Towards Efficient Fine-Grained Access Control and Data Privacy Protection for Smart Home

Shuaiyong Shen[1(✉)], Yang Yang[1,2], Zuobin Ying[3], and Ximeng Liu[1,2]

[1] College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China
n190320059@fzu.edu.com

[2] Fujian Provincial Key Laboratory of Information Security of Network Systems, Fuzhou University, Fuzhou 350108, China

[3] School of Electrical and Electronic Engineering, Nanyang Technological University, Nanyang 639798, Singapore
james.ying@ntu.edu.sg

Abstract. Fine-grained access control become a research spotlight in the scenario of the smart home with Internet-of-Things (IoTs). However, most of the existing works could only realize ciphertext outsourcing with an unchangeable encryption key in the cloud. That is because the trivial solution needs great communication overhead. To overcome this challenge, we propose an updatable encryption scheme named SM9-UE for the IoT smart home scenario. The scheme in constructed on the basis of SM9, a Chinese official cryptography standard. SM9-UE realizes secure and lightweight ciphertext updating through using the token generated by the data owner, which is information independent with the plaintext. We give the formal security definition and prove it to be IND-UPD secure. Theoretical comparisons demonstrate that our scheme is efficient in terms of computation and storage. Experimental results also indicate that SM9-UE can be practically applied to IoT smart home.

Keywords: Internet of Things · Smart home · Updatable encryption · SM9 · Key rotation

1 Introduction

With the rapid development of modern information technology, Internet-of-Things (IoTs) technology has made our life more convenient and comfortable, such as monitoring and control [15]. It is estimated that more than 50 billion devices will be connected to the Internet by 2020, according to the report [12]. So far, as a response to the trend of our times, various mainstream companies, such as Samsung, Apple and Google [18], has launched their own framework for connecting diverse devices. Smart home is one of the most exciting IoT application scenarios that can revolutionlize our lives. For instance, those devices that

J. Xiong et al. (Eds.): MobiMedia 2021, LNICST 394, pp. 501–515, 2021.
https://doi.org/10.1007/978-3-030-89814-4_36

need to be controlled, such as air conditioners, TVs, doors and windows, can be conveniently closed outdoors by your cell phone.

Although smart home paints a beautiful blueprint for our daily lives, it also brings us serious security problems [13]. There are many ways to attack [11, 21], making the user's private data unsealable, easily leaked or tampered with by illegal elements, resulting in great economic losses to uses. At present, the situation of software access control determines whether users' private information can be safely maintained, many researchers have pointed out that some software can lead to users' data leakage due to over-privileged in the smart home. For instance, a smart doorlock allows the user to lock the door remotely through an App when he forgot to lock the door when leaving home. However, Apps often try to get more privileges than they acturally needs, such as access to the user's location information, access to the user's contact information, access to the battery status of the door lock, and so on. Whereas, users may not intend to grant these privileges at present. [10,19,20].

Both Lee *et al.* [16] and Yan *et al.* [23] have put forward schemes to solve this problem of over-privileged with better protection of user's data privacy. To enable more fine-grained access control, both schemes are functonally-based units of action. Lee *et al.* finish the least privilege and availability of device functionality, it sacrifices time overhead due to the access control list is adopted by access procedure. To overcome these deficiency, Yan *et al.* propose a new scheme with a cryptographic technique. It uses the IBE encryption scheme to encrypt the user data, and takes function ID as the public key to realize secure cloud outsourcing. However, when private data is stored permanently in an untrusted cloud, robust key management should periodically change its own secret key to keep the data secure: updating encrypted data from an old key to a fresh one. For example, the Payment Card Industry Data Security Standard (PCI DSS) emphasizes that credit card data must be stored in an encrypted form requiring key rotation [22]. So, a new scheme, which makes the secret key update periodically, is urgently needed. To address this security issue, we propose an SM9-UE scheme. Our contributions can be summarized as follows:

- ***Fine-grained access control***. The proposed SM9-UE scheme takes the function ID of the device as the basic unit and implements fine-grained access control with solving the over-privilege problem.
- ***Key rotation***. We propose a novel *ciphertext-independent* updatable encryption based on SM9 that can realize rotating the encryption key periodically, as well as alter current ciphertexts from the old key to the fresh one without costly computation and communication overhead.
- ***Security and practicability***. We give the formal security definition and prove the SM9-UE is adaptive update indistinguishablility (IND-UPD) secure. Theoretical analysis and experimental results demonstrate that SM9-UE can meet the security needs in the IoT smart home.

1.1 Organization

The rest of our paper is organized as follows. In Sect. 2, we introduce preliminaries about this work. In Sect. 3, we present our system model and attack model.

Table 1. Summary of notations

Notation	Definition
$x\|\|y$	The concatenation of x and y, where x and y are bit strings or byte strings
$\oplus$	The bitwise XOR operator that operates on two bit strings of the same length
hid	Identifier of the encryption private key generating function
$H_v(\cdot)$	A cryptographic hash function
$H_1(\cdot), H_2(\cdot)$	Cryptographic functions derived from the cryptographic hash function
KDF(·)	The key derivation function
N	The order of the cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, which is a prime number greater than 2^{191}
$BITS(m)$	Count the bit length of a bit string m
Δ_e	An update token under epoch e
$[u]g$	The u multiple of the element g in the additive groups $\mathbb{G}_1$ or $\mathbb{G}_2$
g^u	g to the power of u, where g is an element in the multiplicative group $\mathbb{G}_\mathbb{T}$ and u is a positive integer
l	The length of secret key

Section 4 gives the concrete construction of the SM9-UE scheme. In Sect. 5, we give the formal security definition and prove its security. The performance of the proposed scheme is evaluated in Sect. 6. In Sect. 7, we draw a conclusion remark.

2 Preliminaries

In this section, we outline the definition of Updatable Encryption and SM9 Cryptographic Scheme as well as introduce some of the math involved. We follow the syntax of prior works in [6–8] and summarize the key notations used in this paper in Table 1.

2.1 Bilinear Group

$\mathbb{G}_1, \mathbb{G}_2$ are additive groups and $\mathbb{G}_T$ is a multiplicative group. All three groups have prime order N. g_1 and g_2 are the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. At the same time, there exist a homomorphism ψ from $\mathbb{G}_2$ to $\mathbb{G}_1$ such that $\psi(g_2) = g_1$. Bilinear pairing $\hat{e}$ is a map of $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ satisfying the following conditions:

- **Bilinearity**: for any $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, $a, b \in \mathbb{Z}$, we have $\hat{e}\left(g_1^a, g_2^b\right) = \hat{e}(g_1, g_2)^{ab}$.
- **Non − degeneracy**: $\hat{e}(g_1, g_2) \neq 1_{\mathbb{G}_T}$.
- **Computability**: for any $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, there exists an efficient algorithm to compute $\hat{e}\left(g_1^a, g_2^b\right)$.

2.2 Hard Problem

The security of our updatable encryption based SM9 construction is built upon following hard problems.

Problem 1. (Bilinear Inverse Diffie-Hellman Problem, BIDH [6,9]) For any $a, b \in \mathbb{Z}_N^*$, it is hard to compute the result of $\hat{e}\left(g_1, g_2\right)^{b/a}$.

Problem 2. (Decisional Bilinear Inverse Diffie-Hellman Problem, DBIDH [6,9]) For $a, b, r \in \mathbb{Z}_N^*$, it is very hard to distinguish $\left(g_1, g_2, g_1^a, g_2^b, \hat{e}(g_1, g_2)^{b/a}\right)$ from $\left(g_1, g_2, g_1^a, g_2^b, \hat{e}(g_1, g_2)^r\right)$.

Problem 3. (τ-Bilinear Inverse Diffie-Hellman Problem, τ-BDHI [6,9]) For an integer τ and $\alpha \in \mathbb{Z}_N^*$, given $(g_1, g_2, g_i^{\alpha}, g_i^{(\alpha^2)}, \ldots, g_i^{(\alpha^\tau)})$ for some value $i \in \{1, 2\}$, computing $\hat{e}\left(g_1, g_2\right)^{1/\alpha}$ is hard.

Problem 4. (τ-Gap-Bilinear Inverse Diffie-Hellman Problem, τ-Gap-BIDH [6,9]) For integer τ and $\alpha \in \mathbb{Z}_N^*$, given$(g_1, g_2, g_i^{\alpha}, g_i^{(\alpha^2)}, \ldots, g_i^{(\alpha^\tau)})$ for some value $i \in \{1, 2\}$ and the DBIDH algorithm, it is hard to compute $\hat{e}\left(g_1, g_2\right)^{1/\alpha}$.

2.3 Updatable Encryption

We follow the syntax of prior work [17] and describe the definition of Updatable Encryption (UE).

Definition 1. *An updatable encryption scheme UE for message space $\mathcal{M}$ as a tuple of algorithms* $\{$**UE.Setup**, **UE.Next**, **UE.Enc**, **UE.Dec**, **UE.Upd**$\}$ *as follows.*

- **UE.Setup**: This algorithm takes a security parameter λ as input for getting a secret key k_0.
- **UE.Next**: This algorithm takes a secret key k_e for epoch e as input, it outputs a new secret key k_{e+1} and an update token Δ_{e+1} for each epoch $e+1$.
- **UE.Enc**: According to the input of a message $m \in \mathcal{M}$ and key k_e for epoch e, this algorithm outputs a ciphertext C.
- **UE.Dec**: The deterministic algorithm is also run by client. In terms of input of a ciphertext C_e and k_e of some epoch e return $\{m' / \perp\} \leftarrow$ **UE.Dec** $\left(k_e, C_e\right)$.
- **UE.Upd**: In terms of input of a ciphertext C_e from epoch e and the update token Δ_{e+1}, it returns the updated ciphertext $C_{e+1} \leftarrow$ **UE.Upd**(Δ_{e+1}, C_e).

2.4 SM9 Identity-Based Encryption (SM9-IBE)

SM9 Cryptographic Scheme is a standard for identification and cryptography adopted by China [9]. It is mainly composed of four aspects, namely Digital signature algorithm, Key exchange protocol, Key encapsulation mechanism (KEM), public key encryption algorithm. SM9-IBE is actually a hybrid scheme which consists of KEM and a Symmetric Encryption (SE) algorithm. It contains four operations: **Setup, KeyGen, Encrypt and Decrypt** as follows:

- **Setup**$(\lambda) \rightarrow (MPK, MSK)$. Given the security parameter λ, it outputs a master public key MPK and a master secret key MSK.
- **Extract**$(MSK, ID) \rightarrow d_{ID}$. Given the master secret key MSK and an identity ID, it outputs a personal decryption key d_{ID}.
- **Encrypt**$(MPK, ID, M) \rightarrow C$. Given the master public key MPK and an identity ID, it outputs a ciphertext C.
- **Decrypt**$(d_{ID}, C) \rightarrow M'$. Given a private key d_{ID} for identity ID and ciphertext colllection $C = (C_1, C_2, C_3)$. It outputs a message M'.

3 System Model and Attack Model

we now present some entities involved in our model and the attack model.

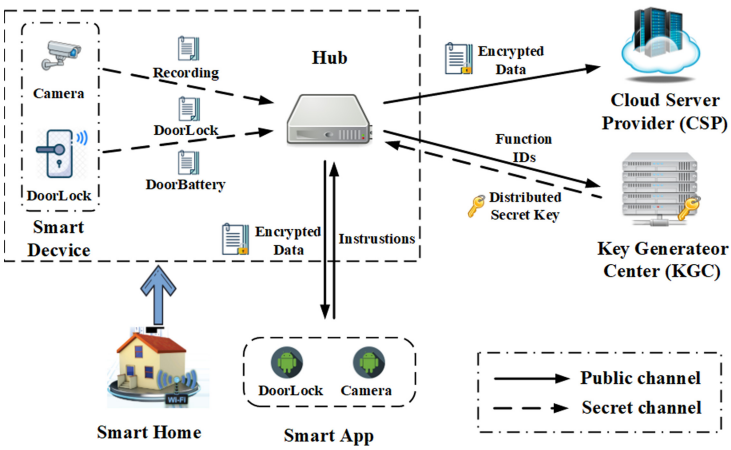

Fig. 1. System model

3.1 System Model

Our system model consists of five entities: Hub, Smart Device, Cloud Service Provider (CSP), Key Generation Center (KGC) and Smart App.

- **Hub** plays an important role since diversity devices may correspond to different communication protocols. It not only helps smart devices connect to the home network, but also serves as a transit point for the data transfer between Smart App and Smart Device. Additionally, this entity replaces the user to encrypt the data generated by the smart device and forward it to the cloud.
- **Smart Device**, such as camera and doorlock, is a data generator that can provide complete and secure data storage on a cloud system through Hub anytime, anywhere. Besides, complex operations is hard to realize on this entity side due to limitations in computing power, storage space, and bandwidth.

- **CSP** is the manager of the cloud server and also an honest but curious third party, which provides quite a few services such as data storage, transmission and timely updating encrypted data after the secret key has been updated.
- **KGC** is assumed to be absolutely trusted third party, who is responsible for the selection of system parameters, generation of the encryption master keys, and generate of user's encryption secret keys according to the its identity. More precisely, it needs to not only protect the secret key but also initializes the relevant system parameters.
- **Smart App**, such as camera app and doorlock app, installed on the user's mobile phone for providing an operation interface of smart devices. If the permissions of Smart App is too large, it is very likely that the sensitive data of the users will be stolen by the malicious app while using it to manage the smart devices.

3.2 Overview

As depicted in Fig. 1, the process of the proposed system model is specialized as follows.

- *System setup and secret key distribution.* At first, KGC selects an appropriate security parameter λ and then takes advantage of Setup algorithm in SM9-UE to generate the system's master public key MPK and master secret key MSK based on this parameter. Then, it generates the corresponding secret keys based on the series of function identity ID, afterwards it receives a certain epoch e sent by Hub and forwards those secret keys back to Hub, starting with $e = 0$. All secrets keys that are distributed must take place on a secure channel.
- *Update key distribution.* When moving from epoch e to epoch $e + 1$, Smart App first destroy by melting or burning the secret key assigned by Hub, Hub will send a series of function identity to KGC, and then KGC will update the corresponding secret key and generates an update token Δ_{e+1} using the KeyUpd algorithm in SM9-UE scheme, and then KGC will send the updated secret key and the corresponding update token to Hub. After Hub receive these, it thoroughly remove all previous secret key and update token.
- *Private the data encryption and upload.* Before storing the user's private data generated by diversity Smart Device on the CSP, Hub collect these data M and run the encrypt algorithm of SM9-UE scheme to encrypt them with the function identity ID and a secret key. Finally, Hub upload these ciphertext with corresponding function identity to the CSP.
- *Secure access to privacy data.* After the Smart App are installed on the phone user must carefully and prudently define the Smart devices function ID that can be accessed in each epoch. Then, Hub will send the corresponding secret key according to the defined function ID to the Smart App. Next, when Smart App needs private data, it will request to the Hub according to the corresponding function ID, Hub will download ciphertext data from CSP based on these function ID, and then send it to the Smart App. Smart App can decrypt it through using the secret key assigned by Hub.

– *Secure ciphertext update.* When moving from epoch e to epoch $e+1$, after CSP received the update token Δ_{e+1} sent by Hub, it first deletes update token Δ_e and then revokes an algorithm CTUpd to update all previously stored ciphertexts.

3.3 Attack Model

Our system will confront the following possible attacks: 1) The app would use malicous logic to obtain privacy while it is installed on the phone; 2) Vulnerable app design flaws that can be exploited by malicious attackers to upgrade their privileges, access unauthorized functions, and steal sensitive data, when they are installed on mobile phones; 3) Vulnerable app may be used by attackers to steal keys, so as to decrypt ciphertext from semi-trusted cloud and obtain sensitive data of user.

4 SM9-UE Scheme

We present a SM9-UE construction scheme, which is *ciphertext-independent* updatable encryption based on SM9-IBE algorithm.
Setup$(\lambda) \rightarrow (MSK, MPK)$. On input security parameter λ, the operation run as follows:

– Select three groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_\mathbb{T}$ of prime order N and a bilinear pairing map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Pick generator $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ randomly.
– Pick a random number $MSK \in \mathbb{Z}_N^*$ as master secret key of system and the master public key MPK can be computed as $MPK \leftarrow [MSK]\ g_1 \in \mathbb{G}_1$.
– Select the function identifier *hid* that use one byte to represent.
– So the public parameters is denoted as $params = \{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathbb{T}, \hat{e}, g_1, g_2, hid, N\}$ and output MSK and MPK.

KeyGen$(MSK, ID) \rightarrow k_0$. To generate a secret key pair k_e for an identity ID, some operation are performed as follows:

– Create $k_{0_1} \leftarrow$ SM9-IBE.Extract (MSK, ID).
– Create $k_{0_2} \leftarrow$ SE.KeyGen (λ).
– It outputs a secret key pair $k_0 \leftarrow (k_{0_1}, k_{0_2})$.

KeyUpd$(k_e) \rightarrow (\Delta_{e+1}, k_{e+1})$. To generate a new secret key pair taking input as a key pair k_e for epoch e, some operation will be executed as follows:

– Parse $k_e = (k_{0_1}, k_{e_2})$.
– Create $k_{e+1_2} \leftarrow$ SE.KeyGen (λ).
– It outputs an update token $\Delta_{e+1} \leftarrow (k_{e_2}, k_{e+1_2})$ and a new secret key pair $k_{e+1} \leftarrow (k_{0_1}, k_{e+1_2})$.

Enc$(MPK, ID, k_{e_2}, m) \rightarrow C$. Given a master public key MPK, an identity ID and a key k_{e_2} as outer key for epoch e, some operation are executed as follows:

- Compute $s = [H_1(ID||hid, N)]\ g_1 + MPK \in \mathbb{G}_1$.
- Pick a random element $r \in Z_N^*$.
- Create $C_1 \leftarrow [r]s$.
- Compute $w = \hat{e}\ (MPK, g_2)^r \in \mathbb{G}_T$.
- If the *sequence cipher* algorithm is used to encrypt data:
  i) Compute $k_1||k_2 = \text{KDF}(C_1||w||ID, l)$ where $l = BITS(m) + v$.
  ii) Create $C_2 \leftarrow k_1 \oplus m$ if k_1 is not all zero, otherwise redo the above.
- If the *block cipher* algorithm is used to encrypt data:
  i) Compute $k_1||k_2 = \text{KDF}(C_1||w||ID, l)$.
  ii) Compute $C_2 = \text{SE.Enc}(k_1, m)$, if k_1 is not all zero, otherwise redo the above.
- It finally outputs $C \leftarrow \text{SE.Enc}(k_{e_2}, C_1||C_2||H_v(C_2||k_2))$.

Dec$(k_e, C_e) \rightarrow m'$. Given a secret key pair k_e and a ciphertext C_e for epoch e, it will carry out some action as follows:

- Parse $k_e = (k_{0_1}, k_{e_2})$.
- $C'_e \leftarrow \text{SE.Dec}(k_{e_2}, C_e)$ and parse $C'_e = (C_1, C_2, C_3)$.
- Output *error* if $C^1 \notin \mathbb{G}_1$, otherwise go on.
- Compute $w' = \hat{e}\ (C_1, k_{0_1})$.
- If the data is encrypted by the *sequence cipher*:
  i) Compute $k_1||k_2 = \text{KDF}(C_1||w'||ID, l)$ where $l' = BITS(M) + v$.
  ii) Create $m' \leftarrow k_1 \oplus C_2$. If k_1 is not all zero, otherwise outputs *error*.
- If the data is encrypted by the *block cipher*:
  i) Compute $k_1||k_2 = \text{KDF}(C_1||w'||ID, l)$.
  ii) Create $m' \leftarrow \text{SE.Dec}(k_1, C_2)$.
- Finally, it outputs m' if $C_3 = H_v(C_2||k_2)$, otherwise outputs *error*.

CTUpd $(\Delta_{e+1}, C_e) \rightarrow C_{e+1}$. Given a update token Δ_{e+1} and ciphertext C_e from epoch e, it returns the updated ciphertext. Some operations will be performed as follows:

- Parse $\Delta_{e+1} = (k_1, k_2)$.
- Compute $C' \leftarrow \text{SE.Dec}(k_1, C_e)$
- It outputs $C_{e+1} \leftarrow \text{SE.Enc}(k_2, C')$.

5 Security Analysis

5.1 Security Model

Next, we homoplastically present the definition of Adaptive Update Indistinguishability (IND-UPD), which defined by *Lehmann et al.* in [17]. The IND-UPD notion ensures that an updated ciphertext obtained from the CTUpd algorithm does not reveal any information about the previous ciphertext, even when $\mathcal{A}$ adaptively compromises a number of keys and tokens before and after the challenge epoch. Then, the security is defined with the following game:

Init: The adversary $\mathcal{A}$ outputs an identity ID^* where it wishes to be challenged.
Setup: The challenger runs the Setup algorithm taking security parameters λ as input. Adversary $\mathcal{A}$ gets all system *params* except master secret key MSK.
Phase 1: The adversary $\mathcal{A}$ adaptively issues queries to following oracles:
- $\mathcal{O}_{\text{Enc}}(m, ID^*)$: The challenger sends a ciphertext C_e for epoch e to $\mathcal{A}$, where C_e is generated with the algorithm Enc as $C_e \leftarrow \text{Enc}(k_e, m)$.
- $\mathcal{O}_{\text{KeyUpd}}(k_e)$: The challenger run KeyUpd algorithm to update secret key k_e as $k_{e+1} \leftarrow \text{KeyUpd}(k_e)$. It output k_{e+1} to $\mathcal{A}$.
- $\mathcal{O}_{\text{CTUpd}}(C_e)$: On input ciphertext C_e for epoch e, run algorithm CTUpd as $C_{e+1} \leftarrow \text{CTUpd}(\Delta_{e+1}, C_e)$ and output a new ciphertext C_{e+1} for epoch $e+1$ to $\mathcal{A}$.

Challenge: Once the adversary $\mathcal{A}$ decides that **Phase 1** is over, it outputs two ciphertext C_0 and C_1 on which it wishes to challenged. The challenger picks a random bit $b \in \{0, 1\}$ and sets the challenge updated ciphertext $\tilde{C} = \text{CTUpd}(C_b)$. It sends $\tilde{C}$ as the challenge to the adversary $\mathcal{A}$.
Phase 2: The adversary $\mathcal{A}$ queries $q_{m+1}, \ldots, q_n$ where query q_i is one of:
- $\mathcal{O}_{\text{Enc}}(m)$: The challenger responds as in **Phase 1**.
- $\mathcal{O}_{\text{KeyUpd}}(k_e)$: The challenger responds as in **Phase 1**.
- $\mathcal{O}_{\text{CTUpd}}(C_e)$: The challenger responds as in **Phase 1**.

Guess: Finally, the adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$. $\mathcal{A}$ wins if $b = b'$.

We refer to such an adversary $\mathcal{A}$ as an IND-UPD adversary. We define the advantage of the adversary $\mathcal{A}$ in attacking the scheme $\mathcal{E}$ as

$$Adv_{\mathcal{E},\mathcal{A}} = \left| \Pr[b = b'] - \frac{1}{2} \right|$$

Definition 2. *We say an updatable encryption scheme $\mathcal{E}$ is* IND-UPD *secure if for all probabilistic polynomial-time adversaries $\mathcal{A}$, it holds that $Adv_{\mathcal{E},\mathcal{A}} < \epsilon(\lambda)$ for some negligible function ϵ.*

5.2 Security Proof

In this section, the security of our proposed scheme will be analyzed precisely under the attack model proposed in Sect. 3.3. In our attack model, **CSP** is considered as an semi-honest entity while **Hub** and **KGC** is accounted as absolutely trusted party. In addition, the **Smart App** may be malicious and attack our system model through the following means to obtain sensitive data of users. We make an assay of how does our scheme defend against these attacks.

Theorem 1. *The proposed* SM9-UE *scheme achieves* IND-UPD *security with respect to Definition 2, if* SM9-IBE *and* SE *scheme is* IND-CPA *secure.*

Proof. Suppose $\mathcal{A}$ has advantage ϵ in attacking the SM9-UE system, then we construct a simulator $\mathcal{B}$ to break the IND-CPA security of the SM9-IBE and SE scheme with a non-negligible advantage. It provides some oracles to $\mathcal{B}$:

- $\mathcal{O}_{\text{Enc}}^{\text{SM9-IBE}}(m, ID_i)$: The challenger run algorithm SM9-IBE.Enc(m, ID_i) and output a ciphertext C to $\mathcal{B}$.
- $\mathcal{O}_{\text{Enc}}^{\text{SE}}(m)$: The challenger run algorithm SE.Enc(m) and output a ciphertext C to $\mathcal{B}$.

The simulator $\mathcal{B}$ works by interacting with the challenger and the adversary $\mathcal{A}$ as follows:

Initialization. The game begins with $\mathcal{A}$ first outputting an identity ID to $\mathcal{B}$. $\mathcal{B}$ initially generates keys $k^i \leftarrow$ SM9-IBE.Extract(ID), sets $e \leftarrow 0$, and guesses the challenge epoch e' uniformly random from $\{0, \ldots \tilde{e}\}$, where $\tilde{e}$ is an upper bound on the number of epochs for $\mathcal{A}$.
Setup. To generate system parameters, challenger run SM9-IBE.Setup algorithm and output $params = \{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathbb{T}}, \hat{e}, g_1, g_2, hid, N\}$ to $\mathcal{B}$.
Phase 1. $\mathcal{A}$ adaptively issues quires to following oracles:
- $\mathcal{O}_{\text{Enc}}(m, ID)$: Compute $C^i \leftarrow$ SM9-IBE.Enc(m, ID), and $\mathcal{B}$ queries C^i to its own oracle $\mathcal{O}_{\text{Enc}}^{\text{SE}}(C^i, k_e^o)$ to obtain a ciphertext C_e. Then $\mathcal{B}$ forwards C_e to $\mathcal{A}$.
- $\mathcal{O}_{\text{KeyUpd}}(k_e)$: Generate key $k_{e+1}^o \leftarrow$ SE.KeyGen(λ) and set $e \leftarrow e + 1$.
- $\mathcal{O}_{\text{CTUpd}}(C_e)$: $\mathcal{B}$ obtains C_{e+1} by querying C^i to its own $\mathcal{O}_{\text{Enc}}^{\text{SE}}(C^i, k_{e+1}^o)$ and return C_{e+1} to $\mathcal{A}$.

Challenge. When $\mathcal{A}$ decides that **Phase 1** is over, it outputs two ciphertext C_0, C_1 for an challenge epoch e'. $\mathcal{B}$ forwards C_0, C_1 to challenger. The challenger pick a random bit $b \in \{0, 1\}$ and set challenge updated ciphertext $\tilde{C} = \text{CTUpd}(C_b)$. Finally $\mathcal{B}$ send it to $\mathcal{A}$.
Phase 2. $\mathcal{A}$ issues more queries to following some oracles:
- $\mathcal{O}_{\text{Enc}}(m_i, ID)$ where $e \neq e'$: $\mathcal{B}$ responds as in **Phase 1**.
- $\mathcal{O}_{\text{KeyUpd}}(k_e)$ where $e \neq e'$: $\mathcal{B}$ responds as in **Phase 1**.
- $\mathcal{O}_{\text{CTUpd}}(C_e)$ where $e \neq e'$: $\mathcal{B}$ responds as in **Phase 1**.

Guess. $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$. $\mathcal{B}$ forward b' to challenger.

As shown above, $\mathcal{A}$ obtain the challenge ciphertext $\tilde{C}$ in IND-CPA game for SE and SM9-UE scheme. If $\mathcal{A}$ successfully guesses which ciphertext is updated in the challenge epoch, $\mathcal{B}$ also outputs the right guess. Therefore, if $\mathcal{A}$ can break proposed SM9-UE scheme with probability $\epsilon(\lambda)$, $\mathcal{B}$ can break SM9-IBE and SE with the same probability. This completes the proof of Theorem 1.

6 Performance Evaluation

In this section, we give thorough and accurate performance analysis.

Table 2. Comparison of computational efficiency

Scheme	Security [4]	Ciphertext independent	Enc	Dec	TokenGen	CTUpd [5]
BLMR [2]	detIND-ENC	✔	2 Exp	2 Exp	2 Exp	$2n$ Exp
BLMR+ [17]	weakIND-UE	✔	2 Exp	2 Exp	2 Exp	$2n$ Exp
RISE [17]	randIND-UE	✔	2 Exp	2 Exp	1 Exp	$2n$ Exp
NYUE [14]	rand-UPD	✔	(60 Exp, 70 Exp)	22 e	2 Exp	(60 Exp, 70 Exp)
E&M [14]	detIND-UPD	✗	3 Exp	3 Exp	3 Exp	3 Exp
ReCrypt [5]	UP-REENC	✗	2 Exp	2 Exp	$2n$ Exp	$2n$ Exp
NYUAE [14]	rand-UPD	✗	(110 Exp, 90 Exp)	29 e	3 Exp	(110 Exp, 90 Exp)
SHINE0 [3]	detIND-UE	✗	1 Exp	1 Exp	1 Exp	1 Exp
mirrorSHINE [3]	detIND-UE	✗	2 Exp	2 Exp	1 Exp	2 Exp
OCBSHINE [3]	detIND-UE	✗	1 Exp	1 Exp	1 Exp	1 Exp
SM9-UE(Ours)	IND-UPD	✔	1 e	1 e	—	—

[4]The notions IND-ENC, randIND-UPD and detIND-UPD are from [17]. The notions UP-IND and UP-REENC are from [3,14,17]. All notions build upon the definitions given by EPRS[4].
[5]Exp and e denote a module exponentiation and a pairing computation, respectively. n denote the number of ciphertexts.

Table 3. Comparison of scheme in IoT

Scheme	Fine-grained access	ACL	Data security	Against over-privilege	Key rotation
FACT [16]	✔	✔	Unsafe	✔	✗
IoT-FBAC [23]	✔	✗	Safe	✔	✗
SM9-UE (Ours)	✔	✗	Safe	✔	✔

6.1 Theoretical Analysis

We analyzed the proposed scheme in theory by comparing it with previous works in terms of communication and storage cost, computation efficiency, which are summarized in Table 2 and Table 3.

As shown in Table 2, we summarize the security and computing efficiency of existing Updatable Encryption (UE) schemes and compare them with ours. Firstly, our scheme is *ciphertext-independent* updatable encryption, which greatly reduces the computation and communication overhead, since the data owner does not download part of the ciphertext from the cloud to make a token. Besides, as our proposed scheme is based on an elliptic curve discrete logarithm problem and bilinear pairings to enhance its security, encryption, and decryption require a pairing computation. In the CTUpd algorithm, the computational cost of our solution up to the SE.Enc algorithm and doesn't need any module exponentiation computation.

In Table 3, we present the comparison between our scheme with other existing IoT schemes from five aspects: fine-grained access, against over-privileged resistance, ACL, data security, and key rotation. To fine-grained access, Lee *et al.*

take the function ID as the object unit to control access and against the drawback of over-privileged in [16]. However, their scheme encountered a bottleneck with the increase in devices due to the introduction of an Access Control List (ACL). Both Yan *et al.* and our solution does not require ACL and safely stored encrypted data on the cloud. Nevertheless, Yan *et al.* is incapable dealing with the problem of secret key rotating except for download all ciphertext from the cloud, re-encrypt it and then upload it, which can incur very high communication and computing overhead. Our solution of the SM9-UE scheme enables the CSP to update the ciphertext independently and securely, which reduces the computational pressure of the data owner and makes the data from the smart device reach safely.

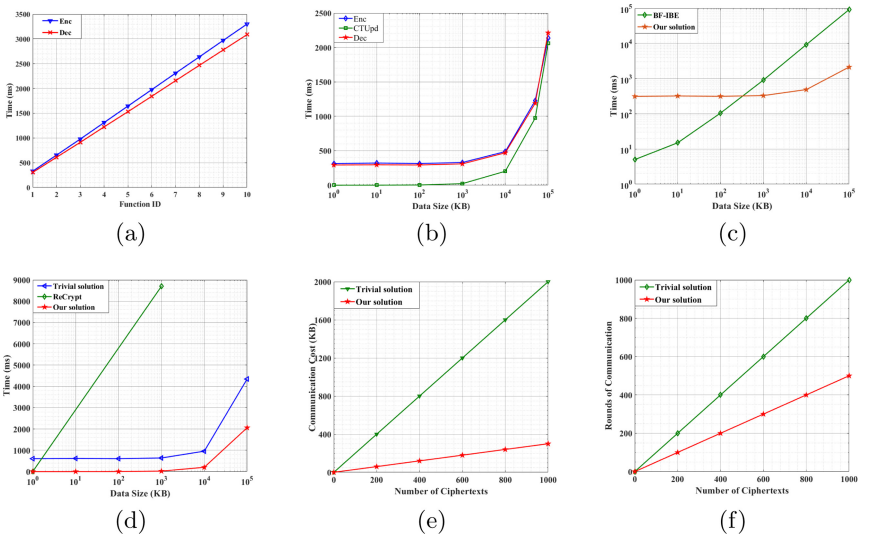

Fig. 2. SM9-UE scheme performance. (Color figure online)

6.2 Implementation and Evaluation

All SM9-UE constructing algorithms are implemented by using C programs. To speed up the simulation we performed, we are based on the the framework proposed by Guan's GmSSL[1] by using a 4.00 GHz Intel®Core™ i5-9500 CPU and 4 GB RAM under the Ubuntu 18.04.5 LTS operating system.

To simulate our scheme, we instantiate and implement the proposed scheme following the standard (Gm/T [9]) and the values of some parameters definition in [8]. The hash function H_v and Symmetric Encryption (SE) is implemented by using SM3 hash algorithm and SM4 block cipher algorithm [8] respectively. The block size is 128 bits. we take the average value of each algorithm executed 100 times as the final result in order to make the experimental results more accurate.

[1] https://github.com/GmSSL.

Time comsumption of Enc and Dec under the different number of function IDs is shown in Fig. 2(a). Let the number of function IDs range from 1 to 10 and the data size is set 1 MB. In the course of encryption phase (the blue line), the greater the number of the function IDs is, the more time that it will cost. When the number of function IDs reaches 10, encryption time only needs about 3.3 s. This demonstrate that our solution is practical and effective. After an App is installed, Hub assigns it the corresponding secret key based on the authorization given by the user. During the decryption phase (the red line), App uses the corresponding secret key to decrypt the ciphertext to obtain data. The decryption time also increases with the number of function IDs. Look at the whole picture, the encryption time is slightly higher than the decryption time and the gap grows as the number of function IDs.

To accurately demonstrate the performance of our scheme, we present the experiment result under different data sizes from 1 KB to 100 MB in Fig. 2(b). In the course of encryption phase, Hub encrypts the data generated by smart device under different function ID. The encryption time is relatively stable around 320 ms and does not float much when the data size is from 1 KB to 1 MB. As data size increases from 1 MB to 10 MB, the encryption time starts to grow slowly. When the data size changes from 1 MB to 100 MB, the encryption time increases significantly. The green line shows how the decryption time varies with different data sizes. This process executed by an App. When an App receives the ciphertext downloaded from the cloud by the Hub, it decrypts ciphertext with the secret key given by the Hub to obtain the data. Judging from the tendency of the decryption time with the size of the data, it is almost the same as the encryption time. The time required for encryption and decryption is actually similar for the same size data. During the ciphertext update phase (the blue line), CSP independently update ciphertext after Hub updates the secret key and receiving the token. It takes very little time to update the ciphertext when the encrypted data size is less than 1 MB. The ciphertext update time increases with the data size from 1 MB and getting closer and will even outlast encryption and decryption time.

In Fig. 2(c), we compare the Enc algorithms between our solution and BF-IBE [1] under different data sizes from 1 KB to 100 MB. Although our scheme performs slightly worse than the another when the data size is less than 1 MB, that is because our scheme requires a pair of computation cost during Enc algorithm. After the data size is super 1 MB, the greater the data size is, the better the performance of our solution than the another.

Next, we show a comparison of ciphertext update with a trivial solution. As we all know, a trivial solution requires downloading all ciphertext from the cloud, re-encrypting it and then upload it for the ciphertext update process. It is amazing that our solution only needs to generate a token locally without other operations. In Fig. 2(d), communication overhead required for data upload and download is ignored by us. It only compared the time required to update the ciphertext algorithm, as the data size gradually increases. For traivial solution (the blue line) in terms of his time with the increase in data size increases and

significantly more than our solution. ReCrypt scheme (the green line) proposed by [5] has obvious advantage than trivial solution under tiny data size. However, as the data size gets larger, more time is required. It is estimated that 10 MB of data will require about 90 ms, which is beyond the scope of our diagram. Figure 2(e) and Fig. 2(f) shows a comparison of communication overhead under ours solution and trivial solution. The data size is set to 1 KB and the number of ciphertext ranges from 0 to 1000. Obviously, the communication overhead and communication rounds of our solution are lower than trivial solution.

7 Conclusion and Further Problem

In this paper, for the sake of preventing software over-privilege access and reducing the computing overhead of each entity accordingly, we propose a solution for updatable encryption based on SM9 in the IoT smart home, which is referred to as SM9-UE. It not only allows users to realize fine-grained access control, but also periodically rotate secret keys to enhance the security of the private data. Besides, when the secret key is rotated, the cloud service provider can independently update ciphertext while ensuring that plaintext is not retrieved. The reason why our way greatly reduces communication and computation overhead is user does not need to download the ciphertext locally and then re-encrypt and upload it. Then, we present a formal security definition and prove that our solution achieves IND-UPD secure. Finally, both the theoretical analysis and the experimental results show that SM9-UE is efficient and practical. Future works includes extending our scheme to other IoTs scenarios such as smart logistics and smart medical, designing new schemes to promote computation efficiency and further security enhancement.

References

1. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. SIAM J. Comput. **32**(3), 586–615 (2003)
2. Boneh, D., Lewi, K., Montgomery, H., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 410–428. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_23
3. Boyd, C., Davies, G.T., Gjøsteen, K., Jiang, Y.: Fast and secure updatable encryption. Technical Representation Cryptology ePrint Archive, Report 2019/1457, 2019. https://eprint.iacr.org (2020)
4. Davidson, A., Deo, A., Lee, E., Martin, K.: Strong post-compromise secure proxy re-encryption. In: Jang-Jaccard, J., Guo, F. (eds.) ACISP 2019. LNCS, vol. 11547, pp. 58–77. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21548-4_4
5. Everspaugh, A., Paterson, K., Ristenpart, T., Scott, S.: Key rotation for authenticated encryption. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 98–129. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_4

6. GM/T: Sm9 identity-based cryptographic algorithms part 1: General. [EB/OL] (2020). http://www.gmbz.org.cn/main/postDetail.html?id=20180322410400
7. GM/T: Sm9 identity-based cryptographic algorithms part 4: Key encapsulation mechanism and public key encryption algorithm. [EB/OL] (2020). http://www.gmbz.org.cn/main/postDetail.html?id=20180322410400
8. GM/T: Sm9 identity-based cryptographic algorithms part 5: Parameter definition. [EB/OL] (2020). http://www.gmbz.org.cn/main/postDetail.html?id=20180322410400
9. GM/T0044-2016: Sm9 identity-based cryptographic algorithms. [EB/OL] (2020). http://www.gmbz.org.cn/main/postDetail.html?id=20180322410400
10. Grace, M.C., Zhou, W., Jiang, X., Sadeghi, A.R.: Unsafe exposure analysis of mobile in-app advertisements. In: Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 101–112 (2012)
11. Huang, Z., Lai, J., Chen, W., Li, T., Xiang, Y.: Data security against receiver corruptions: soa security for receivers from simulatable dems. Inf. Sci. **471**, 201–215 (2019)
12. IDC, I.D.C.: Idc market in a minute: Internet of Things. [EB/OL] (2020). http://www.idc.com/downloads/idc_market_in_a_minute_iot_infographic.pdf
13. Khan, M.A., Salah, K.: Iot security: review, blockchain solutions, and open challenges. Future Generat. Comput. Syst. **82**, 395–411 (2018)
14. Klooß, M., Lehmann, A., Rupp, A.: (R)CCA secure updatable encryption with integrity protection. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 68–99. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_3
15. Lee, I., Lee, K.: The internet of things (iot): applications, investments, and challenges for enterprises. Busin. Horiz. **58**(4), 431–440 (2015)
16. Lee, S., Choi, J., Kim, J., Cho, B., Lee, S., Kim, H., Kim, J.: Fact: Functionality-centric access control system for IOT programming frameworks. In: Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies, pp. 43–54 (2017)
17. Lehmann, A., Tackmann, B.: Updatable encryption with post-compromise security. In: Nielsen, J., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 685–716. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_22
18. Neagle, C.: A guide to the confusing internet of things standards world networkworld. Network World (2014)
19. Pearce, P., Felt, A.P., Nunez, G., Wagner, D.: Addroid: privilege separation for applications and advertisers in android. In: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, pp. 71–72 (2012)
20. Ronen, E., Shamir, A.: Extended functionality attacks on IOT devices: the case of smart lights. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 3–12. IEEE (2016)
21. Stergiou, C., Psannis, K.E., Kim, B.G., Gupta, B.: Secure integration of IOT and cloud computing. Future Gener. Comput. Syst. **78**, 964–975 (2018)
22. v3.2, P.D.: Pci security standards council: requirements and security assessment proce- dures. [EB/OL]. https://www.pcisecuritystandards.org/ Accessed 2020
23. Yan, H., Wang, Y., Jia, C., Li, J., Xiang, Y., Pedrycz, W.: Iot-fbac: function-based access control scheme using identity-based encryption in IOT. Future Gener. Comput. Syst. **95**, 344–353 (2019)