ responds differently in the following cases.
   - In the $\mathcal{J}$-th public key generation $Z_i \neq Z_*$,
     * $\lambda = \lambda_*$. If this is the $\mathcal{K}$-th query, set $h_i = h_*$, else randomly select $h_i \in \mathbb{Z}_q^*$, after putting $(\lambda \| m,\ h_i,\ \perp,\ \perp)$ in the list, return h_i
     * $\lambda = \lambda_{\mathcal{J}}$. Randomly sample $(a_i, b_i) \in \mathbb{Z}_q^2$ and compute $u_i = \mathcal{F}([a_i]G + [b_i][h_*][\alpha]G + [b_i][\lambda_{\mathcal{J}} - \lambda_*][s]G)$, $v_i = u_i/b_i$, and $h_i = a_i u_i / b_i$. $\mathcal{C}$ returns h_i after putting $(\lambda \| m,\ h_i,\ u_i,\ v_i)$ into $\mathcal{H}_3$.
   - Otherwise, if this is the $\mathcal{K}$-th query with λ_*, then after putting $(\lambda \| m,\ h_*,\ \perp,\ \perp)$ in the list, return h_*. Otherwise, randomly sample $(a_i, b_i) \in \mathbb{Z}_q^{*2}$ and compute $u_i = \mathcal{F}([a_i]G + [b_i][h_*][\alpha]G)$, $v_i = u_i/b_i$, and $h_i = a_i u_i / b_i$. $\mathcal{C}$ returns h_i after putting $(\lambda \| m, h_i, u_i, v_i)$ into $\mathcal{H}_3$.

- **CL.Get-Public-Key**($\mathtt{ID}_i$,*bNewKey*). If *bNewKey* is false and at least one entry in $\mathcal{T}$ includes $\mathtt{ID}_i$, then $\mathcal{C}$ returns P_i in the latest entry of $\mathtt{ID}_i$ in $\mathcal{T}$. Otherwise, let $Z_i = H_1(PI \| \mathtt{ID}_i)$, and $\mathcal{C}$ responds differently in the following cases:

  1. If this is the $\mathcal{J}$-th public key generation in the game, then compute $P_i = [h_* \alpha]G - [s\lambda_*]G$. Put $(P_i \| Z_*, \lambda_*)$ in $\mathcal{H}_2$. If $Z_i = Z_*$, set $\lambda_i = \lambda_*$, else put $(P_i \| Z_i, \lambda_{\mathcal{J}})$ in $\mathcal{H}_2$ and set $\lambda_i = \lambda_{\mathcal{J}}$. If the list has an entry indexed by $(P_i \| Z_i)$ that has different value from λ_i, terminate the game (**Event 1**). Put $(\mathtt{ID}_i,\ P_i,\ \lambda_i,\ \perp,\ \perp,\ \perp,\ \perp)$ in $\mathcal{T}$. $\mathcal{C}$ returns P_i.
  2. Else, randomly select $x_i, d_i, \lambda_i \in \mathbb{Z}_q^*$ (if $\lambda_i = \lambda_*$ or $\lambda_i = \lambda_{\mathcal{J}}$, terminate the game (**Event 1**)), compute $P_i = [x_i]G + [d_i]G - [\lambda_i][s]G$, put $(P_i \| Z_i, \lambda_i)$ in $\mathcal{H}_2$ and $(\mathtt{ID}_i,\ P_i,\ Z_i,\ \lambda_i,\ [x_i]G,\ x_i,\ d_i,\ x_i + d_i)$ in $\mathcal{T}$. $\mathcal{C}$ returns P_i.

- **CL.Get-Private-Key**($\mathtt{ID}_i, P_i$). If there is no entry indexed by $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$, return error. Otherwise, if s_i of the found entry is $\perp$, then terminate the game (**Event 2**), or return s_i.

- **CL.Get-User-Key**($\mathtt{ID}_i, P_i$). If there is no entry indexed by $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$, return error. Otherwise, if x_i of the found entry is $\perp$, then terminate the game (**Event 3**), or return x_i.

- **CL.Get-Sign**($\mathtt{ID}_i, P_i, m_i$). If there is no entry indexed by $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$, return error. Otherwise, use λ_i from the found entry to query $H_3(\lambda_i \| m_i)$ and respond as follows:

  1. If s_i from the found entry is not $\perp$, then use s_i as the private key and P_i as the public key to sign the message and return signature.
  2. Else, use $\lambda_i \| m_i$ to search the list $\mathcal{H}_3$.
     - If u_i is $\perp$ on the found entry, then terminate the game (**Event 4**).
     - Else, return (u_i, v_i) as the signature.

- **CL.Verify**($M_{\mathfrak{pt}}, \mathtt{ID}_*, P_*, m_*, \sigma_*$). If $Z_* \neq H_1(PI \| \mathtt{ID}_*)$ or $\lambda_* \neq H_2(P_* \| Z_*)$ or $h_* \neq H_3(\lambda_* \| m_*)$, then terminate the game (**Event 5**). Otherwise, parse σ_* as (u_*, v_*) and output $(u_*, v_*/h_*)$.

It is easy to verify that if the game is not terminated prematurely, then the simulation is indistinguishable from the environment. In particular, if the targeted $\texttt{ID}_* \neq \texttt{ID}_{\mathcal{J}}$ in the **CL.Get-Public-Key** query, $\mathcal{C}$ still answers the **CL.Get-Sign**$(\texttt{ID}_{\mathcal{J}}, P_{\mathcal{J}}, m_i)$ properly. Precisely, $O_{\mathcal{J}} = P_{\mathcal{J}} + [\lambda_{\mathcal{J}}][s]G = [h_*\alpha]G + [\lambda_{\mathcal{J}} - \lambda_*][s]G$. According to **CL.Verify**, we have $v_i^1 = v_i^{-1}h_i = a_i$, $v_i^2 = v_i^{-1}u_i = b_i$, $Q'_i = [v_i^1]G + [v_i^2]O_{\mathcal{J}} = [a_i]G + [b_i][h_*][\alpha]G + [b_i][\lambda_{\mathcal{J}} - \lambda_*][s]G$. Hence, $x_{Q'_i} = u_i$, which means the signature is valid. The final output is the solution of the SLP. Let **Event 1** be that the hash collision happens on either H_1 or H_2. If the attacker chooses the $\mathcal{I}$-th identity and the $\mathcal{J}$-th public key and the $\mathcal{K}$-th message queried with λ_*, then **Event 2, 3, 4** and **5** won't happen. Hence, $\mathcal{C}$ solves the SLP with probability at least $\frac{\epsilon(k)}{\mathcal{N}_{H_1} \cdot \mathcal{N}_{Key} \cdot \mathcal{N}_{TH_3}} - 2\frac{\mathcal{N}_{H_2} + \mathcal{N}_{Key}}{q} - \frac{\mathcal{N}_{H_1}}{2^n}$ and time $O(t(k))$. ■

Lemma 3 *If there exists an adversary $\mathcal{A}_{Ia}$ that has a non-negligible probability of success in Game 1 against CL-PKS2 in the random oracle model, then the DLP in group $\mathbb{G}$ can be solved in polynomial time.*

The reduction in Lemma 2 can be modified easily for CL-PKS2 but still based on the strong semi-logarithm assumption. Applying the *Multiple-Forking Lemma* [10], the security of CL-PKS2 against $\mathcal{A}_{Ia}$ can be further reduced to the DLP. We skip the details.

Lemma 4 *If there exists an adversary $\mathcal{A}_{Ib}$ that has a non-negligible probability of success in Game 1 against CL-PKS2 in the random oracle model, then the DLP in group $\mathbb{G}$ can be solved in polynomial time.*

Proof. Suppose that in the game, H_2 and H_3 are queried $\mathcal{N}_{H_2}$ and $\mathcal{N}_{H_3}$ times respectively, and $\mathcal{A}_{Ib}$ wins the game with probability $\epsilon(k)$ in time $t(k)$. Given a DLP ($\mathbb{G}$, G, $[\alpha]G$), we use $\mathcal{A}_{Ib}$ to construct $\mathcal{C}$ to solve the problem. $\mathcal{C}$ maintains a tuple $\mathcal{T}$ in the form of $\langle \texttt{ID}_i, P_i, \lambda_i, U_i, x_i, d_i, s_i \rangle$, which is indexed by $(\texttt{ID}_i, P_i)$. $\mathcal{C}$ answers the queries as follows:

- **CL.Setup**(1^k). $\mathcal{C}$ sets $P_{KGC} = [\alpha]G$, and passes $M_{\mathfrak{pk}}$ to $\mathcal{A}_{Ib}$. $\mathcal{C}$ randomly chooses three values $Z_*, h_* \in \{0,1\}^n$ and $\lambda_* \in \mathbb{Z}_q^*$.
- $H_1(PI \| \texttt{ID}_A)$. $\mathcal{C}$ maintains a list $\mathcal{H}_1$ in the form of $\langle I_i, Z_i \rangle$. If the input is on the list, then the hash value is returned. Otherwise, it randomly samples $Z_i \in \{0,1\}^n$, and returns Z_i after putting the pair into $\mathcal{H}_1$.
- $H_2(P_A \| Z)$. Similarly, $\mathcal{C}$ has a list $\mathcal{H}_2$ in the form of $\langle I_i, \lambda_i \rangle$. If the input is on the list, then the hash value is returned. Otherwise, it randomly samples $\lambda_i \in \mathbb{Z}_q^*$, and returns λ_i after putting the pair into $\mathcal{H}_2$.
- $H_3(u \| \lambda \| m)$. $\mathcal{C}$ maintains a list $\mathcal{H}_3$ in the form of $\langle I_i, h_i \rangle$. If the input is on the list, then the hash value h_i is returned. Otherwise, randomly sample $h_i \in \{0,1\}^n$, return h_i after putting $(u \| \lambda \| m, h_i)$ into $\mathcal{H}_3$.
- **CL.Extract-Partial-Key**$(M_{\mathfrak{pk}}, M_{\mathfrak{sk}}, \texttt{ID}_i, U_i)$. $\mathcal{C}$ randomly selects $d_i, \lambda_i \in \mathbb{Z}_q^*$, and computes $P_i = [d_i]G + U_i - [\lambda_i][\alpha]G$. $\mathcal{C}$ puts $(P_i \| Z_i, \lambda_i)$ in $\mathcal{H}_2$ with $Z_i = H_1(PI \| \texttt{ID}_i)$ and returns (P_i, d_i).

- **CL.Get-Public-Key**($\mathtt{ID}_i$,*bNewKey*). If *bNewKey* is false and at least one entry in $\mathcal{T}$ includes $\mathtt{ID}_i$, then $\mathcal{C}$ returns P_i in the latest entry of $\mathtt{ID}_i$ in $\mathcal{T}$. Otherwise, randomly select $x_i, d_i, \lambda_i \in \mathbb{Z}_q^*$, compute $P_i = [x_i]G + [d_i]G - [\lambda_i][\alpha]G$, put $(P_i \| Z_i, \lambda_i)$ in $\mathcal{H}_2$ with $Z_i = H_1(PI \| \mathtt{ID}_i)$ and $(\mathtt{ID}_i, P_i, \lambda_i, [x_i]G, x_i, d_i, x_i + d_i)$ in $\mathcal{T}$.
- **CL.Get-Private-Key(**$\mathtt{ID}_i, P_i$**)**. If there is no entry indexed by $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$, return error. Otherwise, return s_i from the found entry.
- **CL.Get-User-Key(**$\mathtt{ID}_i, P_i$**)**. If there is no entry indexed by $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$, return error. Otherwise, return x_i from the found entry.
- **CL.Get-Sign**$(\mathtt{ID}_i, P_i, m_i)$. If there is no entry indexed by $(\mathtt{ID}_i, P_i)$ in $\mathcal{T}$, return error. Otherwise, use the found s_i as the private key and P_i as the public key to sign the message and return its signature.

$\mathcal{C}$ perfectly simulates the attacking environment. Before applying the *Multiple-Forking Lemma* [10] to argue the security, we make several assumptions, so to make the analysis simpler. First, as explained in Section 3, the use of H_1 is unnecessary, in the following analysis we assume H_1 to be a normal collision-resistance hash function and $\mathcal{C}$ requires the attacker to output Z instead of ID. This assumption is sound because the reduction above does not make use of any random oracle property of H_1. Second, in the attacking process $\mathcal{A}$ may query $H_3(u\|\lambda\|m)$ before querying $\lambda = H_2(P\|Z)$. However, as H_2 is simulated as a random oracle, there is only a negligible probability that this event has happened and at the same time σ is valid. We henceforth ignore this event in the analysis. $\mathcal{C}$ runs the multiple-forking algorithm $\mathbf{MF}_{\mathcal{A}_{Ib},3}(M_{\mathfrak{pk}})$ and gets four forgeries $(Z, m, P, (u_i, v_i))$, $i = 0, \ldots, 3$ for some Z, some message m and some P. Moreover each u_i corresponds to a point $Q_i = \pm[r_i]G$, and $u_0 = u_1$ and $u_2 = u_3$. If the forged signatures are valid, by assuming $Q_0 = [r_0]G$ and $Q_2 = [r_2]G$, we have

$$\begin{aligned}
Q_0 &= [v_0^{-1}h_0]G + [v_0^{-1}u_0](P + [\lambda_0][\alpha]G),\\
Q_0 &= [v_1^{-1}h_1]G + [v_1^{-1}u_0](P + [\lambda_0][\alpha]G),\\
Q_2 &= [v_2^{-1}h_2]G + [v_2^{-1}u_2](P + [\lambda_2][\alpha]G),\\
Q_2 &= [v_3^{-1}h_3]G + [v_3^{-1}u_2](P + [\lambda_2][\alpha]G).
\end{aligned}$$

Let $a_i = h_i/v_i$ for $i = 0, \ldots, 3$, $b_0 = -u_0/v_0$, $b_1 = -u_0/v_1$, $b_2 = -u_2/v_2$ and $b_0 = -u_2/v_3$. $\mathcal{C}$ computes α' as follows:

$$\alpha' = \frac{(a_0 - a_1)(b_2 - b_3) - (a_2 - a_3)(b_0 - b_1)}{(\lambda_0 - \lambda_2)(b_0 - b_1)(b_2 - b_3)}.$$

If $Q_0 = -[r_0]G$ or $Q_2 = -[r_2]G$, $\mathcal{C}$ can compute α' in a similar way and test its correctness by checking if $[\alpha']G = [\alpha]G$ and find the solution to the DLP. By the *Multiple-Forking Lemma*, $\mathcal{C}$ solves the DLP with probability $O(\frac{\epsilon^4(k)}{(\mathcal{N}_{H_2}+\mathcal{N}_{H_3})^6})$ and time $O(t(k))$. [2] ■

Theorem 4 *If there exists an adversary $\mathcal{A}_I$ that has a non-negligible probability of success in Game 1 against CL-PKS2 in the random oracle model, then the DLP in group $\mathbb{G}$ can be solved in polynomial time.*

[2]With the help of λ in H_3, a tighter reduction could be established but with much more complicated analysis.

Theorem 4 follows from Lemma 3 and 4.

Theorem 5 *If there exists an adversary $\mathcal{A}_{II}$ that has a non-negligible probability of success in Game 2 against CL-PKS2 in the random oracle model, then the DLP in group $\mathbb{G}$ can be solved in polynomial time.*

The reduction in Theorem 3 can be simply modified for CL-PKS2 but still based on the strong semi-logarithm assumption. Applying the *Multiple-Forking Lemma*, the security of CL-PKS2 against $\mathcal{A}_{II}$ can be reduced to the DLP. We skip the details.

Similar techniques used in the reductions for CL-PKS2 can be applied to analyze CL-PKS3 and we have following results. Again the use of λ in H_3 can help to construct tighter reductions.

Theorem 6 *If there exists an adversary $\mathcal{A}_I$ that has a non-negligible probability of success in Game 1 against CL-PKS3 in the random oracle model, then the DLP in group $\mathbb{G}$ can be solved in polynomial time.*

Theorem 7 *If there exists an adversary $\mathcal{A}_{II}$ that has a non-negligible probability of success in Game 2 against CL-PKS3 in the random oracle model, then the DLP in group $\mathbb{G}$ can be solved in polynomial time.*

Overall, CL-PKS2 and CL-PKS3 are secure schemes with regard to Definition 2 in the random oracle model based on the DL assumption. With two results from Lemma 2 and Theorem 3, CL-PKS1 still lacks a formal security analysis against the $\mathcal{A}_{Ib}^{-}$ adversary without resorting to the generic group model or introducing new complexity assumption. On the other hand, the argument in Section 4.3 has demonstrated its security strength against potential attacks. Particularly, with the result of Lemma 1, it is shown that CL-PKS1 is more secure than ECQV+ECDSA.

6 Performance Evaluation and Application

We first compare the proposed CL-PKS schemes with the related schemes including existing CL-PKS schemes and standard signature schemes using implicit certificates. Many CL-PKS schemes with or without pairing are proposed in the literature. Pairing (denoted by $\boldsymbol{P}$, which is a bilinear map: $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$ such that $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic groups and $\mathbb{G}_3$ is a related extension field) is a much heavier computation operation than the point scalar (denoted by S) or exponentiation (denoted by E) in field $\mathbb{G}_3$. We don't list all the existing CL-PKS schemes. Instead, only some commonly referred pairing-based schemes and some most efficient pairing-free schemes are compared. $|\mathbb{G}|$ and $|q|$ denote the bit length of the size of a group $\mathbb{G}$ and an integer q respectively.

According to Table 8, it is known that our schemes are among the most efficient ones. One may compute O_A first and then call ECDSA verification process so to for example make use of the hardware implementation of the algorithm. When a pre-computation of point scalar of P_{KGC} is implemented, computing O_A in CL-PKS1 could be six to eight times faster than the one in ECQV+ECDSA. Moreover, CL-PKS1 doesn't suffer from the Kravitz attack that affects ECQV+ECDSA.

As explained, CL-PKS1 can be realized by reusing the existing implementation of ECDSA. This is a particularly important advantage in practice because many security

Table 8: Performance Comparison

Scheme	Key size		Computation		Signature
	Private	Public	Signing	Verification	size
AP[1]	$\|\mathbb{G}_1\|$	$2\|\mathbb{G}_1\|$	$1\mathbf{P}+3S$	$4\mathbf{P}+1E$	$\|\mathbb{G}_1\|+\|q\|$
CPHL[19]	$\|\mathbb{G}_1\|$	$\|\mathbb{G}_1\|$	$2S$	$2\mathbf{P}+2S$	$2\|\mathbb{G}_1\|$
HMSWW[30]	$\|q\|+\|\mathbb{G}_1\|$	$\|\mathbb{G}_1\|$	$1S$	$3\mathbf{P}$	$\|\mathbb{G}_1\|$
ZWXF[52]	$\|q\|+\|\mathbb{G}_1\|$	$\|\mathbb{G}_1\|$	$3S$	$4\mathbf{P}$	$2\|\mathbb{G}_1\|$
ZZZ[53]	$\|q\|+\|\mathbb{G}_1\|$	$\|\mathbb{G}_2\|$	$1S+2E$	$1\mathbf{P}+3E$	$\|\mathbb{G}_1\|+2\|q\|$
HRL[27]	$\|q\|$	$\|q\|+\|\mathbb{G}\|$	$1S$	$5S$	$2\|\mathbb{G}\|$
HCZ[29]	$\|q\|$	$2\|\mathbb{G}\|$	$1S$	$3S$	$\|\mathbb{G}\|+\|q\|$
JHLC[34]	$\|q\|$	$2\|\mathbb{G}\|$	$1S$	$3S$	$\|\mathbb{G}\|+\|q\|$
LXWHH[37]	$2\|q\|$	$2\|\mathbb{G}\|$	$1S$	$3S$	$2\|q\|$
YSCC[51]	$\|q\|$	$\|\mathbb{G}\|$	$1S$	$3S$	$\|\mathbb{G}\|+\|q\|$
PH/OMC+ ECDSA[42, 14]	$\|q\|$	$\|\mathbb{G}\|$	$1S$	$3S$	$2\|q\|$
Arazi/ECQV+ ECDSA[5, 17]	$\|q\|$	$\|\mathbb{G}\|$	$1S$	$3S$	$2\|q\|$
CL-PKS1	$\|q\|$	$\|\mathbb{G}\|$	$1S$	$3S$	$2\|q\|$
CL-PKS2	$\|q\|$	$\|\mathbb{G}\|$	$1S$	$3S$	$\|p\|+\|q\|$
CL-PKS3	$\|q\|$	$\|\mathbb{G}\|$	$1S$	$3S$	$\|\mathbb{G}\|+\|q\|$
CL-PKS4	$\|q\|$	$\|\mathbb{G}\|$	$1S$	$3S$	$2\|q\|$

elements (SE) have ECDSA embedded and the private key is protected within the SE. Deploying CL-PKS1 doesn't need to modify existing hardware chips and won't cause extra security concerns because the signing process can use the private key stored in SE in the same way as ECDSA.

We have implemented CL-PKS1 on the 32-bit Cortex-M4 MCU STM32F4 to evaluate the performance. STMicroelectronics provides a crypto library [47], which has interfaces to access the implementation of ECDSA and point scalar operation over the NIST p256 elliptic curve. The signing process of CL-PKS1 can directly call ECDSA signature generation function in the library by signing on $(\lambda\|m)$. The verification process first calls the scalar and addition operations to compute O_A and then calls the verification function of ECDSA in the library. We have also implemented CL-PKS1 from scratch to evaluate the performance of a native implementation of the scheme. In the implementation, the Montgomery modular is applied to compute multiplication in $\mathbb{F}_p$. The addition and multiplication operations are implemented with the assembly language. The code is compiled with -O3 option and speed is measured with STM32F4 working at 168MHz.

Table 9: Implementation of CL-PKS1 on STM32F4

Implementation	STM crypto lib.	Our software
Code size	15K	11K
Stack size	0.5K	0.7K
Signing time	0.078s	0.058s
Verification time	0.076s(scalar)+0.104s(ECDSA ver.)	0.132s

Our software implementation is even faster than the one using the library provided by STMicroelectronics. The speed of the implementation appears quick enough for most applications.

Systems employing CL-PKS will enjoy the benefit of lightweight key management. For example, inter-domain authentication in the Internet of Things such as V2X communication [16] requires PKC-based security solutions. Considering the constrained resource, diversity of devices and the scale of the IoT, an efficient CL-PKS scheme like CL-PKS1 offers clear advantages over the certificate-based, identity-based, and raw public key with out-of-band validation (RPK-OOBV) solutions. The certificate size and the complicated validation process could quickly drain available resources of a constrained device (see [46] for a detailed evaluation of the impact of a certificate on IoT devices). The RPK-OOBV has small public key data but requires other validation mechanisms such as DNSSEC. On the other hand, the proposed CL-PKS has small key size as RPK-OOBV and removes the necessity of public key validation. With only slightly larger communication overhead by including the public key P_A as part of a signature as suggested in [7], CL-PKS can work just like an IBS but is free from the key-escrow concern. Certainly, CL-PKS1 can play the role of ECQV+ECDSA in [16] to achieve better security and possibly a higher performance if the public key calculation process can make use of point scalar pre-computation with P_{KGC}.

7 Conclusion

In this work, we redefine the formulation of CL-PKC to unify it with security mechanisms using implicit certificates. We then construct a CL-KGA from the Schnorr signature and prove its security in the random oracle model. Furthermore, we demonstrate that using the assignment computed in the CL.Extract-Partial-Key process as the key prefixing in the message signing process helps improve the security of a CL-PKS that combines a secure CL-KGA with a standard signature algorithm. Several of such CL-PKS schemes are described. CL-PKS1 can be implemented based on existing security elements that support ECDSA. Security analysis shows that CL-PKS1 has stronger security than the composition of ECQV with ECDSA. CL-PKS2 and CL-PKS3 have full security reductions based on the discrete logarithm assumption in the random oracle model. The results presented in the work may also shed light on the way of using ECQV with ECDSA. With little cost, the security of the ECQV-based signature scheme can benefit from the key prefixing technique. However, whether using the assignment as the key prefixing allows universal composability of a secure CL-KGA with a EUF-CMA-secure DSA, which fits in with the general framework defined in ISO/IEC 14888-3, remains an open problem.

Acknowledgments

Liqun Chen's work is supported by the EU Horizon 2020 research and innovation program under grant agreement No 779391 (FutureTPM).

References

[1] S. S. Al-Riyami and K. G. Paterson. Certificateless Public Key Cryptography. In *Proc. of Asiacrypt 2003*, LNCS 2894, pp. 452–473, 2003.

[2] S. S. Al-Riyami and K. G. Paterson. CBE from CL-PKE: a Generic Construction and Efficient Schemes. In *Proc. of PKC 2005*, LNCS 3386, pp. 398–415, 2005.

[3] J. H. An, Y. Dodis, and T. Rabin. On the Security of Joint Signature and Encryption. In *Proc. of Eurocrypt 2002*, LNCS 2332, pp. 83–107, 2002.

[4] A. Antipa, D. Brown, A. Menezes, and et al. Validation of Elliptic Curve Public Keys. In *Proc. of PKC 2003*, LNCS 2567, pp. 211–223, 2003.

[5] B. Arazi. Certification of DL/EC Keys. Submission to P1363 meeting, http://grouper.ieee.org/groups/1363/StudyGroup/contributions/arazi.doc, 1998.

[6] J. Baek, R. Safavi-Naini, and W. Susilo. Certificateless Public Key Encryption without Pairing. In *Proc. of ISC 2005*, LNCS 3650, pp. 134–148, 2005.

[7] M. Bellare, C. Namprempre, and G. Neven. Security Proofs for Identity-Based Identification and Signature Schemes. In *J. of Cryptology*, Volume 22, pp. 1–61, 2009.

[8] K. Bentahar, P. Farshim, J. Malone-Lee, and N. P. Smart. Generic Constructions of Identity-Based and Certificateless KEMs. In *J. of Cryptology*, Volume 21, pp. 178–199, 2008.

[9] D. J. Bernstein. Multi-User Schnorr Security, Revisited. Cryptology ePrint Archive, Report 2015/996, 2015.

[10] A. Boldyreva, A. Palacio, and B. Warinschi. Secure Proxy Signature Schemes for Delegation of Signing Rights. In *J. of Cryptology*, Volume 25, 57–115, 2012.

[11] E. Brickell, D. Pointcheval, S. Vaudenay, and M. Yung. Design Validations for Discrete Logarithm Based Signature Schemes. In *Public Key Cryptography 2000*, LNCS 1751, pp. 276–292, 2000.

[12] D. Brown. Generic Groups, Collision Resistance, and ECDSA. In *Designs, Codes and Cryptography*, Volume 35, pp. 119–152, 2005.

[13] D. Brown. On the Provable Security of ECDSA. In *Advances in Elliptic Curve Cryptography*, pp. 21–40. Cambridge University Press (2005).

[14] D. Brown, M. Campagna, and S. Vanstone. Security of ECQV-Certified ECDSA Against Passive Adversaries. Cryptology ePrint Archive, Report 2009/620, 2009.

[15] D. Brown, R. Gallant, and S. Vanstone. Provably Secure Implicit Certificate Schemes. In *Proc. of FC 2001*, LNCS 2339, pp. 156–165, 2001.

[16] B. Brecht, D. Therriault, A. Weimerskirch, W. Whyte, V. Kumar, T. Hehn, and R. Goudy. *A Security Credential Management System for V2X Communications.* IEEE Transactions on Intelligent Transportation Systems, Vol. 19, no. 12, pp. 3850–3871, 2018.

[17] Certicom Research. SEC 4: Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV). Version 1.0. 2013.

[18] Z. Cheng, L. Chen, L. Ling, and R. Comley. General and Efficient Certificateless Public Key Encryption Constructions. In *Proc. of Pairing 2007*, LNCS 4575, pp. 83–107.

[19] K. Y. Choi, J. H. Park, J. Y. Hwang, and D. H. Lee. Efficient Certificateless Signature Schemes. In *Proc. of ACNS 2007*, LNCS 4521, pp. 443–458, 2007

[20] M. Fersch, E. Kiltz, and B. Poettering. On the One-Per-Message Unforgeability of (EC)DSA and its Variants. In *Proc. of TCC 2017*, LNCS 10678, pp. 519–534, 2017.

[21] GB/T 32918.2-2017. Public Key Cryptographic Algorithm SM2 Based on Elliptic Curves– Part 2: Digital Signature Algorithm.

[22] C. Gentry. Certificate-Based Encryption and the Certificate Revocation Problem. In *Proc. of Eurocrypt 2003*, LNCS 2656, pp. 272–293, 2003.

[23] M.Girault. Self-certified Publick Keys. In *Proc. of Eurocrypt 91*, LNCS 547, pp. 490–497, 1991.

[24] M. Groves. Elliptic Curve-Based Certificateless Signatures for Identity-Based Encryption (ECCSI). RFC 6507, 2012.

[25] GSM Association. RSP Technical Specification, Version 2.0. 2016.

[26] GlobalPlatform Card. Secure Channel Protocol '11' Card Specification v2.3 - Amendment F, Version 1.1. 2017

[27] L. Harn, J. Ren, and C. Lin. Design of DL-based Certificateless Digital Signatures. In *J. of Systems and Software*, Volume 82, Issue 5, pp. 789–793, 2009.

[28] D. He, Y. Chen, J. Chen, and et al. A New Two-Round Certificateless Authenticated Key Agreement Protocol without Bilinear Pairings. In *Mathematical and Computer Modelling*, Volume 54, Issues 11–12, pp. 3143–3152, 2011.

[29] D. He, J. Chen, and R. Zhang. An Efficient and Provably-secure Certificateless Signature Scheme without Bilinear Pairings. In *Int. J. of Communication Systems*, vol. 25, no. 11, pp. 1432–1442, 2012.

[30] X. Huang, Y. Mu, W. Susilo, and et al. Certificateless Signature Revisited. In *Proc. of ACISP 2007*, LNCS 4586, pp. 308–322, 2007.

[31] X. Huang, W. Susilo, Y. Mu, and F. Zhang. On the Security of Certificateless Signature Schemes from Asiacrypt 2003. In *Proc. of CANS 2005*, pp. 13–25, 2005.

[32] ISO/IEC. Information Technology – Secruity Techniques – Digital Signatures with Appendix – Part 3: Discrete Logarithm Based Mechanisms. *ISO/IEC 14888-3:2016.*

[33] ISO/IEC. Information Technology – Security Techniques – Encryption Algorithms – Part 2: Asymmetric Ciphers. *ISO/IEC 18033-2:2006.*

[34] X. Jia, D. He, Q. Liu, and K.-K. R. Choo. An Efficient Provably-Secure Certificateless Signature Scheme for Internet-of-Things Deployment. *Ad Hoc Networks*, Vol 71, pp 78-87, 2018.

[35] J. Lai and W. Kou. Self-Generated-Certificate Public Key Encryption without Pairing. In *Proc. of PKC 2007*, LNCS 4450, pp. 476–489, 2007.

[36] G. Lippold, C. Boyd, and J. G. Nieto. Strongly Secure Certificateless Key Agreement. In *Proc. of Pairing 2009*, LNCS 5671, pp. 206–230, 2009.

[37] W. H. Liu, Q. Xie, S. B. Wang, and et al. Pairing-Free Certificateless Signature with Security Proof. In *J. of Computer Networks and Communications*, Volume 2014.

[38] J. Malone-Lee and N. Smart. Modifications of ECDSA. In *Proc. of Selected Areas in Cryptography – SAC 2002*, LNCS 2595, pp. 1–12, 2003.

[39] A. Menezes and N. P. Smart. Security of Signature Schemes in a Multi-User Setting. In *Designs, Codes and Cryptography* 33, pp. 261–274, 2004.

[40] H. Morita, J. Schuldt, T. Matsuda, and et al. On the Security of the Schnorr Signature Scheme and DSA Against Related-Key Attacks. In *Proc. of ICISC 2015*, LNCS 9558, pp. 20–35, 2015.

[41] NIST SP 800-56A. Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography, Revision 2. May 2013

[42] H. Petersen and P. Horster. Self-certified Keys – Concepts and Applications. In *Proc. of Communications and Multimedia Security*, pp. 102–116 IFIP Advances in Information and Communication Technology, Springer, 1997.

[43] L. A. Pintsov and S. A. Vanstone. Postal Revenue Collection in the Digital Age. In *Financial Cryptography – FC 2000*, LNCS 1962, pp. 105–120, 2001.

[44] D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. In *J. of Cryptology*, 13(3):361–396, 2000.

[45] V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In *Proc. of Eurocrypt '97*, LNCS 1233, pp. 256–266, 1997.

[46] H. Shafagh. Leveraging Public-key-based Authentication for the Internet of Things. Master thesis, https://www.inf.ethz.ch/personal/mshafagh/master_thesis_Hossein_Shafagh_PKC_in_the_IoT.pdf.

[47] STMicroelectronics. UM1924: STM32 Crypto Library. http://www.st.com/resource/en/user_manual/dm00215061.pdf.

[48] Y. Sun, F. Zhang, and J. Baek. Strongly Secure Certificateless Public Key Encryption without Pairing. In *Proc. of CANS 2007*, LNCS 4856, pp. 194–208, 2007.

[49] M. Tian and L. Huang. Cryptanalysis of a Certificateless Signature Scheme without Pairings. In *Int. J. Communication Systems*, 26(11): pp. 1375–1381, 2013.

[50] S. Vaudenay. Digital Signature Schemes with Domain Parameters. In *Proc. of ACISP 2004*, LNCS 3108, pp. 188–199, 2004.

[51] K.-H. Yeh, C. H. Su, K.-K. R. Choo, and W. Chiu. A Novel Certificateless Signature Scheme for Smart Objects in the Internet-of-Things Deployment. In *Sensors*, 2017, 17, 1001.

[52] Z. Zhang, D. Wong, J. Xu, and D. Feng. Certificateless Public-Key Signature: Security Model and Efficient Construction. In *Proc. of ACNS 2006*, LNCS 3989, pp. 293–308, 2006.

[53] L. Zhang, F. T. Zhang, and F. G. Zhang. New Efficient Certificateless Signature Scheme. In *Proc. of the EUC Workshops*, LNCS 4809, pp. 692–703, 2007.

[54] ZigBee Alliance. ZigBee Smart Energy Standard. Revision 19, Version 1.2a, 2014.

8 Appendix

8.1 The ECQV Implicit Certificate Scheme as a CL-KGA

For reference, we reprint the ECQV implicit certificate scheme following the description in [17] under the formulation of CL-KGA. ECQV uses the same **CL.Setup** and **CL.Set-User-Key** as the one in Section 3.

CL.Extract-Partial-Key($M_{\mathfrak{pk}}$, $M_{\mathfrak{sk}}$, $\mathtt{ID}_A$, U_A)

1. $w \in_R \mathbb{Z}_q^*$.
2. $X = [w]G$.
3. $W = U_A + X$.
4. $Cert_A = Encode(W, \mathtt{ID}_A, *)$.
5. $\lambda = H_2(Cert_A)$.
6. $t = (s + \lambda \cdot w) \mod q$.
7. Output $(W_A = W, d_A = t)$.

CL.Set-Private-Key($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, U_A, x_A, W_A, d_A)

1. $Cert_A = Encode(W_A, \mathtt{ID}_A, *)$.

2. $\lambda = H_2(Cert_A)$.

3. Output $s_A = (\lambda x_A + d_A) \mod q$.

CL.Set-Public-Key($M_{\mathfrak{pk}}$, $\texttt{ID}_A$, U_A, W_A)

1. Output $P_A = W_A$.

CL.Calculate-Public-Key($M_{\mathfrak{pk}}$, $\texttt{ID}_A$, P_A)

1. $Cert_A = Encode(P_A, \texttt{ID}_A, *)$.

2. $\lambda = H_2(Cert_A)$.

3. $O_A = [\lambda]P_A + P_{KGC}$.

CL.Verify-Key($M_{\mathfrak{pk}}$, $\texttt{ID}_A$, P_A, s_A)

1. $Cert_A = Encode(P_A, \texttt{ID}_A, *)$.

2. $\lambda = H_2(Cert_A)$.

3. $P'_A = [1/\lambda]([s_A]G - P_{KGC})$.

4. Output **valid** if $P_A = P'_A$, and **invalid** otherwise.

8.2 Another CL-KGA

Here we describe another CL-KGA scheme in which **CL.Extract-Partial-Key** uses U_A as part of $\texttt{ID}_A$ to generate d_A as suggested in [1]. Function **CL.Setup** and **CL.Set-User-Key** remain unchanged as the one in Section 3.

CL.Extract-Partial-Key($M_{\mathfrak{pk}}$, $M_{\mathfrak{sk}}$, $\texttt{ID}_A \| U_A$, $\varnothing$)

1. $Z = H_1(a\|b\|x_G\|y_G\|x_{P_{KGC}}\|y_{P_{KGC}}\|\texttt{ID}_A\|x_{U_A}\|y_{U_A})$.

2. $w \in_R \mathbb{Z}_q^*$.

3. $W = [w]G$.

4. $\lambda = H_2(x_W\|y_W\|Z)$.

5. $t = (w + \lambda \cdot s) \mod q$.

6. Output $(W_A = W, d_A = t)$.

CL.Set-Private-Key($M_{\mathfrak{pk}}$, $\texttt{ID}_A$, U_A, x_A, W_A, d_A)

1. Output $s_A = (x_A + d_A) \mod q$.

CL.Set-Public-Key($M_{\mathfrak{pk}}$, $\texttt{ID}_A$, U_A, W_A)

1. Output $P_A = (U_A, W_A)$.

CL.Calculate-Public-Key($M_{\mathfrak{pk}}$, $\texttt{ID}_A$, P_A)

1. Parse P_A as (U_A, W_A).
2. $Z = H_1(a\|b\|x_G\|y_G\|x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A\|x_{U_A}\|y_{U_A})$.
3. $\lambda = H_2(x_{W_A}\|y_{W_A}\|Z)$.
4. $O_A = U_A + W_A + [\lambda]P_{KGC}$.

CL.Verify-Key($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, s_A)

1. Parse P_A as (U_A, W_A).
2. $Z = H_1(a\|b\|x_G\|y_G\|x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A\|x_{U_A}\|y_{U_A})$.
3. $\lambda = H_2(x_{W_A}\|y_{W_A}\|Z)$.
4. $W'_A = [s_A]G - U_A - [\lambda]P_{KGC}$.
5. Output **valid** if $W_A = W'_A$, and **invalid** otherwise.

Compared with the keypair generation process in Section 3, **CL.Extract-Partial-Key** here treats U_A as part of $\mathtt{ID}_A$ as proposed in [1] and the published public key P_A now has two points. The input to **CL.Set-Public-Key** in [1] has x_A and does not include the output of **CL.Extract-Partial-Key**, so it is unable to include W in P_A. Hence, the construction here does not fully fit with the Al-Riyami-Paterson formulation.

8.3 CL-PKS from SM2-DSA

Here we present a CL-PKS scheme based upon the SM2 digital signature algorithm [21] as in Table 10. The key generation process makes use of the CL-KGA in Section 3. Here, we use $Z\|x_{P_A}\|y_{P_A}$ instead of λ as the key prefixing, and this makes the algorithm closer to the plain SM2.

Table 10: CL-PKS4 from SM2-DSA

CL.Sign($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, s_A, m)	**CL.Verify**($M_{\mathfrak{pk}}$, $\mathtt{ID}_A$, P_A, m, σ)
1. $Z = H_1(a\|b\|x_G\|y_G\|x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$.	1. $Z = H_1(a\|b\|x_G\|y_G\|x_{P_{KGC}}\|y_{P_{KGC}}\|\mathtt{ID}_A)$.
2. $e = H_3(Z\|x_{P_A}\|y_{P_A}\|m)$.	2. $\lambda = H_2(x_{P_A}\|y_{P_A}\|Z)$.
3. $r \in_R \mathbb{Z}_q^*$.	3. $O_A = P_A + [\lambda]P_{KGC}$.
4. $Q = [r]G$.	4. $e = H_3(Z\|x_{P_A}\|y_{P_A}\|m)$.
5. $u = (e + x_Q) \mod q$.	5. $t = (u + v) \mod q$.
6. $v = (1 + s_A)^{-1} \cdot (r - u \cdot s_A) \mod q$.	6. $Q' = [v]G + [t]O_A$.
7. Output $\sigma = (u, v)$.	7. $u' = (e + x_{Q'}) \mod q$.
	8. Output **valid** if $u = u'$, and **invalid** otherwise.

In **CL.Sign**, step 3-7 is exactly the signing process on a message digest e in SM2 and in **CL.Verify** step 5-8 is the verification process in SM2 on a signature with respect to a message digest e and public key O_A. We note that if the position of P_{KGC} and W in the operation to generate Z and λ is switched in the CL-KGA, then e could be computed without input P_A.