Hindawi
Security and Communication Networks
Volume 2021, Article ID 6695964, 11 pages
https://doi.org/10.1155/2021/6695964

Research Article

CLE against SOA with Better Data Security Storage to Cloud 5G

Huige Wang,[1] **Xing Chang**,[2] **and Kefei Chen**[3,4]

[1]*Network and Security Department, Anhui Science and Technology University, Bengbu 233030, China*
[2]*Microsoft Asia-Pacific Technology Co., Ltd., Shanghai 200240, China*
[3]*Department of Mathematics, Hangzhou Normal University, Hangzhou 311121, China*
[4]*Westone Cryptologic Research Center, Beijing 100070, China*

Correspondence should be addressed to Huige Wang; whgexf@163.com

Received 25 December 2020; Revised 4 March 2021; Accepted 9 April 2021; Published 28 May 2021

Academic Editor: Yinghui Zhang

Cloud 5G and Cloud 6G technologies are strong backbone infrastructures to provide high data rate and data storage with low latency for preserving QoS (Quality of Service) and QoE (Quality of Experience) in applications such as driverless vehicles, drone-based deliveries, smart cities and factories, remote medical diagnosis and surgery, and artificial-intelligence-based personalized assistants. There are many techniques to support the aforementioned applications, but for privacy preservation of Cloud 5G, the existing methods are still not sufficient. Public key encryption (PKE) scheme is an important means to protect user data privacy in Cloud 5G. Currently, the most common PKE used in Cloud 5G is CCA or CPA secure ones. However, its security level maybe not enough. SOA security is a stronger security standard than CPA and CCA. Roughly speaking, PKE with SOA security means that the adversary is allowed to open a subset of challenger ciphertexts and obtains the corresponding encrypted messages and randomness, but the unopended messages and randomness remain secure in the rest of the challenger ciphertexts. Security against SOA in PKEs has been a research hotspot, especially with the wide discussion in Cloud 5G. We revisited the SOA-CLE and proposed a new security proof, which is more concise and user friendly to understand privacy preservation in Cloud 5G applications.

1. Introduction

Cloud 5G achieves high data transmission speed, large data storage, and low latency mobile communication. According to the inherent property of electromagnetic waves: the higher the frequency, the shorter the wavelength, so it tends to propagate like a straight line. From the last few years, we have witnessed a paradigm shift with a major focus on mission critical applications and ultra-reliable low latency applications (URLCC) such as AR/VR, autonomous vehicles, e-healthcare, smart education, and so on, the aim of which is to provide QoS (Quality of Service) and QoE (Quality of Experience) to the end users with high data storage and low latency. Starting from driverless vehicles and drone-based deliveries, smart cities and factories, remote medical diagnosis and surgery, and artificial-intelligence-based personalized assistants, there is enormous number of applications around us which require strong network backbone infrastructure for QoS and QoE preservation. Based on the above applications and the advantages in Cloud 5G, in the years to come, Cloud 5G and Cloud 6G technologies are expected to provide high data rate with low latency and large data storage for preserving QoS and QoE. Although there are many techniques in the literature which can resolve these issues, the existing methods are still not sufficient to privacy preservation in the application in Cloud 5G. Hence, secure protocols and encryption schemes are required to resolve the aforementioned issues. Public key encryption (PKE) scheme is an important means to protect user data privacy in Cloud 5G. Currently, the most commonly used means to protect user data privacy is CPA (chosen-plaintext attacks) or CCA (chosen-ciphertext attacks) secure PKEs where the latter provides the decryption queries and thus is stronger than the former. However, SOA is a stronger security standard than CCA because the SOA security allows additional opening partial ciphertexts. Specially, in particular, due to the inherent advantages of certificateless public key (CLE), it solves the certificate

management problem in the traditional public key cryptography and the key-escrow problem [8] in IBE schemes. Security against SOA in CLEs has been a research hotspot, especially with the wide discussion in Cloud 5G [11, 12]. In this paper, we focus on the research on the SOA secure CLE.

The definition of SOA was first proposed by Dwork et al. at FOCS99 [4], which is an important target to measure the security of PKE. SOA security mainly applies to multiple-user settings where a subset of the challenge ciphertexts is allowed to open for the adversary. From the opened ciphertexts, the adversary can get not only the message but also the randomness. The question that we want to solve is how to make the remaining unopened ciphertexts secure? Following Dwork's work, SOA secure IBE and public key encryption (PKE) with SOA security have been widely developed [2, 5, 7]. CLE is another form of public key encryption system. Compared with IBE and PKE, CLE has the advantages of removing the certificate management in PKI-based PKE and key escrow in IBE. However, the study on CLE with SOA security is still rare.

1.1. Motivation. In the CLE system, a user's private key is jointly generated by the KGC and the user. The user's public key is generated by using the secret value generated by itself instead of the identity information. Obviously, compared with PKI-based PKE (hereafter, we abbreviated "PKI-based PKE" as "PKE") and IBE, CLE removes the disadvantages that exist in both schemes, namely, the certificate transaction in PKE and key escrow in IBE. Due to the merits of this notion, many CLEs with various security models (e.g., IND-CPA [9] and IND-CCA [1, 13]) were presented. As in PKE and IBE settings, implementing SOA security in CLE is also important. However, the particular security model makes constructing CLEs with SOA security more intractable. With more and more applications for CLE (such as cloud computing), implementing SOA security in CLE becomes more and more critical. In 2016, Wang et al. proposed an SOA secure CLE [14] under the standard DDH assumptions where the scheme is user friendly in construction and more efficient in practical applications. Recently, the relative discussions about Cloud 5G have become a new research focus, especially its data security and privacy protection. Due to the notable efficiency and security level, SOA secure CLE has been regarded as one of the most practical candidate encryption algorithms for Cloud 5G. However, we find that there are still some disadvantages needed to avoid such as complex security proof and obscure proof process. Based on this, we revisited the scheme in [14] and improved the security proof to make it more concise and easier to understand.

1.2. Reviewing the Contribution in [14]. In the scheme of [3], the authors proposed a one-sided publicly opening identity-based encryption scheme (1SPO-IBE) and, based on which, constructed an IBE scheme with SOA security. Adopting the similar method, the authors in [14] resolved the SOA security in CLE. More concretely, they first proposed a one-sided publicly opening certificateless encryption scheme (1SPO-CLE). Then, based on the proposed 1SPO-CLE, they presented a CLE scheme that is SOA secure in the case of two-type adversary model (i.e., CLE security model where an adversary refers to a user who is granted the ability to change the public key but does not know the master key; another one means the malicious KGC, who is not granted the ability to change the public key but knows the master secret key). The core idea is that we first combined one-bit CLE and 1SPO to generate a 1SPO-CLE with IND-CPA security in the CLE settings and then showed that a multi-bit CLE scheme with SOA security can be constructed from the 1SPO-CLE scheme under the one-time signature and CDH assumptions.

1.3. Revisiting the Reduction from SOA to CPA in [14]. In [14], the authors constructed an IND-CPA secure 1SPO-CLE scheme by combining the 1SPO and one-bit CLE scheme. A CLE scheme that encrypts 1 bit messages is called 1SPO if it is possible, given the public parameter par, public key PK_{id}, and the ciphertext c that encrypts message 0 with the randomness r to efficiently open the ciphertext c into another randomness used to encrypt message 1. In particular, the opening process is required to be done without any secret information. Furthermore, they proved that if the 1 bit 1SPO-CLE is IND-CPA secure, then the multi-bit CLE from it is SOA secure. Specifically, the encryption process is performed as follows. If the message is 1, then the encryption process follows specific rules and the correctness of the resulted ciphertext can be checked with some secret information; otherwise, the generated ciphertext is sampled randomly and uniformly from the ciphertext space. As stated in [3], the domain used as the ciphertext space is also required to have the property of sampleability and invertible sampleability in order to guarantee that the resulted scheme has the property of 1SPO.

1.4. Revisiting 1SPO-CLE Construction in [14]. In [14], the authors gave a concrete construction based on one-time signature and CDH assumptions. Specifically, the 1SPO-CLE is designed as follows. Assume G_1 and G_2 are both sampleable and invertibly sampleable domains as in [3]. If the encrypted message is 1, then the encryption of 1 is processed as $c = (c_1, c_2, \sigma, \mathrm{svk}) \longleftarrow \mathrm{Encrypt}_{ex}(\mathrm{par}, \mathrm{PK}_{id}, 1)$, where par is the public parameter and PK_{id} is the public key, and the first two values c_1, c_2 have certain structure and the value σ is a signature for certain medians generated in the encryption, while the last value svk is the signature verification key. If the encrypted message is 0, then the first three elements of its encryption are all random. In particular, if c is an encryption of 1, then the medians u, K, and r can be always correctly recovered from c with the private key SK_{id}. Then, using these medians and the output of the equations $\mathrm{sign} \cdot \mathrm{Ver}(svk, \sigma, u, id) = 1$ and $u^{x_{id}} = \mathrm{PK}_{id}^{H_2(K \| \mathrm{PK}_{id} \| id)}$, the decryption algorithm $\mathrm{Decryp}_{ex}(\mathrm{par}, c, \mathrm{SK}_{id})$ decides whether the ciphertext c encrypts 0 or 1, where x_{id} is the secret value.

1.5. Revisiting the Security Proof of IND-CPA [14]. In this paper, we revisited the IND-CPA security proof of the 1 bit 1SPO-CLE scheme. Since the security proof in [14] is long and unintelligible, we do not intend to describe the difference between their scheme and ours. Below, we will directly describe our proof ideas and proof process. IND-CPA security means that given a ciphertext, no PPT adversary could distinguish which bit has been encrypted even if the adversary has the ability to replace public key or knows the master key (i.e., type 1 adversary and type 2 adversary) in the SOA security game. We present the proof of IND-CPA security for our concrete construction (for 1SPO-CLE scheme) under the two types of attacks defined in CLE. Briefly, under type 1 attack (where the adversary is granted the ability to change the public key but does not know the master key), we reduce the IND-CPA security to the assumption of one-time signature, where the reduction (the adversary that breaks one-time signature) performs the simulation itself except that the signature part is constructed by querying its signing oracle. However, unfortunately, under type 2 attack (where the adversary knows the master key but cannot change the public key), when we try to complete the reduction from the IND-CPA security to the CDH assumption, some obstacles arise. Namely, in the construction of challenge ciphertext, since the value r, as the exponent part of the challenge g^r, is unknown to the CDH adversary, it results in that the $c_1 = rQ_{id^*}P + rP_{\text{pub}}$ part cannot be computed. Luckily, we find a way to solve this problem. Specifically, we do this by allowing the reduction algorithm (the CDH adversary) to query its CDH challenger to obtain c_1. Of course, to do this, we assume that computing r from rP is not easier than computing r from g^r. In fact, this can be done over the elliptic curve groups.

1.6. Other Related Work. We note that in the past few years, there emerged many remarkable SOA secure systems in PKE setting such as the schemes proposed by Bellare et al. [2], Fehr et al. [5], and Huang et al. [6]. Recently, SOA secure IBE also made rapid progress. In 2011, Bellare et al. [3] proposed two SO-CPA secure IBEs. In 2014, Lai et al. [10] proposed SO-CCA secure IBE using cross authentication codes. In 2016, Wang et al. proposed an SO-CPA secure CLE scheme [14] which avoids the problem of certificate management in PKE settings and key escrow in IBE settings. However, the security proof in [14] is complex and ambiguous.

1.7. Our Contribution. Our SO-CPA secure certificateless encryption scheme (CLE) is constructed based on the technique of one-sided public openability (1SPO) and one-bit CPA secure CLE. Specifically, by combining the techniques of 1SPO and one-bit CLE, we construct an IND-CPA secure 1SPO-CLE scheme. 1SPO means that given a system parameter par, public key PK_{id}, and a ciphertext c encrypting message 0 under randomness r, it enables to open the ciphertext c to another message and randomness pair $(1, r')$. This method is very challenging since the opening process does not need any secret key to participate in. Interestingly, by revisiting, we found that this method can provide us concise security proof in order to obtain the desired security. In particular, this design implies that 1 bit 1SPO-CLE with IND-CPA security implies multi-bit CLE with the same security. In more detail, the scheme is outlined as follows. If the encrypted message is 1, then its ciphertext preserves a certain structure and can be detected with some secret information. On the contrary, if the encrypted message is 0, its ciphertext takes on a random status and thus is not checkable due to its unstructured property. These properties described above are just what we need for revisiting the CLE with SO-CPA security in [14].

2. Preliminary

In the following, we give several assumptions used in this paper.

(i) $\text{sign.Skg}(1^\lambda)$: taking a security parameter 1^λ as input, this algorithm outputs a signature/verification key pair (ssk, svk).

$\text{sign.Sig}(\text{ssk}, m)$: on input signature key ssk and a message $m \in \mathcal{M}$, this algorithm outputs a signature σ.

$\text{sign.Ver}(\text{svk}, (m, \sigma))$: on input a verification key svk, a signature σ and a message m, this algorithm outputs 1, if σ is valid, and 0 otherwise.

Definition 1 (discrete logarithm assumption (DL)). Assume that G is a multiplicative group with prime order q and $g \in G$ is a generator. Given g, $y = g^a$, computing is difficult, where $a \longleftarrow_{\$} \{0, \ldots, q-1\}$. Formally, for all probabilistic polynomial time (short for PPT) adversary $\mathcal{A}$, there exists a negligible function negl such that $\text{Adv}^{\text{DL}}_{G,A}(\lambda) = \Pr[A(g, y) \longrightarrow a | g \in G, y = g^a, a \longleftarrow_{\$} \{0, \ldots, q-1\}] \leq \text{negl}(\lambda)$, where negl (λ) is a negligible function in the security parameter λ.

Definition 2 (computational Diffie–Hellman assumption). Assume that G is a cyclic group with prime order q and $g \in G$ is a generator. Given g, g^a, g^b, *computing* g^{ab} *is difficult, where* $a, b \longleftarrow_{\$} \{0, \ldots, q-1\}$. Formally, for all PPT adversary $\mathcal{A}$, there exists a negligible function negl such that: $\text{Adv}^{\text{CDH}}_{G,A}(\lambda) = \Pr[A(g, g^a, g^b) \longrightarrow g^{ab} | a, \quad b \longleftarrow_{\$} \{0, \ldots, q-1\}] \leq \text{negl}(\lambda)$.

Definition 3 (one-time signature). Let $\mathcal{M}$ be message space, $\mathcal{R}$ be randomness space, and $\mathcal{S}$ be the signature space. A signature scheme sign = (sign.Skg, sign.Sig, sign.Ver) consists of three (probabilistic) polynomial time algorithms:

We say that a message/signature pair (m, σ) is valid if for all λ, all $(\text{ssk}, \text{svk}) \longleftarrow \text{sign.Skg}(1^\lambda)$, all $m \in \mathcal{M}$, and all $\sigma \longleftarrow \text{sign.Sig}(\text{ssk}, m)$, the equation $\text{sign.Ver}(\text{svk}, (m, \sigma)) = 1$ holds.

We say that a signature scheme sign = (sign.Skg, sign.Sig, \\sign.Ver) is one-time unforgeable under chosen-message attack if for any PPT adversary $\mathcal{A}$,

the success probability of $\mathcal{A}$ in the following experiment (see Figure 1) is negligible.

2.1. Detailed Legend for Figure 1. This figure describes one-time unforgeability experiment for one-time signature denoted in Section 2, where an adversary and a challenger participate in the experiment and interact with each other. Specifically, in this experiment, the challenger first invokes the algorithm $(\text{ssk}, \text{svk}) \longleftarrow \text{sign.Skg}(1^\lambda)$ to generate a pair of signature key and verification key (ssk, svk). The signature key ssk is used to sign a message and the verification key svk is used to verify whether a given signature is valid. Given a verification key, the adversary outputs a message/signature forge pair (m^*, σ^*) with multiple times of signature queries to oracle $\mathcal{O}_{\text{ssk}}^{\text{sign.Sig}}(\cdot)$. When the message/signature forge does not belong to the queried items to oracle $\mathcal{O}_{\text{ssk}}^{\text{sign.Sig}}(\cdot)$ and the forge can verify, the experiment outputs 1 which denotes that the adversary wins the experiment. Particularly, the oracle $\mathcal{O}_{\text{ssk}}^{\text{sign.Sig}}(\cdot)$ means that when an adversary delivers a message m, the oracle returns a signature σ.

In the above experiment, we allow the adversary to query $\mathcal{O}_{\text{ssk}}^{\text{sign.Sig}}(\cdot)$ oracle only one time. Assume that the adversary output a message/signature pair satisfying $(m^*, \sigma^*) \neq (m, \sigma)$ and $\text{sign.Ver}(\text{svk}, (m^*, \sigma^*)) = 1$. Then, we say that the adversary gives a successful forge. Formally, the scheme sign is unforgeable, if there exists a negligible function negl such that

$$\text{Adv}_{\text{sign},\mathcal{A}}^{\text{otUF}}(\lambda) = \Pr\left[\text{Exp}_{\text{sign},\mathcal{A}}^{\text{otUF}}(\lambda) = 1\right]. \quad (1)$$

Definition 4 (efficiently sampleable and invertible domain 3). Here, we define two PPT randomized algorithms that are sampleable and invertible, respectively:

(i) (efficient sampling) We say that a domain D is efficiently sampleable if there exists a PPT algorithm Sample s.t. $x \longleftarrow \text{Sample}(D; R)$ is uniformly distributed over D for randomness $R \longleftarrow R_{\text{Sample}}$, where R_{Sample} is randomness space.

(ii) (efficient invertible sampling) We say that a domain D is efficiently invertible sampleable, if there exists a PPT invertible algorithm Sample^{-1} s.t. $\text{Sample}^{-1}(D, x)$ outputs R uniformly distributed over R_{Sample} for $\text{Sample}(D; R) = x$ and any $x \in D$.

Note that the Sample algorithm has sampling failure probability ζ if the sampling algorithm Sample outputs $\perp$ with probability at most ζ and invertible sampling failure probability θ if the invertible algorithm Sample^{-1} outputs $\perp$ with probability at most θ.

Definition 5 (one-sided public openability (1SPO)). A scheme has the 1SPO property if for a ciphertext $C = (c_0, c_1)$ which is the encryption result of 0 under identity ID and public key PK, where c_0 and c_1 are randomly distributed over an efficiently sampleable and invertible domain G w.r.t. algorithms Sample and Sample^{-1}, there exists an algorithm P Open To Zero $(\text{PK}, \text{ID}, C = (c_0, c_1))$ that can use the algorithm Sample^{-1} to open (c_0, c_1). Namely, $(R_0, R_1) \longleftarrow P\,\text{Open}(\text{PK}, \text{ID}, (c_0, c_1))$ with $R_0 \longleftarrow \text{Sample}^{-1}(G, c_0)$ and $R_1 \longleftarrow \text{Sample}^{-1}(G, c_1)$.

$\text{Exp}_{\text{sign},\mathcal{A}}^{\text{otUF}}(\lambda)$:
1. $(ssk, svk) \leftarrow \text{sign.Skg}(1^\lambda)$
2. $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{ssk}^{\text{sign.Sig}}(\cdot)}(svk)$
3. when $(m^*, \sigma^*) \neq (m, \sigma)$ and $\text{sign.Ver}(svk, (m^*, \sigma^*)) = 1$, output 1.

$\mathcal{O}_{ssk}^{\text{sign.Sig}}(m)$:
Return $(m, \sigma) \leftarrow \text{sign.Sig}(ssk, m)$

Figure 1: OT-signature.

3. Extractable 1SPO-CLE

3.1. Extractable 1SPO-CLE. An extractable certificateless encryption consists of the following algorithms:

(i) *Setup*: the algorithm $\text{Setup}_{ex}(1^\lambda)$ takes a security parameter λ as input and outputs a master key msk and a public parameter par, where par defines an identity space ν and ciphertext space Space_c.

(ii) *Partial private key generation*: the algorithm $\text{ParPrivKeyGen}_{ex}(\text{par}, \text{id}, \text{msk})$ takes a public parameter par, an identity $\text{id} \in \text{Space}_{\text{id}}$, and a master key msk as input and outputs the partial private key d_{id}.

(iii) *Secret key generation*: the algorithm $\text{SecValGen}_{ex}(\text{par}, \text{id})$ takes an identity id and the public parameters par as input and outputs the secret value x_{id}.

(iv) *Private key generation*: the algorithm $\text{PrivKeyGen}_{ex}(\text{par}, d_{\text{id}}, x_{\text{id}})$ takes the public parameter par, a user's partial private key d_{id}, and secret value x_{id} as input and outputs the private key $\text{SK}_{\text{id}} = (d_{\text{id}}, x_{\text{id}})$.

(v) *Public key generation*: the algorithm $\text{PubKeyGen}_{ex}(\text{par}, x_{\text{id}})$ takes a public parameter par and a user's secret value x_{id} as input and outputs the user's public key PK_{id}.

(vi) *Encryption*: the algorithm $\text{Encrypt}_{ex}(\text{par}, m, \text{PK}_{\text{id}})$ takes a public parameter par, a message $m \in \{0, 1\}$, and a user's public key PK_{id} and returns the ciphertext c by using the defined algorithm if $m = 1$; otherwise, it returns c by sampling randomly from the ciphertext space.

(vii) *Decryption*: the algorithm $\text{Decrypt}_{ex}(\text{par}, c, \text{SK}_{\text{id}})$ takes a public parameter par, a ciphertext c, and a private key SK_{id} as input and outputs $m \in \{0, 1\}$.

(viii) *Correctness*: the correctness follows that in [14]; here we omitted it in order to save space.

Definition 6 (see [5] (1SPO-CLE)). An extractable 1SPO-CLE is a scheme with the property of one-sided public openability in the CLE setting and is associated with a PPT

public algorithm P Open To Zero, so that for all $(\text{par}, \text{msk}) \longleftarrow \text{Setup}_{ex}(1^k)$, $c \longleftarrow \text{Encrypt}_{ex}(\text{par}, 0, \text{PK}_{\text{id}})$, $\text{PK}_{\text{id}} \longleftarrow \text{PubKeyGen}_{ex}(\text{par}, x_{\text{id}})$, $x_{\text{id}} \longleftarrow \text{SecValGen}_{ex}(\text{par}, \text{id})$ and $\text{id} \in \text{Space}_{\text{id}}$, P Open To Zero $(\text{par}, \text{PK}_{\text{id}}, c)$ distributes uniformly at random over $\text{Coins}(\text{par}, \text{PK}_{\text{id}}, c, 0)$. Here, $\text{Coins}(\text{par}, \text{PK}_{\text{id}}, c, 0)$ represent the set of random coins $\{R | c = \text{Encrypt}_{ex}(\text{par}, 0, \text{PK}_{\text{id}}; R)\}$.

As described in [14], the multi-bit 1SPO-CLE can be constructed from 1 bit 1SPO-CLE. Since the concrete construction and security overlap with that in [14], here we do not dwell on it, but, for completeness, we describe it in Appendices A and B.

4. Proposed Extractable 1SPO-CLE

4.1. Construction. In this section, we describe the 1SPO-CPA secure 1-bit CLE scheme. We mainly focus on the following algorithms:

Setup. The algorithm $\text{Setup}_{ex}(1^\lambda)$ first takes a security parameter λ as input and then runs a group generator $\text{GR}(\lambda)$ to get a group description (G_1, G_2, e, q). Here, G_1 and G_2 are both groups of prime order q, G_1 is an additive group, and G_2 is a multiplicative group. We also notice that both G_1 and G_2 are efficiently sampleable and invertible domain associated with algorithms Sample and Sample^{-1} shown in [3]. e: $G_1 \times G_1 \longrightarrow G_2$ is a non-degenerate bilinear map, and P is a non-zero generator of G_1. Let H_1: $\{0,1\}^l \longrightarrow Z_q^*$, H_2: $G_1 \times G_2 \times \{0,1\}^l \longrightarrow Z_q^*$, H_3: $G_2^2 \longrightarrow G_1$ be three hash functions. Pick $s \longleftarrow_{\$} Z_q^*$, set master key msk: $= s$, and compute $P_{\text{pub}} = sP$ and $g = e(P, P) \in G_2$. Let sign = (sign.Skg, sign.Sig, sign.Ver) be one-time signature scheme with signature space G_2. Finally, the public parameter is set as $par := \{(G_1, G_2, e, q, P), P_{\text{pub}}, g\}$.

Partial Private Key Generation. The algorithm $\text{ParPrivKeyGen}_{ex}(\text{par}, \text{id}, \text{msk})$ first takes the public parameter par, an identity $\text{id} \in \{0,1\}^l$, and the master secret key msk as input and proceeds as follows. It computes the partial private key $d_{\text{id}} = (1/(s + H_1(\text{id})))P \in G_1$. This can be done since if q is large enough, the probability that the unlikely event $s + H_1(\text{id}) = 0 (\text{mod} q)$ happens is negligible.

Secret Key Value Generation. The algorithm $\text{SecValGen}_{ex}(\text{par}, \text{id})$ first takes the public parameters par and an identity id as input and then randomly selects a value $x_{\text{id}} \longleftarrow Z_q^*$ as the secret value.

Private Key Generation. The algorithm $\text{PrivKeyGen}_{ex}(\text{par}, d_{\text{id}}, x_{\text{id}})$ first takes the public parameter par, the partial private key d_{id}, and the secret value x_{id} as input and then returns $\text{SK}_{\text{id}} = (d_{\text{id}}, x_{\text{id}})$ as the private key.

Public Key Generation. The algorithm $\text{PubKeyGen}_{ex}(\text{par}, x_{\text{id}})$ first takes the public parameter par and the secret value x_{id} as input and then computes the public key $\text{PK}_{\text{id}} = g^{x_{\text{id}}}$.

Encryption. The algorithm $\text{Encrypt}_{ex}(\text{par}, m \in \{0,1\}, \text{PK}_{\text{id}})$ first takes the public parameter par, a message $m \in \{0,1\}$, and the public key PK_{id} as input. It then encrypts m as follows:

First, check whether $(\text{PK}_{\text{id}})^q \doteq 1_{G_2}$. If not, abort; otherwise, compute $(\text{ssk}, \text{svk}) \longleftarrow \text{sign.Skg}(1^\lambda)$ and proceed as follows.
If $m = 1$, pick $K \longleftarrow_{\$} G_1$, compute $r = H_2(K, \text{PK}_{\text{id}}, \text{id})$, $c_1 = rH_1(id)P + rP_{\text{pub}}$, $\sigma = \text{sign.Sig}(\text{ssk}, g^r, \text{id}) \in G_2$, and $c_2 = K + H_3(g^r, \text{PK}_{\text{id}}^r)$.
If $m = 0$, pick $c_1 \longleftarrow_{\$} \text{Sample}_{G_1}$, $c_2 \longleftarrow_{\$} \text{Sample}_{G_1}$, and $\sigma \longleftarrow_{\$} \text{Sample}_{G_2}$.
Finally, the ciphertext is set as $c = (c_1, c_2, \sigma, \text{svk})$.

Decryption. The algorithm $\text{Decrypt}_{ex}(\text{par}, c, \text{SK}_{\text{id}})$ takes the public parameter par, a ciphertext c, and a private key SK_{id} as input. To decrypt a ciphertext $c = (c_1, c_2, \sigma, \text{svk})$, firstly compute $u = e(c_1, d_{\text{id}}) = g^r$ and $K = c_2 - H_3(u, u^{x_{\text{id}}})$ and verify whether $\text{sign.Ver}(\text{svk}, \sigma, u, \text{id}) = 1$ holds; if not, outputs $\perp$; otherwise, verify whether $u^{x_{\text{id}}} = \text{PK}_{\text{id}}^{H_2(K, \text{PK}_{\text{id}}, \text{id})}$ holds; if so, set $m = 1$; otherwise, $m = 0$.

Correctness. If $c = (c_1, c_2, \sigma, \text{svk})$ is the encryption of 1, then the equations $u = e(c_1, d_{\text{id}}) = e(rH_1(\text{id})P + rP_{\text{pub}}, (1/(s + H_1(\text{id})))P) = g^r$, $K = c_2 - H_3(u, u^{x_{\text{id}}})$, $\text{sign.Ver}(\text{svk}, \sigma, u, \text{id}) = 1$, and $u^{x_{\text{id}}} = \text{PK}_{\text{id}}^{H_2(K, \text{PK}_{\text{id}}, \text{id})}$ hold, so the decryption always recovers 1. If $c = (c_1, c_2, \sigma, \text{svk})$ is the encryption of 0, since $c_1 \longleftarrow_{\$} \text{Sample}_{G_1}$, $c_2 \longleftarrow_{\$} \text{Sample}_{G_1}$ and $\sigma \longleftarrow_{\$} \text{Sample}_{G_2}$ are sampled uniformly and randomly. So, $\Pr[e(c_1, d_{\text{id}})^{x_{\text{id}}} = \text{PK}_{\text{id}}^r = \text{PK}_{\text{id}}^{H_2(K, \text{PK}_{\text{id}}, \text{id})}] \leq (1/q(\lambda))$ (we assume that $q(\lambda)$ is large enough which, in turn, results in a negligible quantity for $(1/q(\lambda))$).

4.2. Security

Theorem 1. *Assume the hash functions H_1, H_2, and H_3 are random oracles, and the scheme sign = (sign.Skg, sign.Sig, sign.Ver) is one-time signature scheme. Let Π' be extractable 1SPO-CLE scheme proposed in Section 4.1 and G_1 and G_2 be PR-sampleable (pseudorandom-sampleable) with negligible sampling failure probability. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be any IND-CPA type 1 and type 2 adversaries against scheme Π', respectively, and are allowed to make polynomial times of queries to H_2 and H_3; then, the scheme Π' is IND-CPA secure under both type 1 adversary and type 2 adversary.*

Proof. We first prove that, for type 1 adversary, the security can be reduced to the security of one-time signature scheme *sign* and then prove that, for type 2 adversary, the security can be reduced to the computational Diffie–Hellman assumption (short for CDH). In the following, we describe the reduction between the adversary $\mathcal{A}_{sig}$ (which tries to break the one-time signature scheme) and the type 1 adversary $\mathcal{A}_1$ and the reduction between the adversary $\mathcal{A}_{cdh}$ (which tries to break the CDH assumption) and the type 2 adversary $\mathcal{A}_2$, respectively. □

4.2.1. Type 1 Adversary

Setup: the adversary $\mathcal{A}_{\text{sig}}$ (which has the signing verification key svk) first generates the public parameter $par := \{(G_1, G_2, e, q, P), P_{\text{pub}}, g\}$ and the master key s, where $P_{\text{pub}} = sP$ and $g = e(P, P) \in G_2$, and then sends the public parameter par to the adversary $\mathcal{A}_1$.

Partial private key query: on receiving the identity id, if id $\notin$ ChID, where ChID is the challenge identity set, the adversary $\mathcal{A}_{\text{sig}}$ invokes the partial private key generation algorithm to obtain the partial private key d_{id} and sends it to the adversary $\mathcal{A}_1$; otherwise it aborts. Concretely, the adversary $\mathcal{A}_{\text{sig}}$ first queries the random oracle H_1 to get Q_{id} and then computes $d_{\text{id}} = (1/(s + Q_{\text{id}}))P \in G_1$. Note that the oracle H_1 here is stateful and assume that all oracles in the following are stateful.

Pr*ivate key query*: on receiving the identity id, if id $\notin$ ChID, where ChID is the challenge identity set, $\mathcal{A}_{\text{sig}}$ first invokes the secret value generation algorithm and the partial private key generation algorithm to get the secret value x_{id} and the partial private key d_{id} and then sets the private key as $\text{SK}_{\text{id}} = (d_{\text{id}}, x_{\text{id}})$, i.e., $\text{SK}_{\text{id}} = (1/(s + H_1(\text{id}))P, x_{\text{id}})$; otherwise it aborts.

Public key query: on receiving the identity id, $\mathcal{A}_{\text{sig}}$ first invokes the secret value generation algorithm to get x_{id} and then computes the public key as $\text{PK}_{\text{id}} = g^{x_{\text{id}}}$.

Re*place public key query*: on receiving the identity id, $\mathcal{A}_{\text{sig}}$ replaces the original public key $\text{PK}_{\text{id}} = g^{x_{\text{id}}}$ with the new public key $\text{PK}'_{\text{id}} = g^{x_{\text{id}}\prime}$.

Challenge: on receiving the challenge identity id^* and the challenge message $m_0 = 0$, $m_1 = 1$ and the public key PK_{id^*}, the adversary $\mathcal{A}_{\text{sig}}$ computes challenge ciphertext as follows.

First flip a coin $b \longleftarrow_{\$} \{0, 1\}$ and then check whether $(\text{PK}_{\text{id}^*})^q \doteq 1_{G_2}$; if not, abort; otherwise, proceed as follows.

If $m_b = 1$, do the following steps.

(1) First, pick $K \longleftarrow_{\$} G_1$, and then for tuple $(K, \text{PK}_{\text{id}^*}, \text{id}^*)$, query oracle H_2 to get r.
(2) For id^*, query oracle H_1 to get Q_{id^*}.
(3) Compute g^r and $\text{PK}_{\text{id}^*}^r$, and then for $(g^r, \text{PK}_{\text{id}^*}^r)$, query oracle H_3 to get h.
(4) Compute $c_1 = rQ_{\text{id}^*}P + rP_{\text{pub}}$ and $c_2 = K + h$.
(5) For (g^r, id^*), query signature oracle to get σ.

If $m_b = 0$, pick $c_1 \longleftarrow_{\$} \text{Sample}_{G_1}$, $c_2 \longleftarrow_{\$} \text{Sample}_{G_1}$ and $\sigma \longleftarrow_{\$} \text{Sample}_{G_2}$ at random. Then, the final challenge ciphertext is set as $c = (c_1, c_2, \sigma, \text{svk})$.

From above, we can see that the adversary $\mathcal{A}_{\text{sig}}$ provides perfect simulation for $\mathcal{A}_1$. Now we do the following analysis.

Analysis: let the challenge ciphertext $c = (c_1, c_2, \sigma, \text{svk})$. In the experiment, since $\mathcal{A}_1$ does not know d_{id^*}, it cannot compute the value $u = g^r$. Assume $\mathcal{A}_1$ guess $u' = g^{r'}$ randomly. Then, by the one-time signature scheme sign, the verification equation $\text{sign.Ver}(\text{svk}, \sigma, u', \text{id}^*) = 1$ does not hold with overwhelming probability.

4.2.2. Type 2 Adversary

Setup: the adversary $\mathcal{A}_{\text{cdh}}$ (which has the challenge $(\text{PK}_{\text{id}^*}, u = g^r)$) first generates the public parameter $par := \{(G_1, G_2, e, q, P), P_{\text{pub}}, g\}$ and the master key s, where $P_{\text{pub}} = sP$ and $g = e(P, P) \in G_2$, and then sends the public parameter par to the adversary $\mathcal{A}_2$.

Pr*ivate key query*: in this phase, If id $\notin$ ChID, where ChID is the challenge identity set, the adversary $\mathcal{A}_{\text{cdh}}$ first invokes the secret value generation algorithm to get secret value x_{id} and computes partial private key d_{id}, and then sets the private key as $\text{SK}_{\text{id}} = (d_{\text{id}}, x_{\text{id}})$, i.e., $\text{SK}_{\text{id}} = (1/(s + H_1(\text{id}))P, x_{\text{id}})$.

Public key query: in this phase, if id $\notin$ ChID, the adversary $\mathcal{A}_{\text{cdh}}$ first invokes the secret value generation algorithm to get secret value x_{id} and then computes the public key as $\text{PK}_{\text{id}} = g^{x_{\text{id}}}$; otherwise it aborts.

Challenge: on receiving the challenge identity id^* and the challenge message $m_0 = 0$, $m_1 = 1$ and the public key PK_{id^*}, the adversary $\mathcal{A}_{\text{cdh}}$ computes challenge ciphertext as follows. First sample a random $b \longleftarrow_{\$} \{0, 1\}$, then check whether $(\text{PK}_{\text{id}^*})^q \doteq 1_{G_2}$; if not, abort; otherwise, compute $(\text{ssk}, \text{svk}) \longleftarrow \text{sign.Skg}(1^\lambda)$ and proceed as follows.

If $m_b = 1$, do the following steps.

(1) Pick $c_2 \longleftarrow_{\$} G_1$.
(2) For id^*, query oracle H_1 to get Q_{id^*}.
(3) Query the CDH challenger to get c_1, where c_1 is computed as $c_1 = rQ_{\text{id}^*}P + rP_{\text{pub}}$.
(4) For $(u = g^r, \text{id}^*)$, compute signature σ.
(5) Set the challenge ciphertext as $c = (c_1, c_2, \sigma, \text{svk})$.

From above, it is easy to see that we implicitly set $r = H_2(K, \text{PK}_{\text{id}^*}, \text{id}^*)$ for $K = c_2 - h$ and $h = H_3(g^r, \text{PK}_{\text{id}^*}^r)$. In addition, we require here that computing r from rP is not easier than computing r from g^r.

If $m_b = 0$, pick $c_1 \longleftarrow_{\$} \text{Sample}_{G_1}$, $c_2 \longleftarrow_{\$} \text{Sample}_{G_1}$, and $\sigma \longleftarrow_{\$} \text{Sample}_{G_2}$ at random.

Then, set the challenge ciphertext as $c = (c_1, c_2, \sigma, \text{svk})$.

From above, we can see that the adversary $\mathcal{A}_{\text{cdh}}$ provides perfect simulations for the adversary $\mathcal{A}_2$. Now we do the following analysis.

Analysis: let $c = (c_1, c_2, \sigma, \text{svk})$ be the challenge ciphertext. In the experiment, $\mathcal{A}_2$ knows $u = g^r$ and PK_{id^*}; by the CDH assumption, it is still difficult to compute $u^{x_{id^*}}$ and K to make the verification equation $u^{x_{\text{id}^*}} = \text{PK}_{\text{id}^*}^{H_2(K, \text{PK}_{\text{id}^*}, \text{id}^*)}$ hold.

This completes the proof of Theorem 1.

Table 1: Comparison in exponent, pairing, and security model.

	Exponent	Pairing	Security model	Need key escrow?	Simplified proof?
Scheme LoR [3]	14	5	SM	Yes	—
Scheme BBoR [3]	15	1	SM	Yes	—
Scheme [14]	6	2	ROM	No	No
Our scheme	6	2	ROM	No	Yes

5. Comparisons and Discussion

The authors in [14] first proposed an SOA secure certificateless encryption scheme. In this paper, we improved it to make the security proof more concise and user friendly. Although in [14], they gave an efficiency analysis, here, to make it easier to understand, we give a more detailed comparison with the existing similar schemes, especially with that in [3, 14]. The detailed comparison results are shown in Table 1. Similarly, in terms of complexity, we also just make comparisons among them on the cost of the additive and multiplicative operations, especially on the exponent and the pairing operations. In addition, we also compare them in "security model," "whether key escrow is needed," and "whether a simplified proof is provided." From this table, we can see that in [3], the first scheme requires 14 exponents and 5 pairings and the second scheme requires 15 exponents and 1 pairing, while in [14], the scheme only needs 6 exponents and 2 pairings. By comparison, we can see that our scheme not only realizes a simplified security proof but also obtains the same efficiency and security level as that of [14].

6. Result

As shown in Table 1, compared with the schemes in [3], our scheme is practical in real applications which is mainly reflected in the following 4 aspects: (1) our scheme can be instantiated from very standard assumption such as computational Diffie–Hellman; (2) the used one-time signature can be constructed from standard assumption such as one-way function; (3) the hash functions such as random oracles in our scheme are very easily run on a low-configured device; (4) our scheme has better efficiency as analyzed in Section 5. Specifically, our scheme has 8 exponents and 3 pairings less than that of the first scheme in [3] and has 1 pairing more than that of the second scheme in [3], respectively. In addition, compared with [14], our scheme has more concise and user-friendly security proof.

7. Conclusions

This paper proposed a certificateless public key encryption against selective opening attacks (SOA), which is suitable for the data storage in Cloud 5G environment. This scheme is proved secure in the ROM under the assumptions of CDH and security of one-time signature. The advantage of the scheme is that it eliminates both certificate management and key management in PKI-based PKE and IBE settings and is practical in Cloud 5G settings. Compared with [14], our scheme not only has more concise and user-friendly security proof but also achieves the same level of security, which strengthens the data security storage in Cloud 5G applications.

Appendix

A. How to Construct l-Bit 1SPO-CLE from 1-Bit 1SPO-CLE

Let $II = (\text{Setup}_{ex}, \text{ParPrivKeyGen}_{ex}, \text{SecValGen}_{ex}, \text{PrivKeyGen}_{ex}, \text{PubKeyGen}_{ex}, \text{Encrypt}_{ex}, \text{Decrypt}_{ex})$ be a 1 bit 1SPO-CLE scheme. An l-bit CLE scheme $II^l = (\text{Setup}^l_{ex}, \text{ParPrivKeyGen}^l_{ex}, \text{SecValGen}^l_{ex}, \text{PrivKeyGen}^l_{ex}, \text{PubKeyGen}^l_{ex}, \text{Encrypt}^l_{ex}, \text{Decrypt}^l_{ex})$ with message space $\{0,1\}^l$ is constructed as follows:

$$\begin{aligned} \text{Setup}^l_{ex} &= \text{Setup}_{ex}, \\ \text{ParPrivKeyGen}^l_{ex} &= \text{ParPrivKeyGen}_{ex}, \\ \text{SecValGen}^l_{ex} &= \text{SecValGen}_{ex}, \\ \text{PrivKeyGen}^l_{ex} &= \text{PrivKeyGen}_{ex}, \\ \text{PubKeyGen}^l_{ex} &= \text{PubKeyGen}_{ex}, \end{aligned} \tag{A.1}$$

where $c = c[1] \| \cdots \| c[l] \longleftarrow \text{Encrypt}^l_{ex}(\text{par}, \text{PK}_{\text{id}}, M \in \{0,1\}^l)$ such that $c[i] \longleftarrow \text{Encrypt}_{ex}(\text{par}, M[i], \text{PK}_{\text{id}})$ and $M[i]$ is the i-th bit of M.

$\text{Decrypt}^l_{ex}(c)$: decrypt component $c[i]$ for each $i \in [l]$ and every message bit $M[i]$, then return $M = M[1] \cdot M[l]$.

The security is shown in Appendix B.

B. Security

In the security definition, there are two types of adversaries: type 1 adversary $\mathcal{A}_1$ and type 2 adversary $\mathcal{A}_2$. Type 1 adversary is a malicious user, who can replace the user's public key but cannot know the master key. Type 2 adversary is a malicious KGC, who can know the master key but cannot replace the user's public key.

In Figure 2 (resp. Figure 3), IND-CPA1 game is for Type 1 adversary $\mathcal{A}_1$ in CLE (resp. IND-CPA2 is for Type 2 adversary $\mathcal{A}_2$). We have $\text{Adv}^{\text{IND-CPA-1}}_{II}(\mathcal{A}_1) = 2 \cdot \Pr[\text{INDCPA } 1^{\mathcal{A}_1}_{II} \Rightarrow \text{true}] - 1$ (resp. $\text{Adv}^{\text{IND-CPA-2}}_{II}(\mathcal{A}_2) = 2 \cdot \Pr[\text{INDCPA } 2^{\mathcal{A}_2}_{II} \Rightarrow \text{true}] - 1$). We say that II is IND-CPA-1

(resp. IND-CPA-2) secure if $\mathrm{Adv}_{\Pi}^{\mathrm{IND-CPA-1}}(\mathcal{A}_1)$ (resp. $\mathrm{Adv}_{\Pi}^{\mathrm{IND-CPA-2}}(\mathcal{A}_2)$) is negligible for all PPT $\mathcal{A}_1$ (resp. $\mathcal{A}_2$).

B.1. Detailed Legend for Figure 2. This figure describes indistinguishable chosen-message attack1 experiment for certificateless encryption scheme, where an adversary and a challenger participate in the experiment and interact with each other. Specifically, in this experiment, the challenger first invokes the algorithm $(\mathrm{par}, \mathrm{msk}) \longleftarrow \mathrm{Setup}_{ex}(1^k)$ to generate (par, msk), where par is taken as the common input and msk is used to generate private key and partial private key. The partial private key oracle proc.ParPrivKeyGen(id) invokes the partial key generation algorithm $d_{\mathrm{id}} \longleftarrow \mathrm{ParPrivKeyGen}_{ex}(\mathrm{par}, \mathrm{id}, \mathrm{msk})$ to return a partial private key d_{id}. The secret value oracle proc.SecValGen(id) invokes the secret value generation algorithm $x_{\mathrm{id}} \longleftarrow \mathrm{SecValGen}_{ex}(\mathrm{par}, \mathrm{id})$ to return a secret value x_{id}. The private key oracle proc.PrivKeyGen(id) invokes the private key generation algorithm $\mathrm{PrivKeyGen}_{ex}(\mathrm{par}, d_{\mathrm{id}}, x_{\mathrm{id}})$ to return a private key. The oracle proc.PubKeyGen(id) invokes the public key generation algorithm $\mathrm{PubKeyGen}_{ex}(\mathrm{par}, x_{\mathrm{id}})$ which takes as input a public parameter and a secret value and returns a public key $\mathrm{PK}_{\mathrm{id}}$ for user id. The replace public key oracle proc.RePubKey $(\mathrm{PK}_{\mathrm{id}}', \mathrm{PK}_{\mathrm{id}}, \mathrm{id})$ takes as input a fresh public key $\mathrm{PK}_{\mathrm{id}}'$, the original public key $\mathrm{PK}_{\mathrm{id}}$, and an identity is and finally returns the replace public key $\mathrm{PK}_{\mathrm{id}}'$. The challenge oracle proc.LR$(m_0, m_1, \mathrm{PK}_{\mathrm{id}}, \mathrm{id})$ takes as input two messages (m_0, m_1), $\mathrm{PK}_{\mathrm{id}}$, and id and returns a challenge ciphertext c which encrypts challenge message m_0 or m_1 randomly. Finally, the experiment gives an output $b = b'$, which denotes whether the adversary wins or not in the experiment.

B.2. Detailed Legend for Figure 3. This figure describes indistinguishable chosen-message attack2 experiment for certificateless encryption scheme, where an adversary and a challenger participate in the experiment and interact with each other. Specifically, in this experiment, the challenger first invokes the algorithm $(\mathrm{par}, \mathrm{msk}) \longleftarrow \mathrm{Setup}_{ex}(1^k)$ to generate (par, msk), where the public parameter par is taken as a common input in all the other algorithms and msk is used to generate private key and partial private key. The oracle proc.SecValGen(id) invokes to return a secret value x_{id}. The private key oracle proc.PrivKeyGen(id) invokes the private key generation algorithm $\mathrm{PrivKeyGen}_{ex}(\mathrm{par}, d_{\mathrm{id}}, x_{\mathrm{id}})$ to return a private key. The public key oracle proc.PubKeyGen(id) invokes the public key generation algorithm $\mathrm{PubKeyGen}_{ex}(\mathrm{par}, x_{\mathrm{id}})$ to return a public key $\mathrm{PK}_{\mathrm{id}}$. The challenge oracle proc.LR$(m_0, m_1, \mathrm{PK}_{\mathrm{id}}, \mathrm{id})$ takes as input two messages (m_0, m_1) chosen by the adversary, $\mathrm{PK}_{\mathrm{id}}$, and id and returns a challenge c which encrypts challenge message m_0 or m_1 randomly. Finally, the experiment outputs $b = b'$, which denotes whether the adversary wins or not in the experiment.

Figures 4–6 are presented for the SO-CPA security for the scheme Π^l where we define two types of adversaries. Both $\mathcal{M}$ $(\alpha \in \{0,1\}^*)$ and $\mathcal{R}$ denote a randomized algorithm. $\mathcal{A}_1$ and $\mathcal{A}_2$ denote type 1 and type 2 SOA adversaries, respectively. In particular, both of the two type of adversaries are only allowed to make one time of query to NewMg before making the Corrupt query. The simulator $\mathcal{S}$ in Figure 6 is an SOA-simulator and is only required to make one time of query to the oracles NewMg and Corrupt.

We say that a CLE scheme Π^l is SIM-SO-CPA secure if for every PPT $\mathcal{M}$, $\mathcal{R}$, $\mathcal{A}_1$, and adversary $\mathcal{A}_2$, there exists a PPT simulator $\mathcal{S}$ such that $\mathrm{Adv}_{\Pi^l,n,\mathcal{S},\mathcal{M},\mathcal{R}}^{\mathrm{SO-CPA-1}}(\mathcal{A}_1) = \Pr[\mathrm{Game}_{\Pi^l,n,\mathcal{A}_1,\mathcal{M},\mathcal{R}}^{\mathrm{SO-CPA-REAL\,1}} \Rightarrow 1] - \Pr[\mathrm{Game}_{\Pi^l,n,\mathcal{S},\mathcal{M},\mathcal{R}}^{\mathrm{SO-CPA-IDEAL}} \Rightarrow 1] \leq \mathrm{negl}(\lambda)$. $\mathrm{Adv}_{\Pi^l,n,\mathcal{S},\mathcal{M},\mathcal{R}}^{\mathrm{SO-CPA-2}}(\mathcal{A}_2) = \Pr[\mathrm{Game}_{\Pi^l,n,\mathcal{A}_2,\mathcal{M},\mathcal{R}}^{\mathrm{SO-CPA-REAL\,2}} \Rightarrow 1] - \Pr[\mathrm{Game}_{\Pi^l,n,\mathcal{S},\mathcal{M},\mathcal{R}}^{\mathrm{SO-CPA-IDEAL}} \Rightarrow 1] \leq \mathrm{negl}(\lambda)$.

B.3. Detailed Legend for Figure 4. This figure describes selective opening chosen-message attack1 real experiment for certificateless encryption scheme, where an adversary and a challenger participate in the experiment and interact with each other. Specifically, in this experiment, the challenger first invokes $(\mathrm{par}, \mathrm{msk}) \longleftarrow \mathrm{Setup}_{ex}(1^k)$ to generate (par, msk), where the value par is taken as a common input and the value msk is used to generate private key and partial private key. The partial private key oracle proc.ParPrivKeyGen(id) invokes the algorithm $d_{\mathrm{id}} \longleftarrow \mathrm{ParPrivKeyGen}_{ex}(\mathrm{par}, \mathrm{id}, \mathrm{msk})$ to produce a partial private key d_{id}. The secret value oracle proc.SecValGen(id) invokes the secret value generation algorithm $x_{\mathrm{id}} \longleftarrow \mathrm{SecValGen}_{ex}(\mathrm{par}, \mathrm{id})$ to generate a secret value x_{id} associated with id. The oracle proc.PrivKeyGen(id) invokes the private key generation algorithm $\mathrm{PrivKeyGen}_{ex} \backslash\backslash (\mathrm{par}, d_{\mathrm{id}}, x_{\mathrm{id}})$ to generate a private key. The public key oracle proc.PubKeyGen(id) invokes the public key generation algorithm $\mathrm{PubKeyGen}_{ex}(\mathrm{par}, x_{\mathrm{id}})$ to generate a public key PK_{id}. The replace oracle proc.RePubKey $(\mathrm{PK}_{\mathrm{id}}', \mathrm{PK}_{\mathrm{id}}, \mathrm{id})$ replaces an old public key with a freshly replaced $\mathrm{PK}_{\mathrm{id}}'$. The challenge oracle proc.NewMg$(\mathbf{id}, \mathbf{PK}, \alpha)$ first takes as input $\mathbf{id}$, PK, and α, and then checks whether $\mathbf{id}$ has been queried to the private key oracle or the replace public key oracle; if not, the challenger samples a message m according to distribution $\mathcal{M}$ determined by α. Then, it samples a randomness $r[i]$ and computes a challenge ciphertext c for message m. The corrupt oracle proc.Corrupt(I) on input a corrupt set 13 I chosen by the adversary and returns the opening $\mathbf{m}[I], \mathbf{r}[I]$. Finally, the experiment outputs $b = b'$, which denotes whether the adversary wins or not in the experiment.

B.4. Detailed Legend for Figure 5. This figure describes selective opening chosen-message attack1 real experiment for certificateless encryption scheme, where an adversary and a challenger participate in the experiment and interact with each other. Specifically, in this experiment, the challenger first invokes the algorithm $(\mathrm{par}, \mathrm{msk}) \longleftarrow \mathrm{Setup}_{ex}(1^k)$ to sample (par, msk). The oracle proc.SecValGen(id) invokes the algorithm $x_{\mathrm{id}} \longleftarrow \mathrm{SecValGen}_{ex}(\mathrm{par}, \mathrm{id})$ to return a secret value x_{id} for user id. The private key oracle proc.PrivKeyGen(id) invokes the private key generation

proc.Initialization(k)
$(par, msk) \leftarrow \text{Setup}_{ex}(1^k)$;
$b \leftarrow_{\$} \{0,1\}$;
return par
proc.ParPrivKeyGen(id)
If $id \in ChID$ then return $\perp$;
$PpID \leftarrow PpID \cup \{id\}$;
return $\text{ParPrivKeyGen}_{ex}(par, id, msk)$
proc.SecValGen(id)
return $\text{SecValGen}_{ex}(par, id)$
proc.ParPrivKeyGen(id)
If $id \in ChID$ then return $\perp\!\!\perp$;
$SkID \leftarrow SkID \cup \{id\}$;
$d_{id} \leftarrow \text{ParPrivKeyGen}_{ex}(par, id, msk)$;
$x_{id} \leftarrow \text{SecValGen}_{ex}(par, id)$;
return $\text{PrivKeyGen}_{ex}(par, d_{id}, x_{id})$
proc.PubKeyGen(id)
$x_{id} \leftarrow \text{SecValGen}_{ex}(par, id)$;
return $\text{PubKeyGen}_{ex}(par, x_{id})$
proc.RePubKey(PK'_{id}, PK_{id}, id)
$PK_{id} \leftarrow PK'_{id}$;
$RpID \leftarrow RpID \cup \{id\}$;
return PK_{id}
proc.LR(m_0, m_1, PK_{id}, id)
If $id \in SkID$
then return $\perp$;
$ChID \leftarrow ChID \cup \{id\}$;
c= $\text{Encrypt}_{ex}(par, m_b, PK_{id})$;
return c
proc.Finalize(b')
return $(b = b')$

Figure 2: Game INDCPA 1.

proc.Initialization(λ)
$(par, msk) \leftarrow \text{Setup}_{ex}(1^\lambda)$;
$b \leftarrow_{\$} \{0,1\}$;
return (par, msk)
proc.SecValGen(id)
If $id \in ChID$ then return $\perp$;
$SeID \leftarrow SeID \cup id$
return $\text{SecValGen}_{ex}(par, id)$
proc.ParvKeyGen(id)
If $id \in ChID$ then return $\perp$;
$SkID \leftarrow SkID \cup id$;
$d_{id} \leftarrow \text{ParPrivKeyGen}_{ex}(par, id, msk)$;
$x_{id} \leftarrow \text{SecValGen}_{ex}(par, id)$;
return $\text{PrivKeyGen}_{ex}(par, d_{id}, x_{id})$
proc.PubKeyGen(id)
$x_{id} \leftarrow \text{SecValGen}_{ex}(par, id)$;
return $\text{PubKeyGen}_{ex}(par, x_{id})$
proc.LR(m_0, m_1, PK_{id}, id)
If $id \in SeID \quad id \in SkID$
then return $\perp$;
$ChID \leftarrow ChID \cup id$;
$c \leftarrow \text{Encrypt}_{ex}(par, m_b, PK_{id})$;
return c
proc.Finalize(b')
return $(b = b')$

Figure 3: Game INDCPA 2.

proc.Initialization(λ)
$(par, msk) \leftarrow \text{Setup}_{ex}(1^\lambda)$;
return par
proc.ParPrivKeyGen(id)
If $id \in ChID$ then return $\perp$;
$PpID \leftarrow PpID \cup \{id\}$
return $\text{ParPrivKeyGen}_{ex}(par, id, msk)$
proc.SecValGen(id)
return $\text{SecValGen}_{ex}(par, id)$
proc.PrivKeyGen(id)
If $id \in ChID$ then return $\perp$;
$SkID \leftarrow SkID \cup \{id\}$;
$d_{id} \leftarrow \text{ParPrivKeyGen}_{ex}(par, id, msk)$;
$x_{id} \leftarrow \text{SecValGen}_{ex}(par, id)$;
return $\text{PrivKeyGen}_{ex}(par, d_{id}, x_{id})$
proc.PubKeyGen(id)
$x_{id} \leftarrow \text{SecValGen}_{ex}(par, id)$;
return $\text{PubKeyGen}_{ex}(par, x_{id})$
proc.RePubKey(PK'_{id}, PK_{id})
$PK_{id} \leftarrow PK'_{id}$;
$RpID \leftarrow RpID \cup \{id\}$;
return PK_{id}
proc.NewMg($\mathbf{id}, \mathbf{PK}, \alpha$)
If $\mathbf{id} \cap SkID \neq \phi$ or $\mathbf{id} \cap RpID = \phi$
then return $\perp$;
$ChID \leftarrow ChID \cup \mathbf{id}$;
$\mathbf{m} \leftarrow_{\$} \mathcal{M}(\alpha)$;
For i in 1 to n
$\mathbf{r}[i] \leftarrow_{\$} Coins(par, \mathbf{m}[i])$
$\mathbf{c}[i] \leftarrow \text{Encrypt}_{ex}^{I}(par, \mathbf{m}[i], \mathbf{PK}_{\mathbf{id}[i]}; \mathbf{r}[i])$;
return $\mathbf{c}$
proc.Corrupt(I)
return $\mathbf{m}[I], \mathbf{r}[I]$
proc.Finalize(out)
return $\mathcal{R}(\mathbf{m}, ChID, I, out)$

Figure 4: $\text{Game}^{\text{SO-CPA-REAL 1}}_{\Pi^I, n, \mathcal{M}, \mathcal{R}}$.

algorithm $\text{PrivKeyGen}_{ex}\backslash\backslash(\text{par}, d_{\text{id}}, x_{\text{id}})$ to generate a private key. The oracle proc.PubKeyGen (id) invokes the public key generation algorithm PubKeyGen_{ex} (par, x_{id}) to return a public key PK_{id}. The challenge oracle proc.NewMg (**i d**, **PK**, α) first checks whether the identity **i d** is legal; if not, the challenger samples a message m according to

proc. Intialization (λ)
$(par, msk) \leftarrow \text{Setup}_{ex}(1)$;
return (par, msk)
proc.SecValGen (id)
If $ID \cap ChID \neq \phi$ then return $\perp$;
$SeID \leftarrow SeID \cup id$
return $\text{SecValGen}_{ex}(par, id)$
proc.PrivKeyGen (id)
If $id \cap ChID \neq \phi$ then return $\perp$;
$SkID \leftarrow SkID \cup id$;
$d_{id} \leftarrow \text{ParPrivKeyGen}_{ex}(par, id, msk)$;
$x_{id} \leftarrow \text{SecValGen}_{ex}(par, id)$;
return $\text{PrivKeyGen}_{ex}(par, d_{id}, x_{id})$
proc.PubKeyGenn(par, id)
$x_{id} \leftarrow \text{SecValGen}_{ex}(par, id)$;
return $\text{PubKeyGen}_{ex}(par, x_{id})$
proc.NewMg($\mathbf{id}, \mathbf{PK}, \alpha$)
If $\mathbf{id} \cap SeID \neq \phi$ $\mathbf{id} \cap SkID \neq \phi$
then return $\perp$;
$ChID \leftarrow ChID \cup \mathbf{id}$;
$\mathbf{m} \leftarrow_{\$} \mathcal{M}(\alpha)$;
For i in 1 to n
$\mathbf{r}[i] \leftarrow_{\$} Coins(par, \mathbf{m}[i])$;
$\mathbf{c}[i] \leftarrow \text{Encrypt}_{ex}^{l}(par, \mathbf{m}[i], \mathbf{PK}_{\mathbf{id}[i]}; \mathbf{r}[i])$;
return $\mathbf{c}$
proc.Corrupt(I)
return $\mathbf{m}[I], \mathbf{r}[I]$
proc.Finalize (out)
return $\mathcal{R}(\mathbf{m}, ChID, I, out)$

Figure 5: $\text{Game}_{II^l,n,\mathcal{M},\mathcal{R}}^{\text{SO-CPA-REAL 2}}$.

proc.Initialization
return $\perp$
proc.NewMg(id,**PK**,α)
ChID $\leftarrow$ ChID $\cup$ id;
$\mathbf{m} \leftarrow_{\$} \mathcal{M}(\alpha)$;
return $\perp$
proc.Corrupt(I)
return $\mathbf{m}[I]$
proc.Finalize(out)
return $\mathcal{R}(\mathbf{m}, ChID, I, out)$

Figure 6: $\text{Game}_{II^l,n,\mathcal{M},\mathcal{R}}^{\text{SO-CPA-IDEAL}}$.

distribution $\mathcal{M}$ determined by α. Then, it samples a randomness $r[i]$ and invokes encryption algorithm to generate a challenge ciphertext c for message m. The corrupt oracle proc.Corrupt(I) takes as input a corrupt set I (which is chosen by the adversary), and returns the opening messages $\mathbf{m}[I]$ and randomnesses $\mathbf{r}[I]$. Finally, the experiment outputs $b = b'$, which denotes whether the adversary wins or not in the experiment.

B.5. Detailed Legend for Figure 6. This figure describes selective opening chosen-message attack ideal experiment for certificateless encryption scheme, where an adversary and a simulator participate in the experiment and interact with each other. Specifically, in this experiment, during the initialization phase, the challenger returns nothing for an adversary, while the challenge oracle proc.NewMg $(\mathbf{id}, \mathbf{PK}, \alpha)$ only samples messages m according to distribution $\mathcal{M}$ determined by α but returns nothing to the adversary. In the corruption phase, the challenger opens the partial messages $\mathbf{m}[I]$ according to the set I chosen by the adversary. Finally, the experiment returns an output of a relation with respect to an input tuple $(\mathbf{m}, \text{ChID}, I, \text{out})$.

C. Conversion from 1SPO to SIM-SO-CPA

Here, we use a theorem (i.e., Theorem 2) to demonstrate how to reduce the SIM-SO-CPA security to 1SPO security.

Theorem 2 (see [14]). *Let II be a 1-bit 1SPO-CLE scheme with a δ one-sided opener P Open To Zero algorithm [3] and II^l the l-bit 1SPO-CLE scheme from II. $\mathcal{A}_1$ and $\mathcal{A}_2$ are type 1 adversary and type 2 adversary against SO-CPA security of II^l, respectively. Let $\mathcal{R}$ be a PPT relation and $\mathcal{M}$ be a PPT message sampler. Then, there exist $\mathcal{S}$ and two $\mathcal{B}_1$ and $\mathcal{B}_2$ such that*

$$\text{Adv}_{II^l,n,\mathcal{M},\mathcal{R},\mathcal{S}}^{\text{SO-CPA-1}}(\mathcal{A}_1) \leq \text{nl} \cdot \text{Adv}_{II}^{\text{IND-CPA-1}}(\mathcal{B}_1) + \text{nl}\,\delta,$$
$$\text{Adv}_{II^l,n,\mathcal{M},\mathcal{R},\mathcal{S}}^{\text{SO-CPA-2}}(\mathcal{A}_2) \leq \text{nl} \cdot \text{Adv}_{II}^{\text{IND-CPA-2}}(\mathcal{B}_2) + \text{nl}\,\delta. \tag{C.1}$$

Proof. This proof process is exactly the same as that of Theorem 1 in [14], so we will not repeat it here in order to save space. □

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The first author was supported by the National Key Research and Development Program of China (grant no. 2017 YFB0802000), the National Natural Science Foundation of China (grant no. NSFC61702007), and other foundations (grant nos. 2019M661360 (KLH2301024), gxbjZD27, KJ2018A0533, XWWD201801, and ahnis20178002). The third author was supported by the National Key Research and Development Program of China (grant no.

2017YFB0802000) and the National Natural Science Foundation of China (grant no. U1705264).

References

[1] J. Baek, R. Safavi-Naini, and W. Susilo, "Certificateless public key encryption without pairing," in *Lecture Notes in Computer Science*, pp. 134–148, Springer, Berlin, Germany, 2005.

[2] M. Bellare, D. Hofheinz, and S. Yilek, "Possibility and impossibility results for encryption and commitment secure under selective opening," in *Advances in Cryptology-EUROCRYPT 2009*, pp. 1–35, Springer, Berlin, Germany, 2009.

[3] M. Bellare, B. Waters, and S. Yilek, "Identity-based encryption secure against selective opening attack," in *Theory of Cryptography*, pp. 235–252, Springer, Berlin, Germany, 2011.

[4] C. Dwork, M. Naor, O. Reingold, and L. Stockmeyer, "Magic functions," in *Foundations of Computer Science (FOCS 1999)*, pp. 523–534, Springer, Berlin, Germany, 1999.

[5] S. Fehr, D. Hofheinz, E. Kiltz, and H. Wee, "Encryption schemes secure against chosen-ciphertext selective opening attacks," in *Advances in Cryptology-EUROCRYPT 2010*, pp. 381–402, Springer, Berlin, Germany, 2010.

[6] Z. Huang, S. Liu, and B. Qin, "Sender-equivocable encryption schemes secure against chosen-ciphertext attacks revisited," in *Public-Key Cryptography-PKC 2013*, pp. 369–385, Springer, Berlin, Germany, 2013.

[7] D. Jia, Y. Liu, and B. Li, "IBE with tight security against selective opening and chosen-ciphertext attacks," *Designs, Codes and Cryptography*, vol. 88, no. 7, pp. 1371–1400, 2020.

[8] J. T. Ning and G. S. Poh, "Update recovery attacks on encrypted database within two updates using range queries leakage," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[9] J. Lai, R. H. Deng, S. Liu, and W. Kou, "Rsa-based certificateless public key encryption," in *Information Security Practice and Experience*, pp. 24–34, Springer, Berlin, Germany, 2009.

[10] J. Z. Lai, D. Robert, S. Liu, J. Weng, and Y. Zhao, "Identity-based encryption secure against selective opening chosen-ciphertext attack," in *EUROCRYPT*, pp. 11–15, Springer, Berlin, Germany, 2014.

[11] J. Ning, Z. Cao, X. Dong, K. Liang, L. Wei, and K. K. R. Choo, "Cryptcloud+: secure and expressive data access control for cloud storage," *IEEE Transactions on Services Computing*, vol. 14, 2018.

[12] J. Ning, X. Huang, W. Susilo, K. Liang, X. Liu, and Y. Zhang, "Dual access control for cloud-based data storage and sharing," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[13] C. Sur, C. D. Jung, Y. Park, and K. H. Rhee, "Chosen-ciphertext secure certificateless proxy re-encryption," in *Communications and Multimedia Security*, pp. 214–232, Springer, Berlin, Germany, 2010.

[14] H. Wang, K. Chen, B. Qin, and L. Wang, "Certificateless encryption secure against selective opening attack," *Security and Communication Networks*, vol. 9, no. 18, pp. 5600–5614, 2016.