SCIENCE CHINA
Information Sciences

• **Supplementary File** •

Fully distributed certificateless threshold signature without random oracles

Wenjie YANG[1,2,3], Weiqi LUO[1*], Xizhao LUO[4*], Jian WENG[1,2,3] & Anjia YANG[1,2,3]

[1]*College of Cyber Security/College of Information Science and Technology, Jinan University, Guangzhou* 510632*, China;*
[2]*Guangdong Key Laboratory of Data Security and Privacy Preserving, Guangzhou* 511443*, China;*
[3]*Guangzhou Key Laboratory of Data Security and Privacy Preserving, Guangzhou* 511443*, China;*
[4]*School of Computer and Technology, Soochow University, Suzhou 215006, China*

Appendix A Framework and security model

Definition 1. *A certificateless threshold signature scheme consists of the following five algorithms which can be over in polynomial time.*

- **Setup.** When giving a security parameter λ, the algorithm returns a system secret key ssk and a system public key spk. spk is available for all interested parties while ssk is kept secret by itself.
- **ExtractPartialPrivateKeyShare (ExtPPKS).** When giving the system secret key ssk, the system public key spk, an entity e, the number n of signers and a threshold parameter t, this algorithm generates and secretly sends the corresponding partial private key share D_{ei} to each signer $\mathcal{S}_i (1 \leqslant i \leqslant n)$.
- **SetUserKey.** When giving the system public key spk, an entity e, the number n of signers and a threshold parameter t, the n signers jointly generate the entity public key PK_e and the respective entity secret value share SV_{ei}. At last, each signer $\mathcal{S}_i$ holds $SK_{ei} = (SV_{ei}, D_{ei})$.
- **Sign.** When giving a message M, the system public key spk, an entity e with the public key PK_e and the corresponding secret key share SK_{ei}, each signer $\mathcal{S}_i$ generates the respective signature share σ_i on M for e. After that, any signer can recover the full signature σ by combining t valid signature shares.
- **Verify.** When giving a message M, its signature σ, the system public key spk and an entity e with the public key PK_e, this algorithm returns a boolean value $b \in \{0, 1\}$. If $b = 1$, σ is valid signature. Otherwise, σ is invalid.

In this work, the above definition denotes a single-KGC CLTS scheme. When considering the fully distributed CLTS concept, it requires that the system secret key which is not explicitly obtained by any party is generated by m distributed KGCs in a cooperative manner. Each KGC$_i$ for $1 \leqslant i \leqslant m$ only possesses a share ssk_i related to the implicit system secret key ssk. In fact, the values ssk_1, ssk_2, ..., ssk_m form a (k, m)-threshold secret sharing to the system secret key ssk. In **ExtPPKS** algorithm, any distributed KGC personally engages in generating the partial private key share to *each* signer.

In our improved security model, we still consider two types attack: public key replacement (PKR*) attack and malicious-but-passive key generation center (MKGC) attack. Besides their original attack abilities defined in previous schemes, these attackers are allowed to corrupt up to $t-1$ arbitrary signers and $k-1$ arbitrary malicious KGCs where t and k are threshold values used to generate a valid signature and recover the implicit system secret key, respectively. Now, we define the following two games to capture them.

Game 1. (for PKR* attacker $\mathcal{A}_1$)

Init. Given a security parameter λ, $\mathcal{C}$ invokes the **Setup** algorithm to initialize the system secret key ssk and the system public key spk. spk is given to $\mathcal{A}_1$ and ssk is kept by itself.

Phase 1. $\mathcal{A}_1$ is allowed to corrupt up to $k-1$ KGCs. For convenience, we suppose that the corrupted KGCs are $\{KGC_j\}_{j=1}^{k-1}$.

Phase 2. In this phase, $\mathcal{A}_1$ performs the following queries adaptively.

- $\mathcal{O}^{ReqPPK}$: When $\mathcal{A}_1$ requests the partial private key for an entity e, $\mathcal{C}$ invokes the **ExtPPKS** algorithm to generate all the partial private key shares $\{D_{ei}\}_{i=1}^{n}$ and the matching verification key shares $\{fvk_{ei}\}_{i=1}^{n}$, which are given to $\mathcal{A}_1$.

- $\mathcal{O}^{ReqSV}$: When $\mathcal{A}_1$ requests the secret value for an entity e, $\mathcal{C}$ runs the **SetUserKey** algorithm to obtain the secret value shares $\{SV_{ei}\}_{i=1}^{n}$, the verification key shares $\{svk_i\}_{i=1}^{n}$ and the matching public key PK_e. Then, $\mathcal{C}$ sends all outputs $(\{SV_{ei}\}_{i=1}^{n}, \{svk_{ei}\}_{i=1}^{n}, PK_e)$ to $\mathcal{A}_1$. Here, $\mathcal{C}$ returns $\perp$ if the entity public key has been updated.

*Corresponding author (email: huanying-87@163.com, xzluo@suda.edu.cn)

- $\mathcal{O}^{RepPK}$: When $\mathcal{A}_1$ replaces the entity public key PK_e with a new public key PK'_e, $\mathcal{C}$ performs the operation and records this replacement.

- $\mathcal{O}^{Sign}$: When $\mathcal{A}_1$ requests a signature σ on a message M for an entity e, $\mathcal{C}$ runs the **Sign** algorithm to respond $\{\sigma_i\}_{i=1}^{n}$ to $\mathcal{A}_1$. It is clear that $\mathcal{A}_1$ is able to obtain a completed signature σ on M with any t valid signature share σ_i.

Phase 3. When $\mathcal{A}_1$ submits the target entity e^*, $\mathcal{C}$ runs the **ExtPPKS** algorithm to obtain the partial private key shares D^*_{ei} for $i = 1, 2, ..., n$ and gives D^*_{ei} for $i = 1, 2, ..., k-1$ to $\mathcal{A}_1$. Then, $\mathcal{A}_1$ adaptively issues a series of requests as in **Phase 2** expect $\mathcal{O}^{ReqPPK}$ on the entity e^*.

Forgery. $\mathcal{A}_1$ outputs $(e^*, PK^*_e, M^*, \sigma^*)$ and eventually wins the game if the conditions hold:

1. σ^* is a valid signature on M^* under e^* with PK^*_e.
2. $\mathcal{A}_1$ has never queried the oracles $\mathcal{O}^{Sign}$ on (e^*, PK^*_e, M^*).

Game 2. (for MKGC attacker $\mathcal{A}_2$)

Init. $\mathcal{C}$ invokes $\mathcal{A}_2$ to initialize the system public/secret keys (spk, ssk). Here, to mount an attack more easily, $\mathcal{A}_2$ is allowed to embed extra trapdoors in the system parameters.

Phase 1. $\mathcal{A}_2$ is allowed to corrupt up to $t-1$ signers. For convenience, we suppose that the first $t-1$ signers are malicious.

Phase 2. $\mathcal{A}_2$ can adaptively query the oracles $\mathcal{O}^{RepPK}$, $\mathcal{O}^{ReqSV}$ and $\mathcal{O}^{Sign}$ as described in **Game 1**.

Phase 3. When $\mathcal{A}_2$ submits the target entity e^*, $\mathcal{C}$ runs the **SetUserKey** algorithm to obtain the secret value shares SV^*_{ei} for $i = 1, 2, ..., n$ and gives SV^*_{ei} for $i = 1, 2, ..., t-1$ to $\mathcal{A}_2$. Then, $\mathcal{A}_2$ adaptively issues a series of requests as in **Phase 2** expect $\mathcal{O}^{ReqSV}$ and $\mathcal{O}^{RepPK}$ on the entity e^*.

Forgery. $\mathcal{A}_2$ outputs $(e^*, PK^*_e, M^*, \sigma^*)$ and wins this game, if

1. σ^* is a valid signature M^* under e^* with PK^*_e.
2. $\mathcal{A}_2$ has never queried the oracle $\mathcal{O}^{Sign}$ on (e^*, PK^*_e, M^*).

Definition 2. If no attackers can win the above two games with a non-negligible advantage, in a probabilistic polynomial time (PPT), a certificateless threshold signature scheme is said to be existential unforgeability against adaptively chosen message attacks (for short EUF-CLTS-CMA).

Appendix B Security analysis of Hu *et al.*'s fully distributed CLTS Scheme

In this section, we briefly review Hu *et al.*'s fully distributed CLTS scheme [1] and analyze its vulnerability by elaborating two concrete attacks. In addition, we will concisely discuss the reasons why their scheme cannot be proven secure without random oracles.

Appendix B.1 Review of Hu *et al.*'s CLTS scheme

Let $(q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, g)$ be a valid instance, where q denotes a big prime, $\mathbb{G}_1$ and $\mathbb{G}_2$ denotes two q order cyclic groups, $\hat{e}$ denotes an admissible bilinear mapping $\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ and g denotes a generator in $\mathbb{G}_1$. Moreover, two collision-resistant hash functions, $H_u : \{0,1\}^* \to \{0,1\}^{n_u}$ and $H_m : \{0,1\}^* \to \{0,1\}^{n_m}$, can be defined and used to create identities and messages of the desired length. In the following, all identities and messages will be assumed to be bit strings of length n_u and n_m, respectively.

Setup. Let $H : \{0,1\}^* \to \mathbb{G}_1$ be a cryptographic hash function. Then, choose random seeds S_h, S_{g_2}, $S_{u'}$, S_{u_1}, ..., $S_{u_{n_u}}$, $S_{m'}$, S_{m_1}, ..., $S_{m_{n_m}}$ and set $h = H(S_h)$, $g_2 = H(S_{g_2})$, $u' = H(S_{u'})$, $m' = H(S_{m'})$, $\mathbf{U} = \{H(S_{u_1}), ..., H(S_{u_{n_u}})\}$, $\mathbf{M} = \{H(S_{m_1}), ..., H(S_{m_{n_m}})\}$. At last, the public parameters are $params = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, g, h, g_2, u', \mathbf{U}, m', \mathbf{M})$ and the validity seeds are $(S_h, S_{g_2}, S_{u'}, S_{m'}, \{S_{u_i}\}_{i=1}^{n_u}, \{S_{m_i}\}_{i=1}^{n_m})$.

ExtPPKS. Let U be a bit string of length n_u representing an identity and let $U[i]$ be the ith bit of U. Define $\mathcal{U} \subset \{1, 2, ..., n_u\}$ to be the set of indices i such that $U[i] = 1$ and compute $D_u = u' \prod_{i \in \mathcal{U}} u_i$.

(1). In order to generate a user partial private key $D = (D_1, D_2) = (g_2^{\alpha}(D_u)^r, g^r)$, each key generation center KGC_i performs interactively as follows:

(a) KGC_i chooses two random polynomials $f_i(z)$ and $g_i(z)$ over $\mathbb{Z}_p$ of degree $t-1$:$f_i(z) = a_{i0} + a_{i1}z + ... + a_{it-1}z^{t-1}$, $g_i(z) = b_{i0} + b_{i1}z + ... + b_{it-1}z^{t-1}$. Let $z_i = a_{i0} = f_i(0)$. KGC_i broadcasts $C_{il} = g^{a_{il}}h^{b_{il}}$ for $l = 0, 1, ..., t-1$. KGC_i computes the shares $s_{ij} = f_i(j) \mod p$, $s'_{ij} = g_i(j) \mod p$ for $j = 1, 2, ..., n$ and sends s_{ij}, s'_{ij} to KGC_j.

(b) Each KGC_j verifies the shares he received from the other $KGCs$. For each $i = 1, 2, ..., n$, KGC_j checks if $g^{s_{ij}}h^{s'_{ij}} = \prod_{l=0}^{t-1}(C_{il})^{j^l}$. If the verification is not passed, then KGC_j broadcasts a complaint to KGC_i.

(c) Each KGC_i who received a complaint from KGC_j broadcasts the values s_{ij}, s'_{ij}. Otherwise, KGC_i is disqualified.

(d) Each KGC_i marks as disqualified any KGC_j that either received more than $t-1$ complaints in Step 1(b), or answered to a complaint in $Step1(c)$ with invalid values.

(e) Each KGC_i then builds the same set of non-disqualified KGCs Q_{KGC} and sets his secret share as $\alpha_i = \sum_{j \in Q_{KGC}} s_{ji}$ mod p and the value $r_i = \sum_{j \in Q_{KGC}} s'_{ji} \mod p$. The distributed master secret α is not explicitly computed by any party, but it equals $\alpha = \sum_{i \in Q_{KGC}} z_i \mod p$. Then KGC_i $(i \in Q_{KGC})$ computes the partial private key share $(D_{i1}, D_{i2}) = (g_2^{\alpha_i}(D_u)^{r'_{ui}}, g^{r'_{ui}})$ and transmits to the corresponding signer $\mathcal{S}_i$ secretly.

(2). KGCs generate the public key $g_1 = g^{\alpha}$ and the first verification key share fvk_i for $i \in Q_{KGC}$.

(a) Each KGC_i broadcasts $A_{il} = g^{a_{il}}$ for $l = 0, 1, ..., t-1$.

(b) KGC_j verifies the values broadcast by the other KGCs, KGC_j checks if $g^{s_{ij}} = \prod_{l=0}^{t-1}(A_{il})^{j_l}$ for $i \in Q_{KGC}$. If the check fails for an index i, KGC_j complains against KGC_i by broadcasting the value s_{ij}, s'_{ij}.

(c) KGC_i in Q_{KGC} recovers an public parameters $g_1 = \prod_{j \in Q_{KGC}} A_{j0}$ and $fvk_i = g^{\alpha_i} = \prod_{j \in Q_{KGC}} \prod_{l=0}^{t-1}(A_{jl})^{i^l}$.

(3). After receiving the partial private key share (D_{i1}, D_{i2}) from KGC_i, the server $\mathcal{S}_i$ checks if the following equation holds: $\hat{e}(D_{i1}, g) = \hat{e}(D_{i2}, D_u)\hat{e}(g_2, fvk_i)$. If the check fails, the signer $\mathcal{S}_i$ broadcasts a complaint against KGC_i.

SetUserKey.

(1) To generate the user secret value $SV = x \in \mathbb{Z}_p^*$, each signer $\mathcal{S}_i$ performs interactively as follows:

(a) $\mathcal{S}_i$ chooses two random polynomials $f_i(z)$ and $g_i(z)$ over $\mathbb{Z}_p$ of degree $t-1$: $f_i(z) = a_{i0} + a_{i1}z + ... + a_{it-1}z^{t-1}$, $g_i(z) = b_{i0} + b_{i1}z + ... + b_{it-1}z^{t-1}$. Let $z_i = a_{i0} = f_i(0)$. $\mathcal{S}_i$ broadcasts $C_{il} = g^{a_{il}}h^{b_{il}}$ for $l = 0, 1, ..., t-1$. $\mathcal{S}_i$ computes the shares $s_{ij} = f_i(j) \mod p$, $s'_{ij} = g_i(j) \mod p$ for $j = 1, 2, ..., n$ and sends s_{ij}, s'_{ij} to $\mathcal{S}_j$.

(b) Each $\mathcal{S}_j$ verifies the shares he received from the other players. For each $i = 1, 2, ..., n$, $\mathcal{S}_j$ checks if $g^{s_{ij}}h^{s'_{ij}} = \prod_{l=0}^{t-1}(C_{il})^{j^l}$. If the verification is not passed, $\mathcal{S}_j$ broadcasts a complaint to $\mathcal{S}_i$.

(c) Each $\mathcal{S}_i$ who received a complaint from $\mathcal{S}_j$ broadcasts the values s_{ji}, s'_{ji}. Otherwise, $\mathcal{S}_i$ is disqualified.

(d) Each $\mathcal{S}_i$ marks as disqualified any $\mathcal{S}_j$ that either received more than $t-1$ complaints in Step 1b, or answered to a complaint in Step 1c with invalid values.

(e) Each $\mathcal{S}_i$ then builds the same set of non-disqualified Q_S and sets his user secret key share as $x_i = \sum_{j \in Q_S} s_{ji} \mod p$. The distributed user secret key x is not explicitly computed by any player, but it equals $x = \sum_{i \in Q_S} z_i \mod p$.

(2) Signers generate the user public key $(PK_1, PK_2)(= (g^x, g_1^x))$ and the second verification key share $svk_i(= g_1^{x_i})$ where $i \in Q_S$, interactively as follows:

(a) Each $\mathcal{S}_i$ broadcasts $(A_{il}^{(1)}, A_{il}^{(2)}) = (g^{a_{il}}, g_1^{a_{il}})$ for $l = 0, 1, ..., t-1$.

(b) To verify the values broadcasted by the other players, $\mathcal{S}_j$ checks if $g^{s_{ij}} = \prod_{l=0}^{t-1}(A_{il}^{(1)})^{j^l}$ and $g_1^{s_{ij}} = \prod_{l=0}^{t-1}(A_{il}^{(2)})^{j^l}$ for $i \in Q_S$. If the check fails, $\mathcal{S}_j$ complains against $\mathcal{S}_i$ by broadcasting the value s_{ij}, s'_{ij}.

(c) Each server in Q_S computes and opens the user public key and the second verification key share $(PK_1, PK_2) = (\prod_{j \in Q_S} A_{j0}^{(1)}, \prod_{j \in Q_S} A_{j0}^{(2)})$, $svk_i = \prod_{j \in Q_S} \prod_{l=0}^{t-1}(A_{jl}^{(2)})^{i^l}$.

Sign. Let M be a bit string representing a message and $\mathscr{M} \subset \{1, 2, ..., n_m\}$ be the set of indices i such that $M[i]$, where $M[i] = 1$ is the ith bit of M.

(1). Each $\mathcal{S}_i$ first picks $r_{ui}, r_{mi} \leftarrow_R \mathbb{Z}_p$, then computes $D_u = u' \prod_{i \in \mathscr{U}} u_i$ and $D_m = m' \prod_{i \in \mathscr{M}} m_i$, finally obtains

$$\sigma'_i = (\sigma'_{i1}, \sigma'_{i2}, \sigma'_{i3}) = (D_{i1}(D_u)^{r_{ui}}(D_m)^{r_{mi}}, D_{i2}g^{r_{ui}}, g^{r_{mi}}).$$

(2). On input $\sigma'_i = (\sigma'_{i1}, \sigma'_{i2}, \sigma'_{i3})$ and first verification key fvk_i, each member j checks if the following equation holds:

$$\hat{e}(\sigma'_{i1}, g) = \hat{e}(g_2, fvk_i)\hat{e}(D_u, \sigma'_{i2})\hat{e}(D_m, \sigma'_{i3}).$$

(3). Let $\lambda'_1, \lambda'_2, ..., \lambda'_t \in \mathbb{Z}_p$ be the Lagrange coefficients so that $\alpha = \sum_{i=1}^{t} \lambda'_i \alpha_i$. Without loss of generality, we assume that servers $\{\mathcal{S}_i\}_{i=1}^{t}$ were used to generate the signature. Each server $\mathcal{S}_j$ computes and publishes the partial signature share as follows:

$$\sigma' = (\sigma'_1, \sigma'_2, \sigma'_3) = (\prod_{i=1}^{t}(\sigma'_{i1})^{\lambda'_i}, \prod_{i=1}^{t}(\sigma'_{i2})^{\lambda'_i}, \prod_{i=1}^{t}(\sigma'_{i3})^{\lambda'_i}).$$

(4). On input the partial signature share $\sigma' = (\sigma'_1, \sigma'_2, \sigma'_3)$, $\mathcal{S}_i$ computes and publishes the complete signature share as follows:

$$\sigma_i = ((\sigma'_1)^{x_i}, (\sigma'_2)^{x_i}, (\sigma'_3)^{x_i}) = (\sigma_{i1}, \sigma_{i2}, \sigma_{i3}).$$

(5). On input $\sigma_i = (\sigma_{i1}, \sigma_{i2}, \sigma_{i3})$ and second verification key svk_i, each $\mathcal{S}_j$ checks if the following equation holds:

$$\hat{e}(\sigma_{i1}, g) = \hat{e}(g_2, svk_i)\hat{e}(D_u, \sigma_{i2})\hat{e}(D_m, \sigma_{i3}).$$

If the check fails, $\mathcal{S}_j$ broadcasts a complaint against $\mathcal{S}_i$.

(6). Let $\lambda_1, \lambda_2, ..., \lambda_t \in \mathbb{Z}_p$ be the Lagrange coefficients so that $x = \sum_{i=1}^{t} \lambda_i x_i$. The signature of U on message M can be computed as

$$\sigma = (\prod_{i=1}^{t}(\sigma_{i1})^{\lambda_i}, \prod_{i=1}^{t}(\sigma_{i2})^{\lambda_i}, \prod_{i=1}^{t}(\sigma_{i3})^{\lambda_i}) = (\sigma_1, \sigma_2, \sigma_3).$$

Verify. Given a signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ on message M of U and the user public key (PK_1, PK_2), the verifier first computes $D_u = u' \prod_{i \in \mathscr{U}} u_i$ and $D_m = m' \prod_{i \in \mathscr{M}} m_i$, then checks if the following equations hold:

$$\hat{e}(PK_1, g_1) = \hat{e}(g, PK_2) \quad \text{and} \quad \hat{e}(\sigma_1, g) = \hat{e}(g_2, PK_2)\hat{e}(D_u, \sigma_2)\hat{e}(D_m, \sigma_3).$$

Output 1 if it is valid. Otherwise, output 0.

Appendix B.2 Analysis of Hu *et al.*'s CLTS scheme

Here, we point out that their scheme is vulnerable by giving two concrete attacks and illustrate the reasons why the scheme is not secure without random oracles.

Appendix B.2.1 *PKR* attack*

Without losing generality, we assume that the PKR* attacker $\mathcal{A}_1$ corrupt the first $t-1$ signers $\{\mathcal{S}_i\}_{i=1}^{t-1}$ and only one key generation center KGC_t. Using the partial private key shares owned by those corrupters, $\mathcal{A}_1$ can recover the complete partial private key $D = (\prod_{i=1}^{t}(D_{i1})^{\lambda_i}, \prod_{i=1}^{t}(D_{i2})^{\lambda_i})$, where $\lambda_1, \lambda_2, ..., \lambda_t$ be the Lagrange coefficients. Although the numbers of the corrupters are less than their respective threshold values, $\mathcal{A}_1$ has been able to recover the entity partial private key. Next, $\mathcal{A}_1$ picks a random secret value $SV' = x'$ from $\mathbb{Z}_p^*$ and sets $(PK_1', PK_2') = (g^{x'}, g_1^{x'})$ which are used to replace the original public key (PK_1, PK_2). Now, $\mathcal{A}_1$ can forge a new signature σ for any message M under the full private key (D, SV') instead of the entity.

Appendix B.2.2 *MKGC attack*

Let $\mathcal{A}_2$ be a MKGC attacker. Here, we will show how to launch an attack by $\mathcal{A}_2$ under the original security model as follows:

Phase 1. At the beginning of system initialization, $\mathcal{A}_2$ honestly generates the public system parameters ($\mathbb{G}_1$, $\mathbb{G}_2$, $\hat{e}$, h, g_2, u', $\mathbf{U}$, m', $\mathbf{M}$) and the validity seeds $(S_h, S_{g_2}, S_{u'}, S_{m'}, (S_{u_i})_{i=1}^{n_u}, (S_{m_i})_{i=1}^{n_m})$, but maliciously sets the generator g as follows: randomly chooses $t \in \mathbb{Z}_p^*$ and sets $g = g_2^t = (H(S_{g_2}))^t$. Note that, we may need to try a few times to find a appropriate value t.

Phase 2. In this stage, $\mathcal{A}_2$ impersonates an entity U^* and signs a message M^* under the public key (PK_1^*, PK_2^*). $\mathcal{A}_2$ chooses two random values r_u^*, r_m^* from $\mathbb{Z}_p$ and computes the signature

$$\sigma^* = (\sigma_1^*, \sigma_2^*, \sigma_3^*) = (PK_2^{*t^{-1}} D_u^{*r_u^*} D_m^{*r_m^*}, g^{r_u^*}, g^{r_m^*})$$

where $D_u^* = u' \prod_{i \in \mathscr{U}^*} u_i$ and $D_m^* = m' \prod_{i \in \mathscr{M}^*} m_i$.

Obviously, σ^* is valid since the following equations hold, where $g_2 = g^{t^{-1}}$

$$\hat{e}(PK_1^*, g_1) = \hat{e}(g, PK_2^*) \quad \text{and} \quad \hat{e}(\sigma_1^*, g) = \hat{e}(g_2, PK_2^*)\hat{e}(D_u^*, \sigma_2^*)\hat{e}(D_m^*, \sigma_3^*).$$

Therefore, the MKGC can easily break the CLTS scheme as well as the underlying CLS scheme.

Appendix B.2.3 *Some flaws in the security proof*

In HT scheme, the system public key are generated by hashing the public seeds to prevent the MKGC attacker from setting some trapdoor information at the **Setup** phase. Unfortunately, this method makes the security proof of their scheme not be simulated without using random oracles. Specifically, if there are a collision-resistant hash function $H : \mathbb{Z}_p^* \to \mathbb{G}_1$ and an item $g_2 = g^b$ from a CDH instance $(\mathbb{G}_1, g, g^a, g^b)$, it is hard to find the matching public seed S_{g_2} from $\mathbb{Z}_p^*$ where $g_2 = H(S_{g_2})$.

Appendix C Basic CLS Scheme

Here, we give a new certificateless signature scheme and prove its security under PKR attacks and MKGC attacks based on [2] (**PIBS** for short) and [3] (**WS** for short), respectively.

Appendix C.1 Framework

The five algorithms below which can be run in polynomial time comprise a certificateless signature scheme:

- **Setup.** When receiving a security parameter λ, the algorithm outputs the system secret key *ssk* and system public key *spk*. *spk* is available for all the other algorithms.
- **ExtPPK.** When receiving the system secret key *ssk* and an entity e, the algorithm outputs the partial private key D_e and sends it over a secure channel to the entity.
- **SetUserKey.** When receiving an entity e, this algorithm outputs the secret key SK_e and public key PK_e. Note that, SK_e is composed of SV_e and D_e, where SV_e denotes the secret value and is picked by the entity itself independently.
- **Sign.** When receiving an entity's secret key SK_e and a message M, this algorithm outputs a signature σ.
- **Verify.** When receiving a signature σ, a message M and an entity public key PK_e, this algorithm outputs either "accept" or "reject" relying on the signature validity.

Like [4], two types of adversaries are taken into account in the above CLS definition. The first is a public key replacement (PKR) attacker who knows the targeted entity secret value but cannot request the entity partial secret key. The other is a malicious-but-passive KGC (MKGC) attacker that is allowed to embed extra trapdoors in their system parameters before running **ExtPPK**, but cannot know the targeted entity secret value. For more details please refer to [4].

Appendix C.2 Concrete scheme

Let $(q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, g)$ be a valid instance, where q denotes a big prime, $\mathbb{G}_1$ and $\mathbb{G}_2$ denotes two q order cyclic groups, $\hat{e}$ denotes an admissible bilinear mapping $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ and g denotes a generator in $\mathbb{G}_1$. Further, we pick two collision resistant hash functions H_m:$\{0,1\}^* \times spk \times PK \rightarrow \{0,1\}^{n_m}$ and H_e:$\{0,1\}^* \rightarrow \{0,1\}^{n_e}$ to handle arbitrary messages and identities, respectively. Note that, the cyclic groups and hash functions are publicly confirmed by all interested parties.

- **Setup.** First, choose random elements α_1, α_2, x', x_1, x_2, ..., x_{n_e}, y', y_1, y_2, ..., y_{n_m} from $\mathbb{Z}_p^*$. Then, compute $g_1 = g^{\alpha_1}, g_2 = g^{\alpha_2}, e' = g^{x'}, \mathbf{E} = \{e_i\}_{i=1}^{n_e} = \{g^{x_i}\}_{i=1}^{n_e}, w' = g^{y'}, \mathbf{W} = \{w_j\}_{j=1}^{n_m} = \{g^{y_j}\}_{j=1}^{n_m}$. Finally, the system secret key is $ssk =(\alpha_1, \alpha_2, x', x_1, x_2, ..., x_{n_e}, y', y_1, y_2, ..., y_{n_m})$ and the system public key is $spk = (\mathbb{G}_1, \mathbb{G}_2, g, g_1, g_2, e', \mathbf{E}, w', \mathbf{W}, H_e, H_m)$.
- **ExtPPK.** Let e be an entity and set $E = H_e(e)$. Let $\mathscr{E}$ be the set of indices $i \subset \{1, 2, ..., n_e\}$ where $E[i] = 1$. To construct the partial private key D_e of the entity e, pick r_e from $\mathbb{Z}_p^*$ at random and compute:

$$D_e = (D_{e1}, D_{e2}) = (g_2^{\alpha_1}(e' \prod_{i\in\mathscr{E}} e_i)^{r_e}, g^{r_e}).$$

- **SetUserKey.** First, choose elements β_{e1}, β_{e2}, z'_e, z_{e1}, z_{e2}, ..., z_{en_m} from $\mathbb{Z}_p^*$ at random. Then, compute $g_{e1} = g^{\beta_{e1}}, g_{e2} = g^{\beta_{e2}}, v'_e = g^{z'_e}, \mathbf{V}_e = \{v_{ej}\}_{j=1}^{n_m} = \{g^{z_{ej}}\}_{j=1}^{n_m}$. Finally, let the secret value be $SV_e =(\beta_{e1}$, β_{e2}, z'_e, z_{e1}, z_{e2}, ..., $z_{en_m})$ and the public key be $PK_e = (g_{e1}, g_{e2}, v'_e, \mathbf{V}_e)$. Note that, $SK_e = (SV_e, D_e)$.
- **Sign.** Given spk, M and PK_e, the signer sets $\mathscr{E} = \{i|E[i] = 1, i = 1, 2, ..., n_e\}$ where $E = H_e(e)$ and $\mathscr{M} = \{j|N[j] = 1, j = 1, 2, ..., n_m\}$ where $N = H_m(M\|spk\|PK_e)$. Then, the signer picks r_m from $\mathbb{Z}_p^*$ at random and generates σ on M for e as follows:

$$\sigma = (g_{e2}^{\beta_{e1}}(v'_e \prod_{j\in\mathscr{M}} v_{ej})^{r_m}, g_2^{\alpha_1}(e' \prod_{i\in\mathscr{E}} e_i)^{r_e}(w' \prod_{j\in\mathscr{M}} w_j)^{r_m}, g^{r_e}, g^{r_m}).$$

- **Verify.** As in the **Sign** algorithm, first compute the sets $\mathscr{E}$ and $\mathscr{M}$. Then, parse the signature σ as $(\sigma_1, \sigma_2, \sigma_3, \sigma_4)$ and verify the following equations:

$$\hat{e}(\sigma_1, g) \overset{?}{=} \hat{e}(g_{e1}, g_{e2})\hat{e}(v'_e \prod_{j\in\mathscr{M}} v_{ej}, \sigma_4) \quad \text{and} \quad \hat{e}(\sigma_2, g) \overset{?}{=} \hat{e}(g_1, g_2)\hat{e}(e' \prod_{i\in\mathscr{E}} e_i, \sigma_3)\hat{e}(w' \prod_{j\in\mathscr{M}} w_j, \sigma_4).$$

Output 1 if the equations hold. Otherwise, output 0.
For simplicity, we may set $ssk = \alpha_1$ and $SV_e = \beta_{e1}$ because the others called implicit system secret keys/secret values are not used in the real system. Here, we explicitly list these values to easily prove its security in the following subsection.

Appendix C.3 Security analysis

In our CLS scheme, we still consider two types classic adversaries, named PKR attacker and MKGC attacker, defined in the previous works. For more details about these adversaries, please refer to [4].
Lemma 1. If the **PIBS** scheme is sound and H_m is collision resistant, our CLS scheme is existentially unforgeable against the PKR attacker under an adaptive chosen-message attack.
Analysis: Assuming that an attacker $\mathcal{A}_1$ can break our CLS scheme in an adaptive chosen-message attack, we can simulate a PPT attacker $\mathcal{B}_1$ who can produce a weak forgery on the **PIBS** scheme. Here, to consistently answer $\mathcal{A}_1$'s queries, $\mathcal{B}_1$ keeps a table T which is initially empty in our security proofs.

Init. $\mathcal{B}_1$ picks at random two collision resistant hash functions H_e and H_m from the defined hash family and returns $(PIBS.params, H_e, H_m)$ to $\mathcal{A}_1$ as our CLS's system public key spk.

Queries. In this query phase, $\mathcal{B}_1$ responds $\mathcal{A}_1$'s all queries involved in the corresponding security model [4] as follows:

- $\mathcal{O}^{ReqPPK}$: $\mathcal{B}_1$ runs $PIBS.Extract(H_e(e))$ to obtain the entity partial private key D_e and returns it to $\mathcal{A}_1$.
- $\mathcal{O}^{ReqSV}$: $\mathcal{B}_1$ searches T to return the entity secret value $SV_e = (\beta_{e1}$, β_{e2}, z'_e, z_{e1}, z_{e2}, ..., $z_{en_w})$. If it does not exist, $\mathcal{B}_1$ first picks $n_w + 3$ random values $(\beta_{e1}$, β_{e2}, z'_e, z_{e1}, z_{e2}, ..., $z_{en_w})$ from $\mathbb{Z}_p^*$ and stores them in T. Finally, $\mathcal{B}_1$ returns SV_e to $\mathcal{A}_1$ as the secret value for e.
- $\mathcal{O}^{ReqPK}$: $\mathcal{B}_1$ searches T to discover the entity e and returns $PK_e = (g_{e1}$, g_{e2}, v'_e, v_{e1}, ..., $v_{en_w})$ to $\mathcal{A}_1$. If it does not exist, $\mathcal{B}_1$ first picks $n_w + 3$ random values $(\beta_{e1}$, β_{e2}, z'_e, z_{e1}, z_{e2}, ..., $z_{en_w})$ from $\mathbb{Z}_p^*$ and then stores these values in T. Finally, $\mathcal{B}_1$ computes $(g^{\beta_{e1}}$, $g^{\beta_{e2}}$, $g^{z'_e}$, $g^{z_{e1}}$, ..., $g^{z_{en_w}})$ and returns it to $\mathcal{A}_1$ as the entity public key.
- $\mathcal{O}^{RepPK}$: $\mathcal{B}_1$ searches T to find the entity e and update its public key using a new public key PK'_e. If it does not exist, $\mathcal{A}_1$ directly sets the entity public key to be PK'_e.
- $\mathcal{O}^{Sign}$: Given the system public key spk, a message M and the entity public key PK_e, $\mathcal{B}_1$ first invokes $PIBS.Sign$ algorithm to obtain a signature $(\sigma'_1, \sigma'_2, \sigma'_3)$ for the message $H_m(M\|spk\|PK_e)$ on the identity $H_e(e)$. Then, $\mathcal{B}_1$ computes the items $\sigma_1 = g^{\beta_{e1}\beta_{e2}}(\sigma'_3)^{\sum_{i=1}^{n_m} z_{ei}}, \sigma_2 = \sigma'_1, \sigma_3 = \sigma'_2$ and $\sigma_4 = \sigma'_3$. Finally, $\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4)$ is returned to $\mathcal{A}_1$ as the requested signature.

Forgery. If $\mathcal{A}_1$ outputs σ^* on M^* for e^* with PK_e^* such that

- $H_m(M^*\|spk\|PK_e^*) = H_m(M\|spk\|PK_e^*)$ but $M^* \neq M$, $\mathcal{B}_1$ succeeds in finding a concrete collision of H_m.
- $H_m(M^*\|spk\|PK_e^*) \neq H_m(M\|spk\|PK_e^*)$, $\mathcal{B}_1$ succeeds in breaking the **PIBS** scheme by forging a valid signature $(\sigma_2^*, \sigma_3^*, \sigma_4^*)$ for the message $H_m(M^*\|spk\|PK_e^*)$ on an identity $H_e(e^*)$.

Note that, M^* is not among those queried messages under the entity e^* during the Sign query phase.
Lemma 2. If the **WS** scheme is sound and H_m is collision resistant, our CLS scheme is existentially unforgeable against MKGC attacker under an adaptive chosen-message attack.

Analysis: Here, we omit the detailed proof, but note that it is analogous to the reduction of **Lemma 1**.

Combining **Lemma 1** and **Lemma 2**, we argue that the proposed certificateless signature scheme is EUF-CLS-CMA in the standard model under the classic CDH intractability assumption.

Appendix D Single-KGC CLTS Scheme

Here, we construct a single-KGC CLTS scheme (SCLTS) from the above basic CLS scheme and demonstrate its security without using random oracles.

Appendix D.1 Concrete scheme

Let $(q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, g)$ be a concrete instance described in Section 2. Further, pick two collision resistant hash functions H_m:$\{0,1\}^* \times spk \times PK \rightarrow \{0,1\}^{n_m}$ and H_e:$\{0,1\}^* \rightarrow \{0,1\}^{n_e}$ to handle arbitrary messages and identities, respectively. Note that, the cyclic groups and hash functions can be publicly verified by all interested parties.

Setup: Let g, h be two generators of $\mathbb{G}_1$ and $\log_g h$ is unknown to any third party. First, choose random elements α_1, α_2, x', x_1, x_2, ..., x_{n_e}, y', y_1, y_2, ..., y_{n_m} from $\mathbb{Z}_p^*$. Then, compute $g_1 = g^{\alpha_1}, g_2 = g^{\alpha_2}, e' = g^{x'}, \mathbf{E} = \{e_i\}_{i=1}^{n_e} = \{g^{x_i}\}_{i=1}^{n_e}, w' = g^{y'}, \mathbf{W} = \{w_j\}_{j=1}^{n_m} = \{g^{y_j}\}_{j=1}^{n_m}$. Finally, let the system secret key be $ssk =(\alpha_1,\ \alpha_2,\ x',\ x_1,\ x_2,\ ...,x_{n_e},\ y',\ y_1,\ y_2,\ ...,y_{n_m})$ and the system public key be $spk = (\mathbb{G}_1, \mathbb{G}_2, g, h, g_1, g_2, e', \mathbf{E}, w', \mathbf{W})$.

ExtPPKS: Let e be an entity and set $E = H_e(e)$. Set $\mathscr{E}$ to be the set of indices $i \subset \{1, 2, ..., n_e\}$ where $E[i] = 1$. To construct the partial private key share D_{ei} for the signer $\mathcal{S}_i$, KGC performs as follows:

(a) Choose $c_1, c_2, ..., c_{t-1}, c'_0, c'_1, ..., c'_{t-1}$ from $\mathbb{Z}_p^*$ uniformly at random and define $G(x) = c_0 + c_1x + ... + c_{t-1}x^{t-1}$, $G'(x) = c'_0 + c'_1x + ... + c'_{t-1}x^{t-1}$ where $c_0 = \alpha_1$.

(b) Compute and open $T_i = T(c_i, c'_i) = \hat{e}(g, g_2)^{c_i}\hat{e}(g, h)^{c'_i}$ for $i = 0, 1, ..., t-1$ as the commitments of α_1 and $G(x)$.

(c) Pick r_{ei} from $\mathbb{Z}_p^*$ at random and set $D_{ei1} = g_2^{G(i)}(e'\prod_{i\in\mathscr{E}} e_i)^{r_{ei}}$, $D_{ei2} = g^{r_{ei}}$, $D_{ei3} = G'(i)$ for $i = 1, 2, ..., n$. In addition, set the first verification key share $fvk_{ei} = \hat{e}(g, g_2)^{G(i)}$ and secretly send $(D_{ei1}, D_{ei2}, D_{ei3})$ to the signer $\mathcal{S}_i$.

SetUserKey: All of the signers first collectively pick a generator h' of $\mathbb{G}_1$ where $\log_g h'$ is unknown and then interactively execute Gennaro's DKG algorithm to initialize partial implicit secret values $(\beta_{e2}, z'_e, z_{e1}, z_{e2}, ..., z_{en_m})$ from $\mathbb{Z}_p^*$ and the corresponding public keys $(g_{e2} = g^{\beta_{e2}}, v'_e = g^{z'_e}, v_{e1} = g^{z_{e1}}, v_{e2} = g^{z_{e2}}, ..., v_{en_m} = g^{z_{en_m}})$ from $\mathbb{G}_1$.

(1) To generate the distributed secret value $SV_e = g_{e2}^{\beta_{e1}} \in \mathbb{G}_1$, each signer $\mathcal{S}_i$ performs interactively as follows:

(a) $\mathcal{S}_i$ picks two random $(t-1)$-degree polynomials $G_i(x)$ and $G'_i(x)$ over Z_p^*: $G_i(x) = c_{i0} + c_{i1}x + ... + c_{it-1}x^{t-1}$, $G'_i(x) = c'_{i0} + c'_{i1}x + ... + c'_{it-1}x^{t-1}$. Let $P_{i0} = c_{i0} = G_i(0)$. $\mathcal{S}_i$ opens $C_{il} = g^{c_{il}}h'^{c'_{il}} \mod p$ for $l = 0, 1, ..., t-1$. $\mathcal{S}_i$ produces the shares $P_{ij} = G_i(j)$, $P'_{ij} = G'_i(j) \mod p$ for $j = 1, 2, ..., n$ and sends P_{ij}, P'_{ij} to $\mathcal{S}_j$.

(b) $\mathcal{S}_j$ checks the shares from the other signers $\mathcal{S}_i$ by verifying the equality $g^{P_{ij}}h'^{P'_{ij}} = \prod_{l=0}^{t-1}(C_{il})^{j^l} \mod p$. If the equality does not hold for an index i, then $\mathcal{S}_j$ opens a complaint to $\mathcal{S}_i$.

(c) $\mathcal{S}_i$ opens the values P_{ij}, P'_{ij}, if he receives a complaint from $\mathcal{S}_j$. Otherwise, $\mathcal{S}_i$ is disqualified.

(d) $\mathcal{S}_j$ marks as disqualified any $\mathcal{S}_i$ that either received more than $t-1$ complaints in Step 1b, or answered to a complaint in Step 1c with invalid values.

(e) $\mathcal{S}_i$ owns the same non-disqualified set $QUAL_e$ and recovers their secret share $\beta_{e1i} = \sum_{j\in QUAL_e} P_{ji} \mod p$ and $\beta'_{e1i} = \sum_{j\in QUAL_e} P'_{ji} \mod p$. Note that, the completed secret value $g_{e2}^{\beta_{e1}}$ is not explicitly obtained by anyone, but it equals $SV_e = \prod_{i\in QUAL_e} g_{e2}^{P_{i0}} \mod p$.

(2) Signers generate the public key $g_{e1} = g^{\beta_{e1}}$ and the second verification key share $svk_i = \hat{e}(g, g_{e2})^{\beta_{e1i}}$ for $i \in QUAL_e$:

(a) $\mathcal{S}_i$ broadcasts $A_{il} = g^{c_{il}}$ for $l = 0, ..., t-1$.

(b) $\mathcal{S}_j$ verifies the share P_{ij} from the other signers, $\mathcal{S}_j$ checks if $g^{P_{ij}} = \prod_{l=0}^{t-1}(A_{il})^{j^l}$ for $i \in QUAL_e$. If the check fails for an index i, $\mathcal{S}_j$ complains against $\mathcal{S}_i$ by broadcasting the values P_{ij}, P'_{ij}.

(c) $\mathcal{S}_i$ in $QUAL_e$ computes and opens $svk_{ei} = \hat{e}(g, g_{e2})^{\beta_{e1i}} = \hat{e}(g_{e2}, \prod_{j\in QUAL_e}\prod_{l=0}^{t-1}(A_{jl})^{i^l})$ and $g_{e1} = \prod_{j\in QUAL_e} A_{j0}$.

When receiving the partial private key share $D_{ei} = (D_{ei1}, D_{ei2}, D_{ei3})$ from KGC, $\mathcal{S}_i$ verifies if $\hat{e}(D_{ei1}, g)\hat{e}(g, h)^{D_{ei3}} \overset{?}{=} \hat{e}(e'\prod_{i\in\mathscr{E}} e_i, D_{ei2})\prod_{l=0}^{t-1} T_l^{i^l}$ and $\hat{e}(D_{ei1}, g) \overset{?}{=} fvk_i \cdot \hat{e}(e'\prod_{i\in\mathscr{E}} e_i, D_{ei2})$. If the verifications fail, $(D_{ei1}, D_{ei2}, D_{ei3})$ assigned to the signer $\mathcal{S}_i$ is invalid. Otherwise, $\mathcal{S}_i$ sets $SK_{ei} = (SV_{ei}, D_{ei1}, D_{ei2}) = (g_{e2}^{\beta_{e1i}}, g_2^{G(i)}(e'\prod_{i\in\mathscr{E}} e_i)^{r_{ei}}, g^{r_{ei}})$ and $PK_e = \{g, h'_e, g_{e1}, g_{e2}, v'_e, \mathbf{V}_e\}$.

Sign: Let M be a message and set $N = H_m(M\|spk\|PK_e)$. Set $\mathscr{M}$ be the set of indices $j \subset \{1, 2, ..., n_m\}$ where $N[j] = 1$. Each signer $\mathcal{S}_i$ first picks r_{mi} from $\mathbb{Z}_p^*$, then computes $D_w = w'\prod_{j\in\mathscr{M}} w_j$ and $D_v = v'\prod_{j\in\mathscr{M}} v_j$, finally broadcasts

$$\sigma_i = (\sigma_{i1}, \sigma_{i2}, \sigma_{i3}, \sigma_{i4}) = (SV_{ei}D_v^{r_{mi}}, D_{ei1}D_w^{r_{mi}}, D_{ei2}, g^{r_{mi}})$$

On input σ_i, fvk_i and svk_i, each participant checks if the following equations hold:

$$\hat{e}(\sigma_{i1}, g) \overset{?}{=} svk_{ei}\cdot\hat{e}(D_v, \sigma_{i4}) \quad \text{and} \quad \hat{e}(\sigma_{i2}, g) \overset{?}{=} fvk_{ei}\cdot\hat{e}(D_u, \sigma_{i3})\hat{e}(D_w, \sigma_{i4}).$$

If the verification does not hold, the participant opens a complaint to $\mathcal{S}_i$.

For simplicity, we assume that the first t signers are chosen to produce the signature. Let $\lambda_1, \lambda_2, ..., \lambda_t$ be the Lagrange coefficients. The signature σ of M on e can be recovered as

$$\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4) = (\prod_{l=1}^{t} \sigma_{l1}^{\lambda_l}, \prod_{l=1}^{t} \sigma_{l2}^{\lambda_l}, \prod_{l=1}^{t} \sigma_{l3}^{\lambda_l}, \prod_{l=1}^{t} \sigma_{l4}^{\lambda_l})$$

Verify: Given $\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4)$ on M for e and the entity public key $PK_e = \{g, h'_e, g_{e1}, g_{e2}, v'_e, \mathbf{V}_e\}$, the verifier first computes $D_u = e' \prod_{i \in \mathscr{E}} e_i$, $D_w = w' \prod_{j \in \mathscr{M}} w_j$ and $D_v = v' \prod_{j \in \mathscr{M}} v_j$, then checks the following equations:

$$\hat{e}(\sigma_1, g) \overset{?}{=} \hat{e}(g_{e1}, g_{e2}) \hat{e}(D_v, \sigma_4) \quad \text{and} \quad \hat{e}(\sigma_2, g) \overset{?}{=} \hat{e}(g_1, g_2) \hat{e}(D_u, \sigma_3) \hat{e}(D_w, \sigma_4).$$

Output 1 if it is valid. Otherwise, output 0.

Appendix D.2 Security analysis

According to **Theorem 1** in [5], we only need to prove that the underlying CLS scheme is EUF-CLS-CMA and the single-KGC CLTS scheme is simulatable. In above section, we have demonstrated that our basic CLS scheme is EUF-CLS-CMA in the standard model. Now, we will illustrate that our single-KGC CLTS scheme is simulatable by means of the three following lemmas.

Lemma 3. The process of extracting private key shares is simulatable in the single-KGC CLTS scheme.

Analysis. In **ExtPPKS** algorithm, we adopt Zhang *et al*'s verifiable secret sharing (VSS) over bilinear groups to issue the partial private key shares for each signers. Thus, the simulator SIM_{EPPKS} can be simulated in the same way as that in [6] for the proof of Zhang *et al*'s VSS over bilinear groups.

Lemma 4. The process of entity key generation is simulatable in the single-KGC CLTS scheme.

Analysis. In **SetUserKey** algorithm, we adopt Gennaro *et al*'s distributed key generation (DKG) technique to generate the corresponding public/secret value for the entity. Thus, the simulator SIM_{SUK} can be simulated in the same way as that in [7] for the proof of Gennaro *et al*'s DKG scheme.

Lemma 5. The process of signature generation is simulatable in the single-KGC CLTS scheme.

Analysis. On input the system public key *spk*, an entity e, a message M, the corresponding signature σ, partial private key shares $(D_{e1}, D_{e2}, ..., D_{et-1})$ and secret value shares $(SV_{e1}, SV_{e2}, ..., SV_{et-1})$ of the corrupted signers, the adversary picks $r_{mi} \in \mathbb{Z}_p^*$ at random and computes $\sigma_i = (\sigma_{i1}, \sigma_{i2}, \sigma_{i3}, \sigma_{i4}) = (SV_{ei}(v' \prod_{j \in \mathscr{M}} v_j)^{r_{mi}}, D_{ei1}(v' \prod_{j \in \mathscr{M}} v_j)^{r_{mi}}, D_{ei2}, g^{r_{mi}})$ for $1 \leqslant i \leqslant t-1$. Using the signature σ and σ_i for $1 \leqslant i \leqslant t-1$, the simulator SIM_S generates $\sigma_j = \sigma / \prod_{i=1}^{t-1} \sigma_i'^{\lambda_{j,i}}$ for $j = t, t+1, ..., n$, with known Lagrange interpolation coefficients $\lambda_{j,i}$. The simulated $\sigma_i (t \leqslant i \leqslant n)$ is correct and identically distributed to the one in the real execution of the "Sign".

Appendix E Security analysis to the fully distributed CLTS

In our single-KGC CLTS scheme, the only KGC picks and sets the system secret key, while in the fully distributed CLTS scheme, all the distributed KGCs cooperatively generate the system secret key by adopting Gennaro *et al.*'s DKG scheme. The only difference between the single-KGC CLTS scheme and the fully distributed CLTS scheme is the system secret key generation ways. Moreover, the single-KGC CLTS scheme has been proven to be secure in the above section. Thus, the security of our fully distributed CLTS scheme is reduced to the security of Gennaro *et al.*'s DKG scheme according to **Theorem 1** in [5]. For more details of the simulation, please refer to [7].

Appendix F Related works

In 2007, Wang *et al.* [8] introduced the primitive of certificateless threshold signature (CLTS). Meanwhile, they gave a concrete scheme and claimed the scheme is provably secure under random oracles [9]. To prevent single point of failure, they also brought in a clerk which distributed its system secret key to multiple KGCs. Nevertheless, it is essentially equal to single-KGC CLTS, in the sense that the clerk itself is the bottleneck of the protocol. Before long, Xiong *et al.* [10] presented a new signle-KGC CLTS scheme provably secure without random oracles. However, Yuan *et al.* [5] in 2010 showed that both [8] and [10] are insecure under their respective security model. They also developed the single-KGC CLTS notions by getting rid of the redundant distributed KGCs. In their scheme, the *single* KGC is in charge of generating the entity partial private key and distributing it by the threshold secret sharing [11]. Since then, some similar single-KGC CLTS schemes were proposed in [12–14] .

In 2010, Xiong *et al.* [15] reconsidered the concept of multiple distributed KGCs in single-KGC CLTS to avoid single KGC of failure or abuse, which is named fully distributed CLTS. In this scenario, the system secret key is produced by all distributed KGCs and each KGC only has a share of the implicit system secret key. Recently, Hu *et al.* [1] illustrated that all the existing CLTS schemes [5,8,10,12–16] are vulnerable against either malicious KGC attacks or public key replacement attacks and gave an efficient fully distributed CLTS scheme which is claimed to be secure in the standard model.

To make it clear, we show a comparison of our CLTS schemes with existing schemes in Table F1. Here, we classify these protocols into two types: single-KGC CLTS and fully distributed CLTS. Note that [8] and [16] were claimed to be fully

1) Clerk denotes a trusted third party who can generate any entity's full partial private key independently.

Table F1 Comparison between our schemes and existing protocols

CLTS	Oracle	Full Dist.	Clerk[1)]	PKR	PKR*	MKGC
WCL [8]	√	×	√	√	×	×
XQL [10]	×	×	-	×	-	×
YZH [5]	√	×	-	√	-	×
YCD [16]	×	×	√	×	×	×
SCLTS	×	×	-	√	-	√
XLQ [15]	×	√	×	×	×	×
HHW [1]	×	√	×	√	×	×
FCLTS	×	√	×	√	√	√

distributed CLTS schemes, but as mentioned before, their protocols are essentially single-KGC CLTS schemes due to the existence of the clerk. From the analysis given in the table, only our FCLTS is a fully distributed CLTS scheme provably secure without using random oracles and can resist all the three types of attacks. In the meantime, as an intermediate result, our SCLTS is also the only single-KGC CLTS protocol that can be proven to be secure in the standard model.

References

1 Hu G, Han L, Wang Z, Xia X. Cryptanalysis and improvement of a certificateless threshold signature secure in the standard model. Inform Sciences, 2013, 247: 174 – 187

2 Paterson K, Schuldt J. Efficient identity-based signatures secure in the standard model. In: Proceedings of Australasian Conference on Information Security and Privacy, Melbourne, 2006. 207 – 222

3 Waters B. Efficient identity-based encryption without random oracles. In: Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, 2005. 114 – 127

4 Au M, Chen J, Liu J, Mu Y, Wong D, Yang G. Malicious KGC attacks in certificateless cryptography. In: Proceedings of ACM Symposium on Information, Computer and Communications Security, Singapore, 2007. 302 – 311

5 Yuan H, Zhang F, Huang X, Mu Y, Susilo W, Zhang L. Certificateless threshold signature secure from bilinear maps. Inform Sciences, 2010, 180: 4714 – 4728

6 Zhang F, Zhang J. Efficient and information-theoretical secure verifiable secret sharing over bilinear groups. Chinese J Electron, 2014, 23: 13 – 17

7 Gennaro R, Jarecki S, Krawczyk H, Rabin T. Secure distributed key generation for discrete-log based cryptosystems. In: Proceedings of International Conferences on the Theory and Application of Cryptographic Techniques, Prague, 1999. 295 – 310

8 Wang L, Cao Z, Li X, Qian H. Simulatability and security of certificateless threshold signatures. Inform Sciences, 2007, 177: 1382 – 1394

9 Canetti R, Goldreich O, Halevi S. The random oracle methodology, revisited. J ACM, 2004, 51: 557 – 594

10 Xiong H, Qin Z, Li F. Simulatability and security of certificateless threshold signatures without random oracles. In: Proceedings of International Conference on Computational Intelligence and Security, Suzhou, 2008. 308 – 313

11 Baek J, Zheng Y. Identity-based threshold signature scheme from the bilinear pairings. In: Proceedings of International Conference on Information Technology: Coding and Computing, Las Vegas, 2004. 124 – 128

12 Yan Zhi, Zhang F, Yang W. Cryptanalysis to a certificateless threshold signature scheme. TELKOMNIKA Indonesian Journal of Electrical Engineering, 2012, 10: 1496 – 1502

13 Zhang W, Li Z, Xu Z, Yang J. Notice of retraction a new certificateless threshold signature scheme for mobile ad hoc network. In: Proceedings of International Conference on Computer and Communication Technologies in Agriculture Engineering, Chengdu, 2010. 338 – 342

14 Zhong Z, Zhang W, Xu Z, Lin C. Certificateless threshold signature scheme without a trusted party. In: Proceedings of International Conference on Communication Systems Networks and Applications, Hong Kong, 2010. 97 – 101

15 Xiong H, Li F, Qin Z. Certificateless threshold signature scheme provably secure in the standard model. Inform Sciences, 2013, 237: 73 – 81

16 Yang P, Cao Z, Dong X. Efficient certificateless threshold signatures without random oracles. J Syst Sci Complex, 2010, 23: 1167 – 1182