Distributed Key Generation for SM9-Based Systems

Rui Zhang[1,2(✉)], Huan Zou[1,2(✉)], Cong Zhang[1,2(✉)], Yuting Xiao[1,2(✉)], and Yang Tao[1(✉)]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100195, China
{r-zhang,zouhuan,zhangcong,xiaoyuting,taoyang}@iie.ac.cn
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract. Identity-Based Cryptography (IBC) is a useful tool for the security of IoT devices, but securely deploying this cryptographic technique to the IoT systems is quite challenging. For instance, a leakage of the master secret key will result in the leakage of all IoT devices' private keys. SM9 is the only approved IBC algorithm standard in China. It is critical to have mechanisms to protect the SM9 master secret keys. In this work, to reduce the risk of the master secret key leakage, we propose a (t, n)-threshold distributed private key generation scheme for SM9 with some techniques from multiparty computation. Our scheme is compatible with all the three SM9 sub-algorithms (i.e., the encryption, signature and key agreement). It is also provably secure and completely eliminates the single point of failures in SM9 that is concerned by the industry. The experimental analysis indicates that the proposed scheme is efficient, e.g., up to 1 million private key generation requests can be handled per day.

Keywords: Identity-Based Cryptography · SM9 · Distributed Key Generation · Threshold cryptography

1 Introduction

Identity-Based Cryptography (IBC) where user's public key is an arbitrary string, is a promising tool for securing the Internet of Things (IoT). In IBCs, all users' private keys are generated from a master secret key msk being privately held by a trusted third party—the Private Key Generator (PKG). Such centralized key generation nature, however, inevitably makes the PKG a single point of failures that is harmful to both system robustness and security: once the single PKG crashes, the user private key generation service halts immediately; once the single PKG is corrupted, the master secret key msk is leaked as a consequence. In fact, the msk leakage problem is of big concern when integrating IBCs to a

Y. Wu and M. Yung (Eds.): Inscrypt 2020, LNCS 12612, pp. 113–129, 2021.
https://doi.org/10.1007/978-3-030-71852-7_8

deployed IoT system. Usually, a user device private key S_{ID} is generated from msk, burned into the device and never changed. It is always more profitable to attack msk than each single device private key. But keeping msk safe seems to be a difficult task. For instance, the master secret key leakage of PlayStation 3[1] has caused tremendous losses.

In general, there are two approaches known in the literature to deal with the msk leakage problem of IBCs. The first approach, such as the certificate-based cryptography [13,15] and certificate-less public key cryptography [3,5,17], lets users contribute to their own private keys with the help of a PKG. Even if the PKG's msk is compromised, the user's private key remains safe as long as the user's secret kept confidential. But this type of solution generally loses in transmission efficiency, since the receiver's certificate or self-generated public key has to be pre-published. Considering IoT networks are often multi-hop routing based, poor transmission efficiency makes this approach less attractive and for most low-cost IoT devices this approach is actually impractical.

The second approach to deal with the msk leakage problem is to adopt the Distributed Key Generation (DKG), by distributing the power of user private key generation among multiple parties rather than a single PKG. The n Key Privacy Authorities (KPAs) based scheme [18] and the n Trusted Authorities (TAs) based scheme [9] allow the n trusted parties to pick their secret keys freely. Both schemes are general methods applicable to all IBC schemes, but they are not compatible with the IBC algorithms after user private key generation (e.g., the encryption, signature and key agreement). In contrast, within schemes following a t-out-of-n DKG fashion, the n PKGs must ensure that their secret keys are sharing one msk. These schemes [7,14,19,20,23] are often based on the Shamir secret sharing or homomorphic Paillier encryption primitive: the former which we refer to as (t, n)-threshold distributed key generation [7,14,23] focuses on the general t-out-of-n case; the latter which we refer to as two-party distributed key generation [19,20] generally focuses on the 2-out-of-2 case specifically. Since the distributedly generated user public/private key keep their original forms, the resulting schemes have good compatibility but heavily rely on concrete mathematical structures. For some IBC schemes with poor homomorphic properties, these schemes could be particularly complicated and inefficient.

SM9 is a Chinese standard for IBC [1,2] that consists of three sub-algorithms: a digital signature scheme, a key agreement scheme and an encryption scheme. Table 1 compares four DKG solutions feasible for SM9. The (t, n)-threshold DKG seems to be the most desirable one, since only it completely eliminates the single point of failures in SM9 where both the security and robustness are achieved.

Difficulties of (t, n)-Threshold DKG for SM9. As stated before, the construction of (t, n)-threshold DKG heavily relies on concrete IBC schemes' mathematical structures. Earlier techniques based on the IBC scheme proposed by Boneh and Franklin [7] (BF-IBC), and proposed by Sakai and Kasahara [14] (SK-IBC) cannot be directly adopted to SM9. For schemes enjoying fully homomor-

[1] The Sony PS3 and Bitcoin crypto hacks. https://tinyurl.com/udg5tyg.

Table 1. Distributed user private key generation schemes for SM9

	Construction	Round	*need a* Secure channel	*eliminate* Key escrow	Compatible	Robust
n KPAs based	Generic	N/A	×	√	×	×
n TAs based	Generic	N/A	√	√	×	×
Two party based	Specific	2	×	√	√	×
(t, n)-threshold based (this work)	Specific	1	√	√	√	√

† Since no negotiation happens among key generation authorities, the "Round" item for the KPA and TA based schemes are listed as not applicable (N/A). A scheme is *compatible* if it doesn't modify the IBC algorithms after private key generation, and is *robust* if key generation authorities can go offline without interrupting the private key generation. *Key escrow* refers to the situation that a single PKG can generate all users' private keys.

phic property like BF-IBC [7], where the user private key $S_{\mathrm{ID}} = [msk]h_{\mathrm{ID}}$ with $[\cdot]$ denoting the elliptic curve scalar multiplication operation and h_{ID} denoting an elliptic curve point hashing from a user's identity string, it is quite straightforward to generate a t-privately Shamir share of $S_{\mathrm{ID}} = [msk]h_{\mathrm{ID}}$ from a t-privately msk share. Whereas in SM9 [10], the user private key $S_{\mathrm{ID}} = [\frac{msk}{msk+F(\mathrm{ID})}]P_2$ with $F(\mathrm{ID})$ denoting the hash value of a user's identity string and P_2 denoting the generator of an additive elliptic curve point group. It is hard for a PKG holding a t-privately Shamir share of msk to generate a t-privately Shamir share of the user private key $S_{\mathrm{ID}} = [\frac{msk}{msk+F(\mathrm{ID})}]P_2$, since msk appears both in the numerator and denominator. This further positions challenges for constructing an efficient (t, n)-threshold DKG for SM9.

Our Contributions. In this paper, we investigate the problem of distributed key generation for SM9 and propose a scheme where both the master secret key msk and user private key S_{ID} are generated in a (t, n)-threshold way. To the best of our knowledge, the proposed (t, n)-threshold Distributed Private Key Generation ((t, n)-DPKG)[2] is the first work that completely eliminates the single point of failures in SM9. Besides security and robustness, our scheme also presents an efficient distributed extraction protocol for the *exponent inversion* IBE family, an open challenge in [14]. By removing one semi-honest BGW distributed multiplication protocol [4,16], the round complexity of our protocol is only 1-round, while the best known solution [14] was with 3-rounds.

Related Work. To reduce the risk of msk leakage, (t, n)-DPKG divides msk into n shares. Each PKG privately holds a share and generates a private key fragment for the user. t PKGs or less cannot derive any information about the

[2] DKG vs. DPKG: DPKG is a branch of DKG. Within IBCs, DPKG captures the property of distributedly generating user private keys more precisely. Besides user private keys, our scheme also generates the master secret key distributedly.

msk, and the complete user private key S_{ID} can only be extracted from at least $t+1$ S_{ID} fragments. Thus (t,n)-DPKG relies heavily on concrete mathematical structures of IBE schemes. Boneh and Franklin [7] came up with the first (t,n)-DPKG scheme based on BF-IBE. As BF-IBE user private key enjoys fully homomorphic property, their scheme allows non-interactive partial private key generation. In comparison, designing such schemes for the *exponent inversion* IBE family [8] (e.g., SK-IBE [21] and SM9-IBE [10]) is not that straightforward. Facilitated by the *sharing the inverse of a shared secret* multiparty computation protocol [6], Smart and Geisler developed a (t,n)-DPKG scheme for SK-IBE [14]. Their scheme requires 3-rounds interaction between PKGs during the partial private key generation phase and a more efficient protocol remains open. Kate and Goldberg then revisited above schemes in [14], and extended them to malicious PKG case with non-interactive proofs of knowledge.

In particular, we notice Xu et al. [23] have presented a similar (t,n)-threshold distributed private key generation solution for SM9. But some insufficiencies exist in Xu et al.'s solution: (1) *correctness*. By distributing $\frac{1}{msk}$ among n PKGs, Xu et al.'s scheme successfully extracts the user private key, but it seems very hard to extract the master public key $P_{pub}=[msk]P_1$ from the shares of $\frac{1}{msk}$ to further extract the user public key. Our scheme shares msk instead, and facilitated by multiparty computation techniques, our scheme can efficiently extract both the user public/private keys from the shares of msk; (2) *completeness*. Xu et al.'s solution didn't describe how to share $\frac{1}{msk}$ among n PKGs in the setup phase. Whereas, we present a completely distributed master key generation protocol which removes the need of pre-distributing msk; (3) *efficiency*. In Xu et al.'s scheme, the distributed extraction phase requires 3-rounds interaction of PKGs. Whereas, only 1-round is needed in our scheme.

2 Preliminaries

Notations. For an integer n, $[n]$ denotes the set $\{1,2,\ldots,n\}$. For a real number n, $\lfloor n \rfloor$ denotes the greatest integer less than or equal to n. Given a set I, $|I|$ denotes the cardinality of I. Vector $\boldsymbol{v}$ having n components is denoted as $\boldsymbol{v}^n$ with n being a non-negative integer. The set of all finite binary strings as $\{0,1\}^*$. If A is an algorithm, then $\mathsf{A}(x)\rightarrow y$ means that running the algorithm A with x as its input gets the output y. Furthermore, we let $y\leftarrow \mathsf{A}(x)$ denote the output y of running the algorithm A with x as its input. The term PPT is abbreviated for probabilistic polynomial-time. A function $negl(\cdot)$ is called negligible, if for any polynomial $p(\cdot)$, there exists some λ_0 such that $negl(\lambda)\leq 1/p(\lambda)$ for every $\lambda>\lambda_0$. Throughout the paper, λ will denote the security parameter.

2.1 (t,n)-Secret Sharing

Definition 1 ((t,n)-Secret Sharing). *A (t,n)-secret sharing in the finite field $\mathbb{F}_p$ is a pair of algorithms (*Share, Reconstruct*):*

- $\mathsf{Share}(s, 1^\lambda)$*: A probabilistic algorithm takes as input the security parameter* 1^λ *and a secret* $s \in \mathbb{F}_p$*. It returns* n *shares* $\{s_1, \ldots, s_n\}$ *of* s*.*
- $\mathsf{Reconstruct}(s_{i_1}, \ldots, s_{i_{t+1}})$*: A deterministic algorithm takes as input at least* $t+1$ *shares* $\{s_{i_1}, \ldots, s_{i_{t+1}}\}$ *of some secret. It returns the secret* s*, that is,* $\mathsf{Reconstruct}(s_{i_1}, \ldots, s_{i_{t+1}}) \rightarrow s$*.*

Definition 2 (Perfect Security of (t, n)-Secret Sharing). *A* (t, n)*-secret sharing scheme (*$\mathsf{Share}, \mathsf{Reconstruct}$*) in finite field* $\mathbb{F}_p$ *is of perfect security if the following properties hold:*

- *Correctness:* $\forall s \in \mathbb{F}_p, \forall I \subset [n]$ *s.t.* $|I| > t, \Pr[\mathsf{Reconstruct}(s_i : i \in I, s_1, \ldots, s_n \leftarrow \mathsf{Share}(s)) = s] = 1$
- *Security:* $\forall s, s' \in \mathbb{F}_p, \forall I \subset [n]$ *s.t.* $|I| \leq t$*, the two distributions are the same:* $\{\{s_i\}_{i \in I} : \{s_1, \ldots, s_n\} \leftarrow \mathsf{Share}(s)\}$ $\{\{s'_i\}_{i \in I} : \{s'_1, \ldots, s'_n\} \leftarrow \mathsf{Share}(s')\}$.

2.2 Identity-Based Encryption with a Single PKG

Boneh and Franklin [7] formalized an Identity-Based Encryption (IBE) scheme as four algorithms:

- $\mathsf{Setup}(1^\lambda) \rightarrow (msk, mpk)$: The setup algorithm takes 1^λ as its input. It returns a master public key mpk and a master secret key msk.
- $\mathsf{Extract}(mpk, msk, \mathrm{ID}) \rightarrow S_{\mathrm{ID}}$: The private key extraction algorithm takes as input a key pair (mpk, msk) and an identity $\mathrm{ID} \in \{0,1\}^*$. It returns a user private key S_{ID} for identity ID.
- $\mathsf{Enc}(mpk, \mathrm{ID}, m) \rightarrow c$: The encryption algorithm takes as input the master public key mpk, an identity ID, and a message m. It returns a ciphertext c.
- $\mathsf{Dec}(mpk, S_{\mathrm{ID}}, c) \rightarrow m$ or $\perp$: The decryption algorithm takes as input the master public key mpk, a user private key S_{ID}, and a ciphertext c. It returns a message m or $\perp$ denoting a failure.

Boneh and Franklin [7] also formalized the security notion of an IBE scheme as IND-ID-CCA secure, by defining the following two-stage game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:

- **Setup**. $\mathcal{C}$ runs the setup algorithm and obtains (mpk, msk). Then $\mathcal{C}$ sends mpk to $\mathcal{A}$ and keeps msk to respond $\mathcal{A}$'s queries.
- **Phase 1**. $\mathcal{A}$ adaptively makes private key extraction queries and decryption queries. For a private key extraction query $\langle \mathrm{ID} \rangle$, $\mathcal{C}$ returns S_{ID} to $\mathcal{A}$ by running $\mathsf{Extract}(mpk, msk, \mathrm{ID})$; For a decryption query $\langle \mathrm{ID}, c \rangle$, $\mathcal{C}$ sends decrypted c to $\mathcal{A}$ by running $\mathsf{Dec}(mpk, \mathsf{Extract}(mpk, msk, \mathrm{ID}), c)$.
- **Challenge**. $\mathcal{A}$ outputs a tuple $\{m_0, m_1, \mathrm{ID}^*\}$ where m_0 and m_1 are two distinct messages with the same length, ID^* is an identity for which $\mathcal{A}$ never issues a private key extraction query in Phase 1. Then $\mathcal{C}$ picks a random bit $b \in \{0,1\}$, and sends c_b^* to $\mathcal{A}$ by computing $c_b^* = \mathsf{Enc}(mpk, \mathrm{ID}^*, m_b)$.

- **Phase 2**. $\mathcal{A}$ continues to make private key extraction queries and decryption queries. $\mathcal{C}$ responds just as Phase 1 except for the private key extraction query $\langle \mathrm{ID}^* \rangle$ and the decryption query $\langle \mathrm{ID}^*, c_b^* \rangle$.
- **Guess**. $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ of b and wins the game if $b' = b$.

Definition 3 (IND-ID-CCA Security of IBE Scheme). *An IBE scheme is secure in the IND-ID-CCA model if for any* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* $ngel(\cdot)$ *satisfying:*

$$Adv_{\mathcal{A}}^{IND\text{-}ID\text{-}CCA_{IBE}} = 2|Pr[b' = b] - \frac{1}{2}| \leq negl(\lambda).$$

2.3 The SM9 Private Key Generation

There are 3 sub-algorithms, namely encryption, signature, key agreement in SM9; their key generation is essentially the same. Besides, our proposal will only affect the key generation phase. Due to space limitation, here we only introduce some necessary notions used in the SM9 private key generation. For complete SM9 schemes, one can redirect to [10] for more details.

Let a bilinear pairing mapping define as $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1$, $\mathbb{G}_2$ are additive groups and $\mathbb{G}_T$ is a multiplicative group. All three groups have prime order p. $\mathbb{G}_1$ and $\mathbb{G}_2$ are generated by $P_1 \in \mathbb{G}_1$, $P_2 \in \mathbb{G}_2$, respectively. Assume a random $s \in \mathbb{Z}_p^*$ is chosen as a global master secret key and the master public key for SM9 encryption scheme can be defined as $P_{pub} = [s]P_1$. Then the private key extraction algorithm computes the user's public key as $Q_{\mathrm{ID}} = [F(\mathrm{ID})]\, P_1 + P_{pub}$ and the user's private key as $S_{\mathrm{ID}} = \left[\frac{s}{s+F(\mathrm{ID})}\right] P_2$,

2.4 Dealerless Replicated Secret Sharing Protocol $\mathcal{P}_{\mathrm{rep}}^{\sigma}$

$\mathcal{P}_{\mathrm{rep}}^{\sigma}$ is a protocol that allows n players to jointly determine a random secret σ of a t-privately replicated secret sharing scheme, without a trusted dealer:

$$\mathcal{P}_{\mathrm{rep}}^{\sigma}(\mathcal{G}, \ldots, \mathcal{G}) = (\sigma_1, \ldots, \sigma_n)$$

The input for each player is the public system parameters $\mathcal{G} = \{t, n, p\}$, where t is the threshold, n is the number of players and p is a prime number. The output for each player P_i is a t-privately replicated share σ_i of secret σ. The protocol proceeds as follows: each player $P_{i,i\in[n]}$ chooses a random secret $\mu_i \in \mathbb{Z}_p$ and shares μ_i to player $P_{j,j\in[n]}$ according to the replicated secret sharing scheme [22]. The share that P_i sends to P_j is denoted as $\mathcal{R}_{(t,n)}^{\mu_i}(j)$. Then $P_{i,i\in[n]}$ outputs $\sigma_i = \sum_{j=1}^{n} \mathcal{R}_{(t,n)}^{\mu_j}(i)$. Finally, the shares $\{\sigma_1, \ldots, \sigma_n\}$ determine a random replicated secret scheme $\mathcal{R}_{(t,n)}^{\sigma}$ where $\sigma = \mu_1 + \ldots + \mu_n$ and $\sigma_i = \mathcal{R}_{(t,n)}^{\sigma}(i)$.

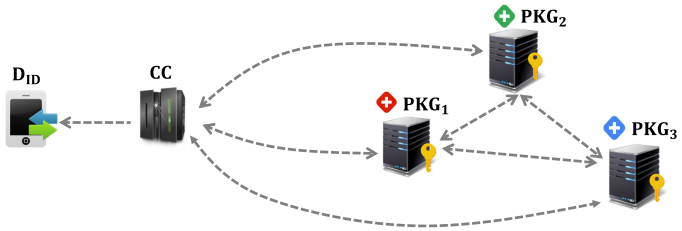

Fig. 1. Architecture of the distributed private key generation scheme

2.5 Share Conversion Algorithm

$\mathsf{Conv}^{*}_{(t,n)}$ is a share conversion algorithm used in the Pseudo-Random Secret Sharing (PRSS) protocol [12,22] that locally converts player $P_{i,i\in[n]}$'s t-privately replicated share $\mathcal{R}^{\sigma}_{(t,n)}(i)$ to a t-privately pseudo-random Shamir share $\mathcal{S}^{z}_{(t,n)}(i)$ sharing a pseudo-random secret z. st is a common input for all n players.

$$\mathsf{Conv}^{*}_{(t,n)}(\mathcal{R}^{\sigma}_{(t,n)}(i), st) \rightarrow \mathcal{S}^{z}_{(t,n)}(i)$$

$\mathsf{Conv}^{0}_{(2t,n)}$ is a share conversion algorithm used in the Pseudo-Random Zero Sharing (PRZS) protocol [12,22] that locally converts player $P_{i,i\in[n]}$'s t-privately replicated share $\mathcal{R}^{\sigma}_{(t,n)}(i)$ to a $2t$-privately pseudo-random Shamir share $\mathcal{S}^{0}_{(2t,n)}(i)$ sharing secret 0. st is a common input for all n players.

$$\mathsf{Conv}^{0}_{(2t,n)}(\mathcal{R}^{\sigma}_{(t,n)}(i), st) \rightarrow \mathcal{S}^{0}_{(2t,n)}(i)$$

3 Threshold Distributed Private Key Generation for IBE

In this section, we introduce the system model, formal definition and properties of (t, n)-threshold Distributed Private Key Generation ((t, n)-DPKG) for IBE.

3.1 System Model and Security

The proposed distributed private key generation scheme involves 3 entities shown in Figure 1. Their characteristics and functionalities are introduced as follows:

- **Private Key Generator (PKG):** It is a powerful entity holding a master secret key share, who generates private key fragments for IoT devices.
- **Combine Center (CC):** It is a stateless entity, whose major task is to perform some complex cryptographic computation. CC collects private key fragments $S^{(i)}_{\mathrm{ID}}$ from PKGs, extracts the complete private key S_{ID} by combing the $S^{(i)}_{\mathrm{ID}}$ fragments then installs S_{ID} into the IoT device. Once the S_{ID} has been successfully installed, CC immediately erases the memory related to S_{ID}.

– **IoT Device ($\mathbf{D_{ID}}$):** It is a resource-constrained entity. D_{ID} wants to get its private key S_{ID} installed before leaving the factory.

Since key generation takes place inside the factory, not exposed in an open environment. We assume all communications shown in Fig. 1 are done via secure channels under synchronous network setting.

For the security goals, the proposed scheme should prevent two types of adversaries – one residing in the private key generation centers, and the other residing in the private key combine center.

– **Corrupted PKG Coalition.** We assume the adversary can control up to t PKGs, learning at most t master secret key shares. Specifically, we assume the adversary is *static* – the corrupted PKG set is fixed before the game. Since behaviors deviating the predefined rules will be quickly detected, the corrupted PKG coalition is assumed to be *semi-honest* – they will fulfill faithfully promised tasks. The security goal is that this corrupted PKG coalition learns no more information than its members' master secret key shares.
– **Corrupted CC.** The adversary residing at the combine center is assumed to be *active*. It is able to generate arbitrary legal identities representing IoT devices and normally interact with PKGs. The security goal is to ensure that this adversary learns no more information than S_{ID} for which it has queried.

3.2 Security Definition

In this part, we revisit the security definition of (t, n)-DPKG for IBE proposed by Kate and Goldberg [14]. For comprehension, these schemes are described within the background of the proposed system model. An IBE scheme with (t, n)-threshold distributed private key generation consists of four components:

– The distributed setup: $\mathsf{DSetup}(t, n, \mathcal{G}) \rightarrow (msk_i, \boldsymbol{mpk}^{n+1})$. Each $\mathrm{PKG}_{i,i\in[n]}$ takes in a threshold t, the number of PKGs n and public system parameters $\mathcal{G}$. It returns a t-privately share msk_i of msk and a vector $\boldsymbol{mpk}^{n+1} = \{mpk_1, \ldots, mpk_n, mpk\}$, where mpk_i denotes the i_{th} share of mpk.
– The distributed extraction: it involves a *distributed extraction* protocol DExtract ran by PKGs and a Combine algorithm locally ran by the CC.

$$\mathsf{DExtract}(\mathrm{ID}, msk_i, mpk) \rightarrow S_{\mathrm{ID}}^{(i)}$$
$$\mathsf{Combine}(S_{\mathrm{ID}}^{(1)}, S_{\mathrm{ID}}^{(2)}, \ldots, S_{\mathrm{ID}}^{(m)}) \rightarrow S_{\mathrm{ID}}$$

In DExtract, each $\mathrm{PKG}_{i,i\in[n]}$ takes in an identity ID, a t-privately share msk_i of msk and mpk. It will output a t-privately share $S_{\mathrm{ID}}^{(i)}$ of the device private key S_{ID}. Having received $m \geq t + 1$ shares of S_{ID}, CC will run the Combine algorithm to compute S_{ID}.
– The encryption: $\mathsf{Enc}(mpk, \mathrm{ID}, m) \rightarrow c$. It is the same as the single PKG.
– The decryption: $\mathsf{Dec}(mpk, S_{\mathrm{ID}}, c) \rightarrow m$ or $\perp$. It is the same as the single PKG.

Kate and Goldberg [14] formalized the security notion of (t, n)-DPKG for IBE as IND-ID-CCA secure, by defining an IND-ID game that a challenger $\mathcal{C}$ plays against a Byzantine adversary who can control up to t PKGs and make them behave arbitrarily. In this work, we assume the corrupted PKGs are semi-honest instead of malicious, so we have made two modifications to Kate and Goldberg's IND-ID game definition [14]: (1) proofs for private key shares are not required, since the semi-honest assumption implies that all PKGs will fulfill their tasks faithfully and will always generate correct shares as required; (2) only $n \geq t$ is required, instead of $n \geq 2t + 1$ required by the malicious PKG assumption. The IND-ID game under the semi-honest PKG assumption is defined as:

Before the game, the adversary $\mathcal{A}_{(t,n)}$ fixes a set of corrupted PKGs denoted as A with $|A| \leq t$ (for general purpose, we assume $|A| = t$), and the challenger $\mathcal{C}$ will simulate the rest $n - t$ honest PKGs denoted as B with $|B| = n - t$.

- **Setup**. $\mathcal{C}$ simulates $\text{PKG}_{i,i\in B}$ and runs the distributed setup protocol with $\mathcal{A}_{(t,n)}$. In the end, $\mathcal{A}_{(t,n)}$ will receive $\boldsymbol{msk}^t = \{msk_i\}_{i\in A}$ contains t shares of msk for $\text{PKG}_{i,i\in A}$, and $\boldsymbol{mpk}^{n+1} = \{mpk_1, \ldots, mpk_n, mpk\}$ contains n shares of mpk generated by $\text{PKG}_{i,i\in A\cup B}$ and mpk.
- **Phase 1**. $\mathcal{A}_{(t,n)}$ adaptively makes private key extraction $\langle \text{ID} \rangle$ queries and decryption $\langle \text{ID}, c \rangle$ queries. For a $\langle \text{ID}, c \rangle$ query, $\mathcal{C}$ decrypts c using its msk then sends decrypted c to $\mathcal{A}_{(t,n)}$. For a $\langle \text{ID} \rangle$ query, $\mathcal{C}$ simulates $\text{PKG}_{i,i\in B}$ running the distributed private key extraction protocol with $\mathcal{A}_{(t,n)}$, and sends $\boldsymbol{S}_{\text{ID}}^{n-t}$ to $\mathcal{A}_{(t,n)}$ where $\boldsymbol{S}_{\text{ID}}^{n-t} = \{S_{\text{ID}}^{(i)}\}_{i\in B}$ are shares of S_{ID} generated by $\text{PKG}_{i,i\in B}$.
- **Challenge**. $\mathcal{A}_{(t,n)}$ outputs a tuple $\{m_0, m_1, \text{ID}^*\}$ where m_0 and m_1 are two distinct messages with the same length, an identity ID^* for which $\mathcal{A}_{(t,n)}$ never issues a private key extraction query in Phase 1. Then $\mathcal{C}$ picks a random bit $b \in \{0, 1\}$, and sends c_b^* to $\mathcal{A}_{(t,n)}$ by computing $c_b^* = \mathsf{Enc}(mpk, \text{ID}^*, m_b)$.
- **Phase 2**. $\mathcal{A}_{(t,n)}$ continues to make private key extraction queries and decryption queries. $\mathcal{C}$ responds just as Phase 1 except for the private key extraction query $\langle \text{ID}^* \rangle$ and the decryption query $\langle \text{ID}^*, c_b^* \rangle$.
- **Guess**. $\mathcal{A}_{(t,n)}$ outputs a guess $b' \in \{0, 1\}$ of b and wins the game if $b' = b$.

Definition 4 (IND-ID-CCA Security of IBE Scheme With (t, n)-DPKG). *With (t, n)-threshold distributed private key generation, an IBE scheme is secure in the IND-ID-CCA model if for any* PPT *adversary $\mathcal{A}_{(t,n)}$, there exists a negligible function $ngel(\cdot)$ satisfying:*

$$Adv_{\mathcal{A}_{(t,n)}}^{IND\text{-}ID\text{-}CCA_{(t,n)}\text{-}IBE} = 2|Pr[b' = b] - \frac{1}{2}| \leq negl(\lambda).$$

In fact, $\mathcal{A}_{(t,n)}$ depicted in the above IND-ID-CCA game models an attacker $\mathcal{A}'$ who corrupts t PKGs as well as the CC. $\mathcal{A}_{(t,n)}$'s failure in the IND-ID-CCA game also indicates that attacker $\mathcal{A}'$ learns no more information than t corrupted PKGs' msk shares and the device private keys S_{ID} for IDs it has queried for.

4 Construction of (t, n)-DPKG for SM9

As the (t, n)-threshold Distributed Private Key Generation ((t, n)-DPKG) won't change the original forms of user public/private key, it is compatible with the original SM9 algorithms after user private key extraction. Hence we only focus on the first two phases – distributed setup and extraction. Construction of (t, n)-DPKG for SM9 relies on (t, n)-Shamir secret sharing, and the challenging task is to let $\text{PKG}_{i,i\in[n]}$ holding a t-privately Shamir share msk_i to generate a t-privately Shamir share of $S_{\text{ID}} = [\frac{msk}{msk+F(\text{ID})}]P_2$. To do this, we first rewrite S_{ID} as $[1 - \frac{F(\text{ID})}{msk+F(\text{ID})}]P_2$, and employ a *sharing the inverse of a shared secret* protocol [6] enabling PKG_i holding a msk share msk_i to obtain a t-privately Shamir share θ_i of $\frac{1}{msk+F(\text{ID})}$. Then PKG_i can locally convert θ_i to a t-privately Shamir share of $S_{\text{ID}} = [\frac{msk}{msk+F(\text{ID})}]P_2$ by computing $[1-F(\text{ID})\cdot\theta_i]P_2$. Besides, to reduce interactions between PKGs, an auxiliary variable σ and share conversion algorithms $\mathsf{Conv}^*_{(t,n)}$ and $\mathsf{Conv}^0_{(2t,n)}$ are introduced to provide Shamir shares.

A. System Bootstrapping

In this phase, n PKGs are supposed to collaboratively determine the following public system parameters:

(1) Determine PKG group size n and threshold t such that $n \geq 2t + 1$.[3] If unsatisfied, decline to proceed.
(2) Agree on parameters $\mathcal{G} = \{1^\lambda, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, \hat{e}, P_1, P_2, H_v, hid\}$ required by the SM9-IBE scheme [10].

B. Distributed Setup

In this phase, n PKGs jointly determine a secret msk where $\text{PKG}_{i,i\in[n]}$ obtains a t-privately msk share msk_i.

- $\mathsf{DSetup}(t, n, \mathcal{G}) \rightarrow (msk_i, \boldsymbol{mpk}^{n+1})$: is a protocol jointly ran by n PKGs.

(1) $\text{PKG}_{i,i\in[n]}$ jointly runs the $\mathcal{P}^\sigma_{\text{rep}}$ protocol defined in Sect. 2.4, that is, $\mathcal{P}^\sigma_{\text{rep}}(\{t, n, p\}) \rightarrow \sigma_i$. At the end of $\mathcal{P}^\sigma_{\text{rep}}$ execution, n PKGs will collaboratively determine a global secret σ and $\text{PKG}_{i,i\in[n]}$ will privately output σ_i, which represents a t-privately replicated share $\mathcal{R}^\sigma_{(t,n)}(i)$ sharing the secret σ.
(2) $\text{PKG}_{i,i\in[n]}$ locally runs the PRSS share conversion algorithm $\mathsf{Conv}^*_{(t,n)}(\sigma_i, st_0) \rightarrow s_i$, where st_0 is a string representing an agreed-upon initial global state (e.g. Lamport timestamp). The private output s_i is a t-privately pseudo-random Shamir share, sharing the master secret key s.
(3) $\text{PKG}_{i,i\in[n]}$ *publicly* outputs a t-privately Shamir share mpk_i of mpk by computing $mpk_i = [s_i]P_1$.

[3] $n \geq 2t+1$ is required because the distributed extraction phase of SM9 involves secret reconstruction from $2t$-privately Shamir shares.

(4) $\text{PKG}_{i,i\in[n]}$ reconstructs the master public key mpk by performing Lagrange polynomial interpolation on received $m \geq t+1$ mpk shares. That is, $mpk = \sum_{i=1}^{m}[c_i]mpk_i$ where $c_i = \prod_{j=1,j\neq i}^{m} \frac{-j}{i-j}$.
(5) $\text{PKG}_{i,i\in[n]}$ *privately* outputs $msk_i = \{\sigma_i, s_i\}$ and *publicly* outputs $\boldsymbol{mpk}^{n+1} = \{mpk_1, \ldots, mpk_n, mpk\}$. Here, s_i is a t-privately Shamir share of the master secret key of SM9 and σ_i is an auxiliary variable to generate Shamir shares (all Shamir shares can be re-computed from σ_i including s_i). To avoid confusion, the master secret key of SM9 is denoted as s instead of msk in the following discussions.

C. Distributed Private Key Extraction

This phase includes a $\mathsf{DExtract}$ protocol jointly ran by n PKGs where $\text{PKG}_{i,i\in[n]}$ will generate a private key fragment $S_{\text{ID}}^{(i)}$ for the IoT device D_{ID}, and a $\mathsf{Combine}$ algorithm locally ran by the CC where the CC will extract the complete private key S_{ID} from the received private key fragments.

- $\mathsf{DExtract}(\text{ID}, msk_i, mpk) \rightarrow S_{\text{ID}}^{(i)}$: is a protocol collaboratively ran by n PKGs.

(1) $\text{PKG}_{i,i\in[n]}$ locally computes $x_i = s_i + F(\text{ID})$, which is a t-privately Shamir share sharing secret $x = s + F(\text{ID})$.
(2) $\text{PKG}_{i,i\in[n]}$ locally invokes $\mathsf{Conv}^{*}_{(t,n)}(\sigma_i, st) \rightarrow r_i$, where r_i is a pseudo-random t-privately Shamir share sharing some pseudo-random secret r. Here, st denotes the current agreed-upon global state.
(3) $\text{PKG}_{i,i\in[n]}$ locally invokes $\mathsf{Conv}^{0}_{(2t,n)}(\sigma_i, st) \rightarrow y_i$, where y_i is a $2t$-privately pseudo-random Shamir share sharing secret 0.
(4) $\text{PKG}_{i,i\in[n]}$ locally computes $z_i = x_i \cdot r_i + y_i$, which is a $2t$-privately *pseudo-random* Shamir share sharing secret $z = x \cdot r$, with $x = s + F(\text{ID})$.
(5) $\text{PKG}_{i,i\in[n]}$ first reveals z_i to the rest $n-1$ PKGs, then reconstructs the $2t$-privately secret z. This step requires at least $2t+1$ PKGs to be online, that is, $n \geq 2t+1$.
(6) $\text{PKG}_{i,i\in[n]}$ locally computes $\omega_i = 1 - F(\text{ID}) \cdot \theta_i$, where $\theta_i = \frac{r_i}{z}$ is a t-privately pseudo-random Shamir share of $\frac{1}{s+F(\text{ID})}$. Therefore, ω_i is a t-privately pseudo-random Shamir share of $\frac{s}{s+F(\text{ID})}$.
(7) $\text{PKG}_{i,i\in[n]}$ sends $S_{\text{ID}}^{(i)} = [\omega_i]P_2$ to CC.

In [14], Geisler and Smart reconstructed the product of x and r based on t-privately Shamir shares of $x \cdot r$. However, to obtain this t-privately share, $\text{PKG}_{i,i\in[n]}$ has to run a semi-honest BGW distributed multiplication protocol [4] with the remaining $n-1$ PKGs. We reconstruct $x \cdot r$ from $2t$-privately Shamir shares instead, where $\text{PKG}_{i,i\in[n]}$ can locally obtain its $2t$-privately Shamir share of $x \cdot r$. In this way, we avoid invoking one distributed multiplication protocol and only 1 round interaction between PKGs are needed to recover the secret $x \cdot r$ from its $2t$-privately Shamir shares.

Having received $m \geq t+1$ private key fragments from PKGs, CC invokes the Lagrange polynomial interpolation algorithm to obtain the complete private key.

- $\mathsf{Combine}(S_{\mathrm{ID}}^{(1)}, \ldots, S_{\mathrm{ID}}^{(m)}) \rightarrow S_{\mathrm{ID}}$: is an algorithm locally ran by the CC. CC derives the complete key via:

$$S_{\mathrm{ID}} = \sum_{i=1}^{m} [c'_i] S_{\mathrm{ID}}^{(i)}, \quad \text{where } c'_i = \prod_{j=1, j \neq i}^{m} \frac{-j}{i-j}.$$

Correctness Analysis. To show above construction is correct, we only need to show that the device public/private key generated in a (t, n)-threshold way is the same as the one generated in a centralized way. For the device public key, we only need to show that the master public key $mpk = [s]P_1$ is correctly extracted from $mpk_i = [s_i]P_1$ fragments. Namely, the equation $mpk = [s]P_1 = \sum_{i=1}^{m} [c_i] mpk_i$ should hold where c_i denotes the Lagrange coefficient. Guaranteed by the correctness of (t, n)-threshold Shamir secret sharing, $mpk = [s]P_1 = [\sum_{i=1}^{m} c_i s_i]P_1 = \sum_{i=1}^{m} [c_i s_i]P_1 = \sum_{i=1}^{m} [c_i] mpk_i$ where $mpk_i = [s_i]P_1$. Therefore, the device public key can be correctly extracted. Similarly, we can verify that the device private key can be correctly extracted too.

5 Security Analysis

In this section, we prove that the distributed form of PKGs won't downgrade the security level of SM9 schemes by reducing the multiple PKG scenario to a single PKG scenario. Specifically, we choose SM9-IBE as an illustration. In [11], Cheng gave a rigorous proof of SM9-IBE as IND-ID-CCA secure, so we have:

Theorem 1. *If SM9-IBE scheme is secure in the IND-ID-CCA model, then SM9-IBE scheme with* (t, n)-DPKG *is secure in the IND-ID-CCA model.*

Proof. The central idea of the proof is that, if there exists an adversary $\mathcal{A}_{(t,n)}$ that wins the IND-ID-CCA game under the (t, n)-DPKG setting with advantage ϵ, then we are able to construct a simulator $\mathcal{S}$ to win the IND-ID-CCA game under the single PKG setting with the same advantage ϵ. The key step in this reduction is to create a simulator $\mathcal{S}$ which can perfectly simulate a view for $\mathcal{A}_{(t,n)}$ in a real attack. Specifically, we denote the pre-fixed corrupted PKG set chosen by $\mathcal{A}_{(t,n)}$ as A with $|A| = t$, and the remaining honest PKG set simulated by $\mathcal{S}$ as B with $|B| = n - t$.

■ **Setup.** $\mathcal{S}$ simulates $\mathrm{PKG}_{i,i \in B}$ and runs the distributed setup protocol with $\mathcal{A}_{(t,n)}$. In the end, $\mathcal{A}_{(t,n)}$ receives $(\boldsymbol{msk}^t, \boldsymbol{mpk}^{n+1})$ where $\boldsymbol{msk}^t = \{msk_i\}_{i \in A}$ contains t shares of msk for $\mathrm{PKG}_{i,i \in A}$, and $\boldsymbol{mpk}^{n+1} = \{mpk_1, \ldots, mpk_n, mpk\}$ contains n shares of mpk generated by $\mathrm{PKG}_{i,i \in A \cup B}$ and the mpk.

As $\mathcal{S}$ wants to leverage $\mathcal{A}_{(t,n)}$ to help it answer the challenge proposed by the challenger $\mathcal{C}$ under the single PKG setting, $\mathcal{S}$ should convince $\mathcal{A}_{(t,n)}$ that: 1) $\mathcal{A}_{(t,n)}$'s output $\boldsymbol{msk}^t$ are t shares of the msk chosen by the challenger $\mathcal{C}$ even if $\mathcal{S}$ has no idea about the msk chosen by $\mathcal{C}$; 2) $\mathcal{A}_{(t,n)}$'s output $\boldsymbol{mpk}^n$ generated by $\mathrm{PKG}_{i,i \in A \cup B}$ are n shares of the mpk chosen by $\mathcal{C}$. Concretely, $\mathcal{S}$ works as:

(1) $\mathcal{S}$ gets mpk from $\mathcal{C}$, by running a setup algorithm with $\mathcal{C}$.

(2) $\mathcal{S}$ runs the $\mathcal{P}^{\sigma}_{\text{rep}}$ protocol (defined in Sect. 2.4) with $\mathcal{A}_{(t,n)}$ by simulating $\text{PKG}_{i,i\in B}$, where $\mathcal{S}$ randomly chooses secret μ_i for $\text{PKG}_{i,i\in B}$. At the end of the $\mathcal{P}^{\sigma}_{\text{rep}}$ protocol execution, $\mathcal{A}_{(t,n)}$ will obtain t shares $\boldsymbol{\sigma}^t$ of σ for $\text{PKG}_{i,i\in A}$. Then $\mathcal{A}_{(t,n)}$ can compute the t master secret key shares $\boldsymbol{s}^t \leftarrow \mathsf{Conv}^{*}_{(t,n)}(\boldsymbol{\sigma}^t, st_0)$, and the t master public key shares $\boldsymbol{mpk}^t \leftarrow [\boldsymbol{s}^t]P_1$. Since the simulator $\mathcal{S}$ chooses secrets for $n-t$ honest PKG nodes randomly, $\mathcal{S}$ can perfectly simulate the view for $\mathcal{A}_{(t,n)}$ in a $\mathcal{P}^{\sigma}_{\text{rep}}$ protocol. After running the $\mathcal{P}^{\sigma}_{\text{rep}}$ protocol with $\mathcal{S}$, $\mathcal{A}_{(t,n)}$ obtains $(\{\boldsymbol{\sigma}^t, \boldsymbol{s}^t\}, \boldsymbol{mpk}^t)$ for $\text{PKG}_{i,i\in A}$. Since (t,n)-DPKG for SM9 requires $n-t \geq t+1$, $\mathcal{S}$ holding $n-t$ shares of σ is able to derive all the outputs of $\mathcal{A}_{(t,n)}$.

(3) $\mathcal{S}$ computes the $\boldsymbol{mpk}^{n-t}$ generated by $\text{PKG}_{i,i\in B}$, by performing Lagrange polynomial interpolation with $\boldsymbol{mpk}^t$ generated by $\text{PKG}_{i,i\in A}$ and mpk. This ensures the n shares $\boldsymbol{mpk}^n$ generated by $\text{PKG}_{i,i\in A\cup B}$ are sharing the secret mpk. Then $\mathcal{S}$ sends $\boldsymbol{mpk}^{n-t}$ and mpk to $\mathcal{A}_{(t,n)}$.

(4) $\mathcal{A}_{(t,n)}$ outputs $(\{\boldsymbol{\sigma}^t, \boldsymbol{s}^t\}, \boldsymbol{mpk}^{n+1})$, where $\boldsymbol{msk}^t$ is denoted as $\{\boldsymbol{\sigma}^t, \boldsymbol{s}^t\}$.

Guaranteed by the perfect security of (t,n)-Shamir secret sharing, $\mathcal{A}_{(t,n)}$ holding only t shares $\boldsymbol{s}^t$ of some master secret key s' (s' is the master secret key determined by $\mathcal{S}$ and $\mathcal{A}_{(t,n)}$ running the $\mathcal{P}^{\sigma}_{\text{rep}}$ protocol), cannot tell if $\boldsymbol{s}^t$ is sharing the secret s' or the secret msk chosen by $\mathcal{C}$. Hence the simulation is correct.

■ **Phase 1.** $\mathcal{A}_{(t,n)}$ adaptively makes private key extraction $\langle \text{ID} \rangle$ queries and decryption $\langle \text{ID}, c \rangle$ queries. For a $\langle \text{ID}, c \rangle$ query, $\mathcal{S}$ passes the query and decrypted c back and forth between $\mathcal{A}_{(t,n)}$ and $\mathcal{C}$. For a $\langle \text{ID} \rangle$ query, $\mathcal{S}$ simulates $\text{PKG}_{i,i\in B}$ running the distributed private key extraction protocol with $\mathcal{A}_{(t,n)}$, and sends $\boldsymbol{S}^{n-t}_{\text{ID}} = \{S^{(i)}_{\text{ID}}\}_{i\in B}$ generated by $\text{PKG}_{i,i\in B}$ to $\mathcal{A}_{(t,n)}$. Concretely, $\mathcal{S}$ works as:

(1) $\mathcal{S}$ gets S_{ID} from $\mathcal{C}$, by forwarding $\mathcal{A}_{(t,n)}$'s $\langle\, \text{ID} \,\rangle$ query to $\mathcal{C}$.

(2) $\mathcal{S}$ runs the DExtract protocol with $\mathcal{A}_{(t,n)}$ by simulating $\text{PKG}_{i,i\in B}$, where $\mathcal{S}$ needs to simulate $z_{i,i\in B}$ for $\mathcal{A}_{(t,n)}$. To simulate $z_{i,i\in B}$, $\mathcal{S}$ first computes $z_{i,i\in A}$. Then $\mathcal{S}$ chooses a random z. Next, $\mathcal{S}$ computes $z_{i,i\in B}$ with $z_{i,i\in A}$ and z, ensuring that the $2t$-privately Shamir shares $\{z_i\}_{i\in A\cup B}$ are sharing the secret z. Finally, $\mathcal{S}$ sends $z_{i,i\in B}$ to $\mathcal{A}_{(t,n)}$.

(3) $\mathcal{S}$ computes $\boldsymbol{S}^{n-t}_{\text{ID}}$ generated by $\text{PKG}_{i,i\in B}$. First, $\mathcal{S}$ computes $\boldsymbol{S}^{t}_{\text{ID}} = \{S^{(i)}_{\text{ID}}\}_{i\in A}$. Then $\mathcal{S}$ computes $\boldsymbol{S}^{n-t}_{\text{ID}} = \{S^{(i)}_{\text{ID}}\}_{i\in B}$, by performing Lagrange polynomial interpolation with $\boldsymbol{S}^{t}_{\text{ID}}$ and the S_{ID}. This ensures the t-privately Shamir shares $\{S^{(i)}_{\text{ID}}\}_{i\in A\cup B}$ are sharing the secret S_{ID} returned by the challenger $\mathcal{C}$.

(4) $\mathcal{S}$ sends $\boldsymbol{S_{\text{ID}}}^{n-t}$ to $\mathcal{A}_{(t,n)}$.

In the simulation, $\boldsymbol{S}^{n}_{\text{ID}} = \{S^{(i)}_{\text{ID}}\}_{i\in A\cup B}$ are random shares to $\mathcal{A}_{(t,n)}$, since z are randomly chosen by $\mathcal{S}$. The view is consistent with what $\mathcal{A}_{(t,n)}$ has seen in a real distributed private key extraction protocol. $\mathcal{A}_{(t,n)}$ holding only t shares of z and without any prior-knowledge of z, views z as completely random distribution (guaranteed by perfect security of (t,n)-Shamir secret sharing), which means

the i_{th} key fragment $S_{\mathrm{ID}}^{(i)} = [1 - F(\mathrm{ID}) \cdot \frac{r_i}{z}]P_2$ is actually a random share to $\mathcal{A}_{(t,n)}$. Thus the simulation is correct.

■ **Challenge.** $\mathcal{A}_{(t,n)}$ sends a tuple $\{m_0, m_1, \mathrm{ID}^*\}$ to $\mathcal{S}$, where m_0 and m_1 are two distinct messages with the same length, an identity ID^* which has never been queried in Phase 1. $\mathcal{S}$ forwards the tuple to $\mathcal{C}$. $\mathcal{C}$ picks a random bit $b \in \{0, 1\}$, and sends c_b^* to $\mathcal{S}$ by computing $c_b^* = \mathsf{Enc}(mpk, \mathrm{ID}^*, m_b)$. $\mathcal{S}$ passes c_b^* to $\mathcal{A}_{(t,n)}$.

■ **Phase 2.** $\mathcal{A}_{(t,n)}$ continues to make private key extraction queries and decryption queries. $\mathcal{S}$ responds just as Phase 1 except for the private key extraction query $\langle \mathrm{ID}^* \rangle$ and the decryption query $\langle \mathrm{ID}^*, c_b^* \rangle$.

■ **Guess.** $\mathcal{A}_{(t,n)}$ outputs a guess $b' \in \{0, 1\}$ of b. Then $\mathcal{S}$ outputs b' as its guess.

Obviously, the advantage of $\mathcal{S}$ is the same as $\mathcal{A}_{(t,n)}$'s, because $\mathcal{A}_{(t,n)}$'s guess is exactly what $\mathcal{S}$ needs to attack the IBE scheme under the single PKG setting:

$$\begin{aligned} Adv_{\mathcal{S}}^{\text{IND-ID-CCA}_{\text{IBE}}} &= 2|Pr[b' = b : \mathcal{S} \rightarrow b'] - \frac{1}{2}| = 2|Pr[b' = b : \mathcal{A}_{(t,n)} \rightarrow b'] - \frac{1}{2}| \\ &= Adv_{\mathcal{A}_{(t,n)}}^{\text{IND-ID-CCA}_{(t,n)\text{-IBE}}} = \epsilon \end{aligned} \tag{1}$$

Since the SM9-IBE scheme under the single PKG setting is IND-ID-CCA secure, we have $\epsilon \leq negl(\lambda)$ for some negligible function $negl(\cdot)$. Combing with (1), we come to the conclusion that the SM9-IBE scheme under the (t, n)-DPKG setting is IND-ID-CCA secure, too.

6 Performance Evaluation

In this section, we first present implementation details of the proposed (t, n)-DPKG. Then we focus on the tradeoff between the system security, robustness and efficiency that inherently existed in the threshold key generation. Finally, we justify the feasibility of integrating (t, n)-DPKG to a deployed system, by comparing a $(2, 6)$-DPKG instance with the centralized private key generation.

A. Implementation Details

We construct our code in C and C++, based on the MIRACL library for elliptic curve cryptography. Benchmark tests are done with google benchmark. We have six PKGs deployed on three Alibaba Cloud simple application servers having 1 CPU core with 2 GB RAM, each running two PKG instances. The round-trip latencies among them are 3 ms∼17 ms. We implement CC as a relatively resource-constrained virtual machine on Virturalbox, which is assigned to only 400 MB RAM, 1 CPU core and the CPU frequency is set to 360 MHz. The operating system for PKG/CC is Ubuntu Server 18.04/14.04 LTS.

B. The Tradeoff Between the Security, Robustness and Efficiency

Although (t, n)-DPKG tackles the inherent *single point of failures* problem of IBC schemes, the system security and robustness come at a price. There is an inherent tradeoff between the system security, robustness and efficiency in (t, n)-DPKG , which can be adjusted via the (t, n)-threshold. Table 2 presents

Table 2. (t, n)-threshold's impact on various overheads

	Communication (sent bytes)	Computation Time	Storage (bytes)
PKG at setup	$32(n-1)\binom{n-1}{t}$	$\binom{n-1}{t}\mathcal{T}_{\mathsf{h}} + \mathcal{T}_{\mathsf{pm}}$	$32\binom{n-1}{t} + n\mathcal{L}^{\mathsf{addr}}$
PKG at extract	$32(n-1)$	$(t-1)\binom{n-1}{t}\mathcal{T}_{\mathsf{h}} + \mathcal{T}_{\mathsf{pm}}$	
CC at setup	0	$(t+1)\mathcal{T}_{\mathsf{pm}}$	$n\mathcal{L}^{\mathsf{addr}}$
CC at extract	$n\mathcal{L}^{\mathsf{id}}$	$(t+1)\mathcal{T}_{\mathsf{pm}}$	

† $\mathcal{L}^{\mathsf{id}}$ and $\mathcal{L}^{\mathsf{addr}}$ are the byte-length of the IoT device's identity and the ip address of the PKG respectively. $\mathcal{T}_{\mathsf{h}}$ and $\mathcal{T}_{\mathsf{pm}}$ stand for the time complexity of hashing and elliptic curve scalar multiplication operation respectively.

how the storage overhead, communication overhead and computation overhead change with the (t, n)-threshold. Due to space limitation, Table 2 only lists the most significant item that affects the overhead. In conjunction with Table 2, we can get the following conclusions:

- Small (t, n)-threshold values are preferable in terms of efficiency, as the overheads on PKG/CC will increase exponentially/linearly with increasing (t, n).
- Large (t, n)-threshold values are preferable in terms of security and robustness. Since in the proposed (t, n)-DPKG scheme for SM9, the security connotation is that *at most t PKGs can be corrupted without exposing the master secret key*; the robustness connotation is that *at most* $n - 2t - 1$ *PKGs can go offline half the way without interrupting the user private key generation service.*
- To strike a good tradeoff between the system security, robustness and efficiency, a recommended range for the number of private key generators n is between 3 and 10. On the one hand, (t, n)-DPKG for SM9 requires $n \geq 2t + 1$ and $t > 0$ which implies $n \geq 3$. On the other hand, according to the experiment we find out that after n has climbed to a certain value (roughly around $n = 10$), a slight increase in t will result in the boom of overheads on the PKG side. One can choose the t value on the need, but increasing t will result in increasing system security while decreasing system robustness, and vice versa.

C. Comparison to the Centralized Private Key Generation

A major concern of the proposed (t, n)-DPKG scheme for SM9 is about efficiency, that is, if (t, n)-DPKG is too slow to be integrated into a deployed system. Indeed, efficiency is a practical concern since the key generation centers may need to generate substantial private keys for distinct IDs. To inspect efficiency, we instantiate a $(2, 6)$-DPKG instance and compare it with the centralized key generation setting. We can tell from the outcome presented in Table 3 that when the (t, n)-threshold is small ($t = 2$ and $n = 6$ in our case), the proposed (t, n)-DPKG for SM9 can easily handle up to 1 million private key generation requests per day (only 21ms is required handling per request in $(2, 6)$-DPKG). For IoT device manufactories equipped with much more productive settings, they can trade efficiency for security and robustness by working with larger (t, n) values.

Table 3. Comparison between the centralized key generation and $(2, 6)$-DPKG

	Communication Traffic	Computation Time		Key Generation Time	Secure	Robust
		on PKG	on CC			
Centralized	43 Bytes	1.54 ms	N/A	1.54 ms	×	×
$(2, 6)$-DPKG	1218 Bytes	2.11 ms	33.49 ms	21.32 ms	√	√

† We assume the IoT device identity's byte-length $\mathcal{L}^{\mathsf{id}} = 10$. The key generation time refers to the time that the PKGs deal with a private key generation request, which excludes the time of combining private key fragments.

7 Conclusion

In this paper, to deal with the master secret key leakage problem in SM9, we propose a (t, n)-threshold Distributed Private Key Generation ((t, n)-DPKG) solution with some techniques from multiparty computation. The proposed scheme achieves better master secret key protection, and doesn't require any modification of original SM9 algorithms after the user private key generation. Besides enhanced security and robustness, we conduct experiments and the results show that with a small (t, n)-threshold value, the proposed scheme can achieve a good balance between the system security, robustness and efficiency.

Acknowledgements. The authors would like to thank the anonymous reviewers for their valuable comments. This work was supported in part by National Natural Science Foundation of China (Nos. 61772520, 61802392, 61972094, 61472416, 61632020), in part by Key Research and Development Project of Zhejiang Province (Nos. 2017C01062, 2020C01078), in part by Beijing Municipal Science and Technology Commission (Project Number Z191100007119007 and Z191100007119002).

References

1. GM/T 0044.1-2016: identity-based cryptographic algorithms SM9-part 1: General. Technical report (2016)
2. GM/T 0044.5-2016: identity-based cryptographic algorithms SM9-part 5: Parameter definition. Technical report (2016)
3. Al-Riyami, S.S., Paterson, K.G.: Certificateless public key cryptography. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 452–473. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40061-5_29
4. Asharov, G., Lindell, Y.: A full proof of the BGW protocol for perfectly secure multiparty computation. J. Cryptology **30**(1), 58–151 (2017)
5. Baek, J., Safavi-Naini, R., Susilo, W.: Certificateless public key encryption without pairing. In: Zhou, J., Lopez, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 134–148. Springer, Heidelberg (2005). https://doi.org/10.1007/11556992_10
6. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Rudnicki, P. (ed.) PODC 1989, pp. 201–209. ACM (1989)

7. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_13
8. Boyen, X.: General *Ad Hoc* encryption from exponent inversion IBE. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 394–411. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_23
9. Chen, L., Harrison, K., Soldera, D., Smart, N.P.: Applications of multiple trust authorities in pairing based cryptosystems. In: Davida, G., Frankel, Y., Rees, O. (eds.) InfraSec 2002. LNCS, vol. 2437, pp. 260–275. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45831-X_18
10. Cheng, Z.: The SM9 cryptographic schemes. IACR Cryptology ePrint Archive 2017, 117 (2017). https://eprint.iacr.org/2017/117.pdf
11. Cheng, Z.: Security analysis of SM9 key agreement and encryption. In: Guo, F., Huang, X., Yung, M. (eds.) Inscrypt 2018. LNCS, vol. 11449, pp. 3–25. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-14234-6_1
12. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 342–362. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_19
13. Gao, W., Wang, G., Wang, X., Chen, K.: Generic construction of certificate-based encryption from certificateless encryption revisited. Comput. J. **58**(10), 2747–2757 (2015)
14. Geisler, M., Smart, N.P.: Distributing the key distribution centre in Sakai–Kasahara based systems. In: Parker, M.G. (ed.) IMACC 2009. LNCS, vol. 5921, pp. 252–262. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10868-6_15
15. Gentry, C.: Certificate-based encryption and the certificate revocation problem. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 272–293. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_17
16. Goldwasser, S., Ben-Or, M., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computing. In: STOC, pp. 1–10 (1988)
17. Lai, J., Kou, W.: Self-generated-certificate public key encryption without pairing. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 476–489. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71677-8_31
18. Lee, B., Boyd, C., Dawson, E., Kim, K., Yang, J., Yoo, S.: Secure key issuing in id-based cryptography. In: Hogan, J.M., Montague, P., Purvis, M.K., Steketee, C. (eds.) ACSW Frontiers 2004, CRPIT, vol. 32, pp. 69–74 (2004)
19. Lindell, Y.: Fast secure two-party ECDSA signing. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 613–644. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_21
20. Long, Y., Xiong, F.: Collaborative generations of SM9 private key and digital signature using homomorphic encryption. In: ICCCS 2020, pp. 76–81. IEEE (2020)
21. Sakai, R., Kasahara, M.: Id based cryptosystems with pairing on elliptic curve. IACR Cryptology ePrint 2003, 54 (2003). https://eprint.iacr.org/2003/054.pdf
22. Smart, N.P.: Cryptography Made Simple. Information Security and Cryptography. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-21936-3
23. Xu, S., Ren, X., Yuan, F., Guo, C., Yang, S.: A secure key issuing scheme of SM9. Comput. Appl. Softw. **37**(01) (2020)