A Redactable Blockchain Framework for Secure Federated Learning in Industrial Internet of Things

Jiannan Wei, *Member, IEEE*, Qinchuan Zhu, Qianmu Li, *Member, IEEE*, Laisen Nie, *Member, IEEE*, Zhangyi Shen, Kim-Kwang Raymond Choo, *Senior Member, IEEE*, and Keping Yu, *Member, IEEE*

***Abstract*—Industrial Internet of Things (IIoT) facilitate private data collecting via (a broad range of) sensors, and the analysis of such data can inform decision making at different levels. Federated learning (FL) can be used to analyze the collected data, in privacy-preserving manner by transmitting model updates instead of private data in IIoT networks. The FL framework is, however, vulnerable because model updates are easily tampered with by malicious agents. Motivated by this observation, we propose a novel chameleon hash scheme with a changeable trapdoor (CHCT) for secure FL in IIoT settings. Our scheme imposes various constraints on the use of trapdoor. We give a rigorous security analysis on our CHCT scheme. We also instantiate the CHCT scheme as a redactable medical blockchain (RMB). The experimental evaluations demonstrate the practical utility of CHCT in terms of accuracy and efficiency.**

***Index Terms*—Blockchain, chameleon hash, federated learning (FL), Industrial Internet of Things (IIoT).**

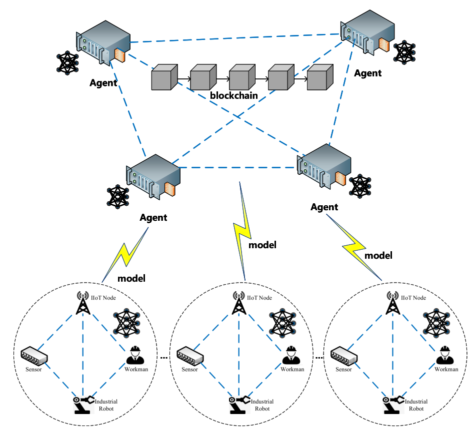


Fig. 1. Overview of distributed architectures for FL in IIoT networks.

I. Introduction

THE RAPID development of the Internet of Things (IoT) technology makes it possible for smart devices to communicate with each other [1]–[3]. Industrial Internet of Things (IIoT) data is of great significance to the management and monitoring of industrial processes, which usually has obvious structural characteristics and has a huge data volume. It is important to store, manage, and analyze these data securely and efficiently since the data generated by IIoT have great value and can be used to extract knowledge. In a traditional IIoT structure [4], [5], a centralized database or cloud server is employed to collect, manage, and analyze all the data. For a long time, the common practice is that factories centrally store the industrial data from IIoT in a database, and the factories are responsible for managing and analyzing the industrial data. This centralized data processing method is conducive to management, but it has problems in data security and privacy protection. Li *et al.* [6] analyzed the drawbacks of this common practice in terms of security privacy protection, and focuses on the necessity of using blockchain to realize the distributed data storage and protection in IIoT networks.

As a learning method for distributed data, federated learning (FL) provides a reliable framework by transmitting model updates instead of industrial data in IIoT networks [7]. Nevertheless, current FL methods are not secure exactly, because these model updates transmitted among agents can be tampered by malicious participants in FL [8]. Blockchain has the nature of decentralization and nonmodification, which can solve this problem to a large extent [9]. Recently, Li *et al.* [10] proposed that blockchain can be used in FL, and the aggregated models can be stored and shared via blockchain. Inspired by [10], we present the distributed architectures for FL in IIoT networks in Fig. 1. In Fig. 1, several IoT agents will participate

Manuscript received 10 December 2021; revised 1 March 2022; accepted 17 March 2022. Date of publication 25 March 2022; date of current version 7 September 2022. This work was supported in part by the 4th Project "Research on the Key Technology of Endogenous Security Switches" under Grant 2020YFB1804604 of the National Key Research and Development Program "New Network Equipment Based on Independent Programmable Chips" under Grant 2020YFB1804600; in part by the National Natural Science Foundation of China under Grant 61702268; in part by the Fundamental Research Fund for the Central Universities under Grant 30918012204 and Grant 30920041112; and in part by the Japan Society for the Promotion of Science (JSPS) Grants-in-Aid for Scientific Research (KAKENHI) under Grant JP18K18044 and Grant JP21K17736. The work of Kim-Kwang Raymond Choo was supported by the Cloud Technology Endowed Professorship. *(Corresponding author: Zhangyi Shen.)*

Jiannan Wei, Qinchuan Zhu, and Qianmu Li are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: jnwei@njust.edu.cn; zhuqinchuan@163.com; qianmu@njust.edu.cn).

Laisen Nie is with the Department of Computer Science, Qingdao Research Institute of Northwestern Polytechnical University, Qingdao 266200, China.

Zhangyi Shen is with the School of Cyberspace, Hangzhou Dianzi University, Hangzhou 310018, China (e-mail: shenzhangyi@hdu.edu.cn).

Kim-Kwang Raymond Choo is with the Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249 USA (e-mail: raymond.choo@fulbrightmail.org).

Keping Yu is with the Global Information and Telecommunication Institute, Waseda University, Shinjuku 169-8050, Japan (e-mail: keping.yu@aoni.waseda.jp).

Digital Object Identifier 10.1109/JIOT.2022.3162499

in the blockchain network and the aggregated models will be stored with the help of blockchain [11]–[14].

In terms of storing data via blockchain, there are two types of solution. One is storing encrypted data in the blockchain directly, the other is using blockchain to assist data storage [15], [16]. Storing data directly in the blockchain is intuitive, which is feasible to realize the decentralized management. However, blockchain is not suitable for storing large amounts of data since it will bring great communication and computation overhead to the blockchain network [17], [18]. When using blockchain to assist data storage, the data are not stored directly in the block and blockchain only records the access link of data [19], [20]. The decentralized and traceable characteristics of blockchain are retained in this type of solution.

Hash technologies such as SHA-256 are used to guarantee the immutability of blockchain. The data in blockchain can not be modified or deleted since blocks are connected by the hash value. Even if the data in a block changes slightly, the original link between blocks will be broken. It is reasonable for the data owners to legally modify or delete their own data while the current blockchain does not support such operation. To solve this problem, Ateniese *et al.* [21] first discussed the feasibility of introducing chameleon hash into the blockchain, specifically using the chameleon hash to replace the traditional hash function. Inspired by [21], numerous chameleon hash schemes [22], [23] were proposed to build the redactable blockchain. These schemes have done great work on how to manage and verify the identity of trapdoor holders. However, few chameleon hash schemes consider the abusement of trapdoor and the potential key exposure problem.

A. Motivation and Our Contribution

Aggregated model in FL may involve the privacy of users, and these data cannot be modified after being recorded in blockchain. To solve this problem, a redactable blockchain can be constructed with the help of the chameleon hash.

Recently, Huang *et al.* [24] proposed a chameleon hash scheme which can achieve the decentralized design of chameleon hash. Each authorized entity only holds a portion of trapdoor and the original trapdoor will be reconstructed with the consent of all authorized entities. The use of trapdoor is restricted to a large degree and this scheme works well between IoT devices. However, it still has some drawbacks. First, authorized entities have the permission to modify the data while they are not the data owner. Second, all authorized entities can use the trapdoor arbitrarily after a round of negotiation, which increases the risk of trapdoor abuse.

Latter, another chameleon hash scheme was proposed in [25], which supports the update of the trapdoor. In [25], a expiration period of trapdoor is chosen and the trapdoor will be updated periodically. However, this expiration period is fixed and the management of trapdoor is not flexible enough. If the data owner changes the trapdoor holder, the trapdoor can still be used by the old trapdoor holder, which is not what we expect. Meanwhile, a key exposure problem exists in this scheme and the chameleon hash collision will enable the adversary to recovery the trapdoor within the expiration period.

Motivated by the above considerations, in this article, we propose a novel chameleon hash scheme with changeable trapdoor (CHCT) to construct a redactable blockchain for secure FL in IIoT. The trapdoor in CHCT will be updated when a pair of collision has been found. The updating process of trapdoor will be supervised without revealing the trapdoor. Below we summarize the contributions of this article.

1) We propose a novel CHCT scheme in which the trapdoor can be abolished at any time. After finding the chameleon hash collision, the trapdoor will be updated to avoid the key exposure problem.
2) We have restricted the use of trapdoor. The data owner can choose the expiration period of trapdoor in a flexible way. Besides, our scheme allows everyone to monitor the use of trapdoor without revealing the trapdoor.
3) We present an application scenario of CHCT and introduce how to a construct redactable medical blockchain (RMB) with the proposed scheme.

B. Related Work

Chameleon hash is a special hash function and the hash collision can be found easily via the trapdoor. The definition and properties of chameleon hash was described clearly in [26]. Latter, Ateniese and De Medeiros [27] extended the work in [26] and proposed the first ID-based chameleon hash scheme together with the first ID-based chameleon signature. Ateniese and De Medeiros [28] introduced the key exposure problem for the first time. Giuseppe pointed that [26] suffers from this problem and this problem was not completely resolved in [27], even though ID-based chameleon hashes was employed.

To solve the key exposure problem, several chameleon hash schemes were proposed in the literature. Chen *et al.* [29] proposed a chameleon hash scheme based on the computation Diffie–Hellman (CDH) problem. Customized identity I was employed in this scheme, which can prevent the adversary from finding collisions for new customized identity I'. Inspired by [29], plenty of chameleon hash schemes were proposed for different applications. Based on the intractability of factoring, a chameleon hash scheme was proposed in [30]. The scheme was proved to enjoy all advantages of the previous work [29]. In [31], some security flaws of [30] were pointed out and an improved chameleon hash scheme without key exposure was proposed. In [32], the first identity-based chameleon hash scheme without key exposure was proposed. Based on the three-trapdoor mechanism, this scheme is proved to achieve all the desired security notions in the random oracle model.

In addition to the chameleon signature, the chameleon hash is also applied in blockchain. Ateniese *et al.* [21] described a modification to the blockchain techniques, which will allow operations such as rewriting blocks. Recently, Huang *et al.* [24] proposed a threshold chameleon hash scheme which achieves the decentralized design of chameleon hash for the first time and is employed to build redactable consortium blockchain (RCB). Different from previous work, the trapdoor is split among authorized entities and is reconstructed using multiparty computation (MPC). Gao *et al.* [33] pointed that the RCB scheme in [24] suffers from a security problem that weakens the crucial redactability. Another chameleon hash scheme ITCH was proposed in [33] to address this threat. A novel chameleon hash scheme which supports the update of trapdoor was proposed in [25]. This scheme employed a fixed expiration time to prevent the abuse of trapdoor and the trapdoor will be updated periodically.

In addition to combining with the blockchain, a chameleon hash can also be applied in the IoT environment. Ramesh and Govindarasu [34] leverage schemes, such as secret sharing, FHE, and chameleon hash functions to tailor a solution that enables long-term privacy-preserving computations for encrypted IoT-device data. Xiang *et al.* [35] presented a lightweight attestation protocol based on an IoT system under an ideal physical unclonable functions environment. With the help of chameleon hash, the scheme has better dynamic adaptability and the destruction of IoT devices will not affect the system Guo *et al.* [36] presented an efficient revocable attribute-based encryption scheme, which employed a chameleon hash to realize revocability. Experiments showed that this scheme satisfies the efficiency requirements of the Internet of Medical Things ecosystem. Peng *et al.* [37] proposed a lightweight data authentication protocol for wireless body area networks (WBANs). Chameleon hash makes that the scheme has minimal computational cost and consumes low power and bandwidth.

C. Article Organization

For the remainder of the article, we give mathematical primitive knowledge in Section II. In Section III, we introduce the whole processing of FL with redactable blockchain, and define the system model, security models of CHCT. Then, we propose our concrete constructions in Section IV. In Section V, we show the security analysis of our scheme. In Section VI, we provide performance evaluation of our proposals. Particularly, we show how to apply it to the medical blockchain in Section VII. Section VIII concludes this article and give the future work.

II. Preliminary

A. Complexity Assumptions

Let G be a cyclic multiplicative group generated by g with the prime order q.

ℓ-Strong Diffie–Hellman (ℓ-SDH) Problem: Given g, g^{α^i} in G for $i = 1, 2, \ldots, \ell$, to compute $g^{\alpha^{\ell+1}}$.

We say $\langle g, g^a, g^b, g^c \rangle$ is a Diffie–Hellman tuple if $c \equiv ab \ mod \ q$ satisfies. ℓ-SDH is the promotion of CDH and its definition can be found in [38].

B. Bilinear Pairing

The symmetric bilinear pairing which satisfies the following.

1) *Bilinearity:* $e(g^a, h^b) = e(g, h^{ab}) = e(g, h)^{ab}$ for all $g, h \in G$ and $a, b \in Z_q^*$.

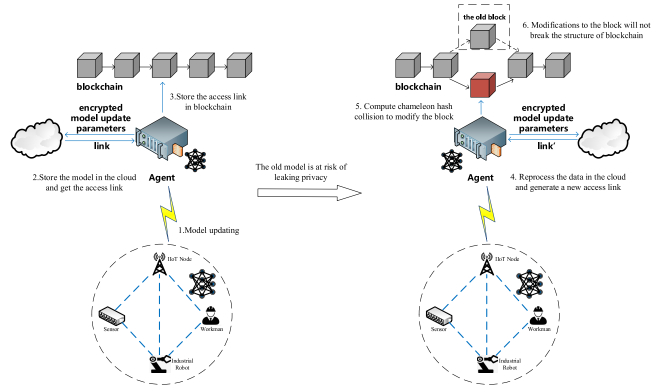


Fig. 2. Whole processing of FL with redactable blockchain.

2) *Nondegeneracy:* There exists $g, h \in G$ such that $e(g, h) \neq 1_{G_T}$.

3) *Computability:* There is an efficient algorithm to compute $e(g, h)$ for all $g, h \in G$.

III. Definition

In this section, we introduce the whole processing of FL with redactable blockchain at first. Since CHCT is the core of redactable blockchain, we will also give the definition of the system model and security models of CHCT.

A. Whole Processing of FL With Redactable Blockchain

Fig. 2 shows the whole processing of FL with redactable blockchain. After obtaining the training model, the agent will store the encrypted model in the cloud and get an access link from the cloud. Then, the agent will broadcast a transaction in the blockchain network and the access link will be stored in blockchain. Once the model has the risk of leaking privacy, the agent will process the data in the cloud and obtain a new access link. To modify the old access link in the blockchain, the agent should calculate a valid chameleon hash collision. With the help of a chameleon hash, the blockchain structure will not be broken.

B. System Model of CHCT

The framework of CHCT contains three major entities, including the data owner P, the data modifier D, and the supervisors. Specifically, we denote the data owner as one who has the ownership of data, denote the data modifier as one who can modify the data with the permission of the data owner, denote the supervisors as one who supervise all the data modification process. P will calculate the initial chameleon hash tuple and send the initial trapdoor to the data modifier D. D will modify the data and calculate the chameleon hash collision with the trapdoor update parameter determined by the supervisors. The supervisors will record the public information and verify each chameleon hash tuple. Besides, the supervisors will verify the identity of P if P wants to revoke the previous authorization to D.

C. Security Models for CHCT

According to [28], a secure chameleon hash scheme should satisfy the following properties.

1) Collision-Resistance: We define the collision-resistance of a chameleon hash scheme via the following security model.

Setup: The challenger $\mathcal{C}$ runs *CHInit* to generate the system public parameter $param_{pub}$. $\mathcal{C}$ chooses message m_0 then runs *CHGen* to generate personal parameter $param_{ID_0}$, chameleon hash tuple CH_0. Finally, $\mathcal{C}$ sends public parameter $param_{pub}$, chameleon hash tuple CH_0 to the adversary $\mathcal{A}$.

Hash Queries: $\mathcal{A}$ can request the hash key h_i, the corresponding chameleon hash $CHash_i$ on any customized identity CID_i and message m_i. $\mathcal{C}$ runs *CHGen* without changing the trapdoor key x_0 to generate h_i and $CHash_i$. Finally, $\mathcal{C}$ returns h_i and $CHash_i$ to $\mathcal{A}$.

Output: Finally, $\mathcal{A}$ finds a pair of chameleon hash collision (CH_u, CH_{u+1}). We describe $CH_i(i = u, u + 1)$ as $CH_i = \langle param_{ID_i}, CHash_i, m_i\rangle$. $\mathcal{A}$ wins the game if:

1) $param_{ID_u} = param_{ID_{u+1}}$;
2) $CHash_u = CHash_{u+1}$, $m_u \neq m_{u+1}$;
3) $CH_i(i = u, u + 1)$ is a valid chameleon hash tuple.

We define the advantage of the adversary as

$$Adv_{\mathcal{A}}(\lambda) = \Pr\big[\mathcal{A} \text{ wins the game}\big].$$

2) Key-Exposure Freeness: The key-exposure freeness of chameleon hash is defined via the following security model.

Setup: The challenger $\mathcal{C}$ runs *CHInit* to generate the system public parameter $param_{pub}$. Then, $\mathcal{C}$ selects constant value mq, which is the maximum times of querying the oracle $\mathcal{O}_{\text{etd\&Forge}}$. Finally, $\mathcal{C}$ sends $param_{pub}$, mq to adversary $\mathcal{A}$.

Queries: In this phrase, $\mathcal{A}$ can query two oracle $\mathcal{O}_{\text{CHIni}}$ and $\mathcal{O}_{\text{etd\&Forge}}$. $\mathcal{A}$ can query $\mathcal{O}_{\text{CHIni}}$ on $\langle x_0, m_0, CID\rangle$, where x_0 is the trapdoor key and *CID* refers to the customized identifier. This oracle returns the corresponding initial chameleon hash tuple CH_0. $\mathcal{A}$ queries $\mathcal{O}_{\text{etd\&Forge}}$ on $\langle m_{u+1}, tv_{u+1}, CID\rangle$ where tv_{u+1} is the trapdoor update parameter. $\mathcal{O}_{\text{etd\&Forge}}$ returns the new chameleon hash tuple CH_{u+1} and the new trapdoor etd_{u+1}.

Output: Finally, the oracle $\mathcal{O}_{\text{etd\&Forge}}$ has been queried mq times. The adversary $\mathcal{A}$ outputs chameleon hash tuple CH_{mq+1} with the trapdoor update parameter tv_{mq+1}. As mentioned above, we will also describe the new chameleon hash tuple CH_{mq+1} as $CH_{mq+1} = \langle param_{ID_p}, CHash_{mq+1}, m_{mq+1}\rangle$. We say $\mathcal{A}$ wins the game if:

1) $m_{mq+1} \notin CH_i, i = 1, 2, \ldots, mq$;
2) CH_{mq+1} is valid, $CHash_{mq+1} = CHash_{mq}$;
3) CH_{mq+1} is the next collision of CH_{mq}.

We define the advantage of the adversary as

$$Adv_{\text{etd\&Forge}} = \Pr\big[\mathcal{A} \text{ wins the game}\big].$$

IV. Proposed CHCT

In this section, we gives the construction of our scheme.

A. Construction

CHInit $(\lambda) \rightarrow param_{pub}$*:* On input the system security parameter λ, choose two groups G, G_T with prime order q and generator g. Let $H : \{0, 1\}^* \rightarrow G$ and $H_1 : \{0, 1\}^* \rightarrow Z_q$ denote cryptographic hash functions. Set the symmetric bilinear map as $e : G \times G \rightarrow G_T$. Set the minimum number l of trapdoor supervisors who will decide whether the trapdoor should be updated. Output the public parameter $param_{pub} = \langle G, G_T, e, g, H, H_1, l\rangle$.

CHGen $(param_{pub}, m_0) \rightarrow CH_0$*:* On input the public parameter $param_{pub}$ and the message m_0, the entity ID_p will run this algorithm and generate the chameleon hash of m_0. The algorithm can be described as follows.

1) Choose two random integer $x_0, \beta \in Z_q^*$, compute a commitment $cmt = H_1(g, g^{x_0}, g^{\beta})$ and $s = \beta - cmt \cdot x_0 \bmod q$, where q is the order of G and x_0 is the trapdoor key of entity ID_p. Then, compute $y_0 = g^{x_0}$.
2) Choose the unique identifier *CID* and compute h as
$$h = H\left(CID \middle\| \left\lceil \frac{ctime()}{h\_valid} \right\rceil\right)$$
where $ctime()$ is a function that returns the timestamp and h_valid is the expiration period chosen by the user. For example, $h\_valid = 86\,400$ means the chameleon hash result of m is valid in 24 h (86 400 s).
3) Choose integer α_0 randomly and compute $R_0 = g^{\alpha_0 x_0}$.
4) Compute the chameleon hash $CHash_0$ for message m_0 as follows:
$$CHash_0 = e\left(y_0, h^{H_1(m_0)}\right) \cdot e\left(g, g^{\alpha_0 x_0}\right).$$
5) Save the parameter s secretly for later use, the algorithm finally outputs the tuple CH_0
$$param_{ID_p} = \langle CID, h\_valid, h, cmt\rangle$$
$$CH_0 = \langle param_{ID_p}, CHash_0, g, y_0, R_0, m_0\rangle.$$

Note that the unique *CID* will be used to identify the chameleon hash tuple CH_i with the same $param_{ID_p}$.

ValidVerify(CH) $\rightarrow$ *Status*: On input the chameleon hash tuple *CH*, this algorithm outputs the current status of *CH* in the following way.

1) Extract h_valid, CID, h from *CH*, use these parameters to recompute h'. If $h' \neq h$, terminate and output $Status = Expired$.
2) Recompute the chameleon hash $CHash'$ using the known parameters. Output $Status = Invalid$ if $CHash' \neq CHash$, else output $Status = Valid$.

EtdShare $(param_{ID_p}, x_0, pubKey) \rightarrow (EtdParam_{enc}, Rec_{CID})$*:* On input personal parameter $param_{ID_p}$ and trapdoor key x_0, the entity ID_p will compute $y_0 = g^{x_0}$ and the ephemeral trapdoor $etd_0 = h^{x_0}$. Then, a constant value $tx \in Z_q^*$ will be chosen for the certain trapdoor holder. The trapdoor parameter can be defined as $EtdParam = \langle etd_0, y_0, tx\rangle$. Then, the entity ID_p encrypts *EtdParam* with the receiver's public key *pubKey* and sends the result $EtdParam_{enc}$ to the receiver. Finally, the entity will broadcast the tuple $Rec_{CID} = \langle param_{ID_P}, h^{tx}, tv\_list, y_0\rangle$. Note that tv_list in Rec_{CID} is the list of the trapdoor update parameters and its initial value is *null*.

PointsGen $(param_{pub}, CID, list, ID_D) \rightarrow Pnts$*:* On input $param_{pub}$, *CID*, a list of entities *list* and identity ID_D who wants to use the trapdoor, this algorithm runs as follows.

1) Extract the parameter l from $param_{pub}$, choose several integers a_i randomly where $i = 0, 1, 2, \ldots, l-1$.
2) Construct a polynomial function $f(n) = a_0 + \sum_{i=1}^{l-1} a_i n^i$.
3) Extract the *list* as $list = \langle ID_1, ID_2, \ldots, ID_{num} \rangle$, where num is the length of *list*. If $num < l$, terminate this algorithm. Choose num different points randomly in the form of $pnt_j = (x_j, f(x_j))$,where $j = 1, 2, \ldots, num$.
4) Encrypt pnt_j with the public key of ID_j, the encrypted result is $Pnt_j = Enc_{ID_j}(pnt_j)$.
5) Broadcast $Pnts = \langle Pnt_1, Pnt_2, \ldots, Pnt_j \rangle$.

Note that if the entity ID_j in *list* allows the trapdoor updating request for CID, ID_j will disclose pnt_j with his signature. A few moments later, some different pnt_j will be broadcasted in the network, which form the collection *pnts*.

EtdConsult$(pnts, param_{pub}, CID) \rightarrow tv_i$: On input the public parameter $param_{pub}$, CID and the collection $pnts = \langle (x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_k, f(x_k)) \rangle$, extract l from $param_{pub}$. If $k < l$, terminate this algorithm. Else, continue this algorithm as follows.

1) Reconstruct the polynomial function $f(n)$ via the collection of *points*.
2) Calculate and output the trapdoor update parameter tv_i for CID as $tv_i = f(0)$.

Note that the new trapdoor etd_{u+1} will be calculated as $etd_{u+1} = {etd_u}^{tv_{u+1} \cdot tx}$, disclosing tv_i will not leak the new trapdoor due to the confidentiality of tx.

CollisionFind $(CH_u, etd_u, m_{u+1}, tv_{u+1}, tx) \rightarrow CH_{u+1}$: To facilitate the description of the algorithm, we also denote $CH_i = \langle param_{ID_p}, CHash_i, g, y_i, R_i, m_i \rangle$, $i = u, u+1$. On input the valid chameleon hash tuple CH_u, the latest trapdoor etd_u, another message m_{u+1}, the trapdoor update parameter tv_{u+1}, and the constant value tx, this algorithm will find collision for CH_u in the following way.

1) Compute $tw_{u+1} = tv_{u+1} \cdot tx$.
2) Calculate and save the latest ephemeral trapdoor etd_{u+1} as $etd_{u+1} = {etd_u}^{tw_{u+1}} = {etd_u}^{tv_{u+1} \cdot tx}$.
3) Extract $R_u = g^{\alpha_u x_u}$, m_u, y_u, h from CH_u. Calculate $y_{u+1} = y_u^{tw_{u+1}}$ and R_{u+1} as

$$\begin{aligned} R_{u+1} &= g^{\alpha_{u+1} x_{u+1}} \\ &= \frac{etd_u^{H_1(m_u)} (g^{\alpha_u x_u})}{etd_{u+1}^{H_1(m_{u+1})}} = \frac{etd_u^{H_1(m_u)} (g^{\alpha_u x_u})}{etd_u^{H_1(m_{u+1}) tv_{u+1} tx}} \\ &= g^{\alpha_u x_u} etd_u^{H_1(m_u) - H_1(m_{u+1}) tv_{u+1} tx}. \end{aligned}$$

4) Reconstruct and output the tuple CH_{u+1} for m_{u+1} by replacing y_u, R_u, m_u in CH_u with $y_{u+1}, R_{u+1}, m_{u+1}$.

CollisionVerify$(CH_u, CH_{u+1}, Rec_{CID}, tv_{u+1}) \rightarrow Flag$: On input two chameleon hash tuples and the corresponding trapdoor update parameter tv_{u+1}, this algorithm verifies whether these tuples forms chameleon hash collision under the same CID. Note that $CH_i = \langle param_{ID_p}, CHash_i, g, y_i, R_i, m_i \rangle$, $i = u, u+1$ and CH_u is a valid chameleon hash tuple verified before. This algorithm runs as follows.

1) Retrieve the record Rec_{CID}, extract h^{tx} and the latest parameter y_u.
2) Check if the equation $e(y_u, (h^{tx})^{tv_{u+1}}) = e(y_{u+1}, h)$ holds, if not, terminate and output $Flag = False$.
3) Run $ValidVerify(CH_{u+1})$. This algorithm will terminate and output $Flag = False$ if CH_{u+1} is invalid or $CHash_{u+1} \neq CHash_u$.
4) This algorithm outputs $Flag = True$ since CH_{u+1} is a valid chameleon hash collision of CH_u. Then the last two parameters in Rec_{CID} will be updated, which means tv_{u+1} will be added to the list *tv_list* and y_{u+1} will replace y_u.

EtdAbolish$(CID, tx) \rightarrow (CH_{u+1}, h^{tx'})$: This algorithm will abolish the old trapdoor and choose a new trapdoor. On input CID, the entity ID_p can get the latest Rec_{CID}, CH_u

$$\begin{aligned} CH_u &= \left\langle param_{ID_p}, CHash_u, g, y_u, R_u, m_u \right\rangle \\ Rec_{CID} &= \left\langle param_{ID_p}, h^{tx}, tv\_list, y_u \right\rangle. \end{aligned}$$

This algorithm runs as follows.

1) Extract $h, y_u, R_u = g^{\alpha_u x_u}$ from CH_u. Then, retrieve all the trapdoor update parameters $tv_1, tv_2, \ldots, tv_u$ from *tv_list* in Rec_{CID}.
2) Calculate the latest ephemeral trapdoor $etd_u = h^{x_u}$, where tx is known to ID_p and the trapdoor key can be calculated as $x_u = x_0 \cdot tx^u \prod_{i=1}^{u} tv_i$.
3) Choose an integer x_{u+1} randomly and calculate these parameters

$$\begin{aligned} y_{u+1} &= g^{x_{u+1}} \\ g^{\alpha_{u+1} x_{u+1}} &= h^{(x_u - x_{u+1}) H_1(m_u)} g^{\alpha_u x_u}. \end{aligned}$$

4) Similar to the process of *CHGen*, choose a new parameter β' and calculate the new commitment $cmt' = H_1(g, g^{x^{u+1}}, \beta')$ for verifying the identity of ID_p in the future. Then, construct $R_{u+1} = g^{\alpha_{u+1} x_{u+1}}$ and replace the parameters cmt, y_u, and R_u in CH_u with cmt', y_{u+1}, and R_{u+1} to get CH_{u+1}.
5) Choose another constant value $tx' \in Z_q^*$, compute $h^{tx'}$. Output CH_{u+1} and $h^{tx'}$.

CmtOpen$(ID_p, s, y_0, tx, CID, CH_{u+1}, h^{tx'}) \rightarrow Status$: This algorithm will open the initial commitment cmt and verify the identity of ID_p. After that, the old chameleon hash tuple CH_u will be replaced by CH_{u+1}. Search by CID, every entity can get the current Rec_{CID}, CH_u

$$\begin{aligned} CH_u &= \left\langle param_{ID_p}, CHash_u, g, y_u, R_u, m_u \right\rangle \\ Rec_{CID} &= \left\langle param_{ID_p}, h^{tx}, tv\_list, y_u \right\rangle. \end{aligned}$$

This algorithm runs as follows.

1) Extract cmt from $param_{ID_p}$ in CH_u. Given the parameters s, y_0, tx, the entity ID_p has the right to abolish the old trapdoor and the algorithm goes on if the following two equations both satisfy:

$$y_u = (y_0)^{tx^u \prod_{i=1}^{u} tv_i}, cmt = H_1\left(g, y_0, g^s y_0^{cmt}\right).$$

Otherwise, terminate the algorithm and output $Status = No\ Right$.

2) Run $ValidVerify(CH_{u+1})$. This algorithm will terminate and output $Status = Invalid\ tuple$ if CH_{u+1} is invalid or $CHash_{u+1} \neq CHash_u$.

3) Update the tuple Rec_{CID}. First, set $tv\_list = null$. Second, extract y_{u+1} from CH_{u+1} and replace the last parameter of Rec_{CID}. Third, update parameter $h^{tx'}$. Finally, Output $Status = Success$.

V. Security Analysis of CHCT

A. Correctness

1) CollisionFind: The correctness of this algorithm means that the trapdoor holder can construct new tuple CH_{u+1} without changing the chameleon hash result $CHash$. Given that $etd_i = h^{x_i}$, $R_{u+1} = g^{\alpha_{u+1}x_{u+1}}$, and $x_{u+1} = x_u \cdot tv_{u+1} \cdot tx$, we can prove $CHash_{u+1} = CHash_u$ as follows:

$$
\begin{aligned}
&CHash_{u+1}\\
&= e\Big(y_{u+1}, h^{H_1(m_{u+1})}\Big) \cdot e(g, R_{u+1}) && (1)\\
&= e\Big(g^{x_{u+1}}, h^{H_1(m_{u+1})}\Big) \cdot e\left(g, \frac{etd_u^{H_1(m_u)} g^{\alpha_u x_u}}{etd_{u+1}^{H_1(m_{u+1})}}\right) && (2)\\
&= e\Big(g^{x_{u+1}}, h^{H_1(m_{u+1})}\Big) \cdot e\left(g, \frac{h^{x_u H_1(m_u)} g^{\alpha_u x_u}}{h^{x_{u+1} H_1(m_{u+1})}}\right) && (3)\\
&= e\Big(g, h^{x_{u+1}H_1(m_{u+1})}\Big) \cdot e\big(g, g^{\alpha_u x_u}\big)\\
&\quad \cdot e\Big(g, h^{x_u H_1(m_u) - x_{u+1}H_1(m_{u+1})}\Big) && (4)\\
&= e\Big(g, h^{x_u H_1(m_u)}\Big) \cdot e\big(g, g^{\alpha_u x_u}\big) && (5)\\
&= e\Big(g^{x_u}, h^{H_1(m_u)}\Big) \cdot e\big(g, g^{\alpha_u x_u}\big) && (6)\\
&= e\Big(y_u, h^{H_1(m_u)}\Big) \cdot e\big(g, g^{\alpha_u x_u}\big) = CHash_u. && (7)
\end{aligned}
$$

2) EtdAbolish: The correctness of this algorithm means that the CH_{u+1} constructed from CH_u is a valid tuple. Given that $m_{u+1} = m_u, y_{u+1} \neq y_u$, We can prove $CHash_{u+1} = CHash_u$ as follows:

$$
\begin{aligned}
&CHash_{u+1}\\
&= e\Big(y_{u+1}, h^{H_1(m_{u+1})}\Big) \cdot e\big(g, g^{\alpha_{u+1}x_{u+1}}\big) && (8)\\
&= e\Big(g^{x_{u+1}}, h^{H_1(m_u)}\Big) \cdot e\Big(g, h^{(x_u - x_{u+1})H_1(m_u)} g^{\alpha_u x_u}\Big) && (9)\\
&= e\Big(g, h^{x_{u+1}H_1(m_u)}\Big) \cdot e\Big(g, h^{(x_u - x_{u+1})H_1(m_u)}\Big) \cdot e\big(g, g^{\alpha_u x_u}\big) && (10)\\
&= e\Big(g^{x_u}, h^{H_1(m_u)}\Big) \cdot e\big(g, g^{\alpha_u x_u}\big) = CHash_u. && (11)
\end{aligned}
$$

B. Collision Resistance

Theorem 1: If there exists an adversary $\mathcal{A}$ which can break the proposed chameleon hash scheme, then we can construct another adversary $\mathcal{B}$ who can use $\mathcal{A}$ to solve the CDH problem.

Proof: Giving a CDH problem instance (g, g^a, g^b), we will construct $\mathcal{B}$ who can use $\mathcal{A}$ to compute g^{ab}. We construct $\mathcal{B}$ as follows.

Setup: $\mathcal{B}$ runs $CHInit$ to generate the public parameter $param_{pub} = \langle G, G_T, e, g, H, H_1, l\rangle$. $\mathcal{B}$ chooses message m_0, then runs $CHGen$ to generate personal parameter $param_{ID_0}$ and chameleon hash tuple $CH_0 = \langle param_{ID_0}, CHash_0, g, y_0, R_0, m_0\rangle$. Note that $\mathcal{B}$ sets $x_0 = a$ without knowing the value of a. In this way, $y_0 = g^a$, $R_0 = (g^a)^\alpha$ where α is a random integer. $\mathcal{B}$ sends $param_{pub}$ and CH_0 to $\mathcal{A}$.

Hash Query: The adversary $\mathcal{A}$ queries hash key and the corresponding chameleon hash $CHash_i$ on any customized identity CID_i, message m_i in this phase. At first, $\mathcal{B}$ chooses customized identity CID^* and calculates $TValid^* = \lceil [ctime()/h\_valid] \rceil$. $\mathcal{B}$ prepares a hash list $\mathcal{L}$ to record all queries and responds, where the hash list is empty at the beginning.

For each query on CID and m, $\mathcal{B}$ calculates the parameter $TValid = \lceil [ctime()/h\_valid] \rceil$. Then $\mathcal{B}$ searches the tuple $\langle CID, TValid, \epsilon, Q\rangle$ in hash list $\mathcal{L}$, where ϵ is a random integer and Q refers to the response of the hash key.

$\mathcal{B}$ returns $Q, CHash$ to $\mathcal{A}$ as follows.

1) If the tuple exists in the hash list $\mathcal{L}$, $\mathcal{B}$ gets Q from $\mathcal{L}$ directly. Otherwise, $\mathcal{B}$ randomly chooses integer ϵ, then computes the hash key Q as follows:
$$Q = \begin{cases} g^b \cdot g^\epsilon, & CID = CID^*, TValid = TValid^* \\ g^\epsilon, & \text{otherwise.} \end{cases}$$
After that, the tuple $\langle CID, TValid, \epsilon, Q\rangle$ will be added to the hash list $\mathcal{L}$.
2) $\mathcal{B}$ randomly chooses an integer α and computes the chameleon hash $CHash$ as follows:
$$CHash = e\Big(y_0, Q^{H_1(m)}\Big) e\Big(g, \big(g^a\big)^\alpha\Big).$$
3) $\mathcal{B}$ returns $Q, CHash$ to $\mathcal{A}$.

Output: Finally, $\mathcal{A}$ finds a pair of chameleon hash collision (CH_u, CH_{u+1}) under the same CID and x_0. This chameleon hash collision means that:

1) $CHash_u = CHash_{u+1}$, $m_u \neq m_{u+1}$;
2) $CH_i (i = u, u+1)$ is a valid chameleon hash tuple.

If $CID \neq CID^*$ or $TValid \neq TValid^*$, aborts and terminates. In this case, $\mathcal{A}$ will finally find a valid collision under $(CID^*, TValid^*)$, which means $\mathcal{B}$ can use the useful attack to solve the CDH problem in the following way.

$\mathcal{B}$ searches $\mathcal{L}$ to get the tuple $(CID^*, TValid^*, \epsilon^*, Q^*)$. Then, $\mathcal{B}$ can deduce $h = H_1(CID^* || TValid^*) = Q^* = g^b \cdot g^{\epsilon^*}$. We have the following equation:

$$CHash_u = CHash_{u+1} = CHash \quad (12)$$
$$CHash_u = e\Big(y_0, h^{H_1(m_u)}\Big) e\big(g, g^{\alpha_u x_0}\big) \quad (13)$$
$$CHash_{u+1} = e\Big(y_0, h^{H_1(m_{u+1})}\Big) e\big(g, g^{\alpha_{u+1} x_0}\big). \quad (14)$$

Since g is the generator of group G and h belongs to G, h can be written as $h = g^V$ where V is an integer. Then, we can use the property of bilinear pairing to get

$$
\begin{aligned}
CHash &= e\Big(y_0, h^{H_1(m_i)}\Big) e\big(g, g^{\alpha_i x_0}\big)\\
&= e\Big(g^{x_0}, g^{V H_1(m_i)}\Big) e\big(g, g^{\alpha_i x_0}\big) && (15)\\
&= e(g, g)^{x_0 V H_1(m_i)} e(g, g)^{\alpha_i x_0} && (16)\\
&= e(g, g)^{x_0 V H_1(m_i) + \alpha_i x_0}, i = u, u+1. && (17)
\end{aligned}
$$

From (17) we can deduce that

$$V H_1(m_u) + \alpha_u = V H_1(m_{u+1}) + \alpha_{u+1} \quad (18)$$

$$V = \frac{\alpha_{u+1} - \alpha_u}{H_1(m_u) - H_1(m_{u+1})} \tag{19}$$

$$h = g^V = g^{\frac{\alpha_{u+1} - \alpha_u}{H_1(m_u) - H_1(m_{u+1})}}. \tag{20}$$

Future, we can compute

$$h^x = \left(\frac{g^{\alpha_{u+1}x}}{g^{\alpha_u x}}\right)^{\frac{1}{H_1(m_u) - H_1(m_{u+1})}}. \tag{21}$$

Although $\mathcal{B}$ dose not know what x or a exactly is, but he knows that $x = a$ and $h = g^b \cdot g^{\epsilon^*}$. So $\mathcal{B}$ can compute

$$\left(\frac{g^{\alpha_{u+1}x}}{g^{\alpha_u x}}\right)^{\frac{1}{H_1(m_u) - H_1(m_{u+1})}} = h^x = g^{ab} \cdot g^{a\epsilon^*} \tag{22}$$

$$\frac{\left(\frac{g^{\alpha_{u+1}x}}{g^{\alpha_u x}}\right)^{\frac{1}{H_1(m_u) - H_1(m_{u+1})}}}{(g^a)^{\epsilon^*}} = g^{ab}. \tag{23}$$

Note that in (23), $g^{\alpha_{u+1}x} = R_{u+1}$, $g^{\alpha_u x} = R_u$ and the values of $R_{u+1}, R_u, H_1(m_u), H_1(m_{u+1})$ can be extracted or calculated from the collision (CH_u, CH_{u+1}), and $\mathcal{B}$ can solve the CDH problem with the help of $\mathcal{A}'s$ useful attack.

In *HashQuery*, the answer from $\mathcal{B}$ is indistinguishable from the point view of $\mathcal{A}$ since each answer contains the random integer ϵ. After querying q_H times, $\mathcal{A}$ has the probability $(1/q_H)$ to guesses the *CID* correctly. Then, the probability of $\mathcal{A}'s$ useful attack is $(1/q_H)$. Suppose the adversary without knowing the trapdoor can find a chameleon hash collision with non-negligible probability ε. Hence, $\mathcal{B}$ has the nonnegative probability (ε/q_H) to solve the CDH problem. ■

C. Key-Exposure Freeness

Theorem 2: If there exists an adversary $\mathcal{A}$ which can break the proposed chameleon hash scheme, then we can construct another adversary $\mathcal{B}$ who can use $\mathcal{A}$ to solve the ℓ-SDH problem.

Proof: Giving a ℓ-SDH problem instance $(g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^l})$, $\mathcal{B}$ controls the oracle $\mathcal{O}_{\text{CHIni}}, \mathcal{O}_{\text{etd\&Forge}}$. We will construct $\mathcal{B}$ who can use $\mathcal{A}$ to compute $g^{\alpha^{l+1}}$. We construct $\mathcal{B}$ as follows:

Setup: $\mathcal{B}$ runs *CHInit* to generate the public parameter $param_{pub}$. $\mathcal{B}$ holds a ℓ-SDH tuple $\langle g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^l}\rangle$ and sets the maximum query times $mq = l$. Finally, $\mathcal{B}$ sends $param_{pub}$ and mq to $\mathcal{A}$.

Query $\mathcal{O}_{CHIni}$: First, $\mathcal{B}$ constructs a list CH_0List and chooses customized identity CID^*. If $\langle x_0, m_0, CID\rangle$ has been queried before, $\mathcal{B}$ returns the corresponding result in CH_0List. Second, for a new query $\langle x_0, m_0, CID\rangle$, $\mathcal{B}$ updates CH_0List in two ways. If $CID \neq CID^*$, $\mathcal{B}$ queries $\mathcal{O}_{\text{CHIni}}$ to get a chameleon hash tuple CH_0. $\mathcal{B}$ can extract h from CH_0. Then, $\mathcal{B}$ chooses tx randomly and computes h^{tx}. If $CID = CID^*$, $\mathcal{B}$ chooses an integer w randomly and calculates $h = g^w$. Then, $\mathcal{B}$ runs *CHGen* with the chosen parameters x_0, h, CID to obtain CH_0. $\mathcal{B}$ sets $tx = \alpha$ and calculates $h^{tx} = (g^w)^{\alpha} = (g^{\alpha})^w$. Third, $\mathcal{B}$ adds the tuple $\langle CH_0, CID, tx, x_0, h^{tx}\rangle$ to CH_0List. Note that tx will be set *null* if $CID = CID^*$. Finally, $\mathcal{B}$ sends CH_0 to $\mathcal{A}$.

Query $\mathcal{O}_{etd\&Forge}$: First, $\mathcal{B}$ constructs a list *CHList* to record $\langle m_i, etd_i, CID, tv_i, curtTimes\rangle$. In *CHList*, etd_i indicates the current trapdoor and *curtTimes* means the given *CID* has been queried *curtTimes* times. Second, for a new query $\langle m_{u+1}, tv_{u+1}\rangle$, $\mathcal{B}$ updates *CHList*. Given a *CID*, if *CID* does not exist in CH_0List or has been queried mq times, the oracle terminates; If $CID \neq CID^*$, $\mathcal{B}$ retrieves the trapdoor information, the latest chameleon hash tuple from CH_0List and *CHList*. Then, $\mathcal{B}$ runs *CollisionFind* to construct a chameleon hash collision with the given m_{u+1}, tv_{u+1}, CID. If $CID = CID^*$, parameter tx will be set *null* in CH_0List and parameters y_{u+1}, etd_{u+1} in CH_{u+1} will be updated in this way

$$\begin{aligned} y_{u+1} &= g^{x_{u+1}} = y_u^{tv_{u+1}tx} = g^{x_u tv_{u+1} tx} \\ &= g^{x_0 \prod_{i=1}^{u}(tv_i tx)} = \left(g^{tx^u}\right)^{x_0 \prod_{i=1}^{u} tv_i} \\ &= \left(g^{\alpha^u}\right)^{x_0 \prod_{i=1}^{u} tv_i} \end{aligned} \tag{24}$$

$$\begin{aligned} etd_{u+1} &= h^{x_{u+1}} = etd_u^{tv_{u+1}tx} = h^{x_u tv_{u+1} tx} \\ &= h^{x_0 \prod_{i=1}^{u}(tv_i tx)} = \left(h^{tx^u}\right)^{x_0 \prod_{i=1}^{u} tv_i} \\ &= \left(g^{\alpha^u}\right)^{w x_0 \prod_{i=1}^{u} tv_i}. \end{aligned} \tag{25}$$

Third, add $\langle m_{u+1}, etd_{u+1}, CID, tv_{u+1}, curtTimes + 1\rangle$ to *CHList*. Finally, $\mathcal{B}$ returns CH_{u+1}, etd_{u+1} to $\mathcal{A}$.

Output: After querying $\mathcal{O}_{\text{etd\&Forge}}$ mq times, $\mathcal{A}$ outputs a valid chameleon hash collision CH_{mq+1} with the trapdoor update parameter tv_{mq+1}. We describe CH_{mq+1} as $CH_{mq+1} = \langle param_{ID_p}, CHash_{mq+1}, g, y_{mq+1}, R_{mq+1}, m_{mq+1}\rangle$. If the parameter *CID* in CH_{mq+1} is not CID^*, abolish and terminate. In this way, the customized identity in CH_{mq+1} will be CID^*. Given the tuple $\langle g, g^{\alpha^i}\rangle, i = 1, 2, \ldots, l$, $\mathcal{B}$ can solve the ℓ-SDH problem with the help of $\mathcal{A}$. $\mathcal{B}$ knows $tx = \alpha$ without knowing the value of α. After searching $CH0List$ and *CHList*, $\mathcal{B}$ can obtain all the trapdoor update parameter tv_i, parameter w, x_0. Using CH_{mq+1}, $\mathcal{B}$ can calculate

$$etd_{l+1}^{H_1(m_{l+1})} = \frac{etd_l^{H_1(m_l)}(g^{\alpha_l x_l})}{R_{l+1}} \tag{26}$$

$$= h^{x_{l+1}H_1(m_{l+1})} = h^{tx^{l+1}H_1(m_{l+1})x_0 \prod_{i=1}^{l+1} tv_i} \tag{27}$$

$$= \left(g^{tx^{l+1}}\right)^{wH_1(m_{l+1})x_0 \prod_{i=1}^{l+1} tv_i}. \tag{28}$$

Since $tx = \alpha$, $\mathcal{B}$ can deduce

$$g^{\alpha^{l+1}} = \left(\frac{etd_l^{H_1(m_l)}(g^{\alpha_l x_l})}{R_{l+1}}\right)^{\frac{1}{wH_1(m_{l+1})x_0 \prod_{i=1}^{l+1} tv_i}}. \tag{29}$$

Note that in (29), $g^{\alpha_l x_l} = R_l$. $\mathcal{B}$ can solve the l-SDH problem with the help of $\mathcal{A}'s$ useful attack since all variables on the right side of (29) are known.

When $\mathcal{A}$ queries $\mathcal{O}_{\text{CHIni}}$, it makes no difference to obtain h^{tx} from random integer tx or g^{α}. Assuming that $\mathcal{A}$ queried n different customized identities in total, $\mathcal{A}$ will guess the correct CID^* with probability $(1/n)$. If $\mathcal{A}$ has non-negligible probability ε to construct new chameleon hash tuple under the certain *CID*, $\mathcal{B}$ has the nonnegative probability (ε/n) to solve ℓ-SDH problem. ■

TABLE I
COMPLEXITY OF DIFFERENT ALGORITHMS

Schemes	Algorithms				
	GenCH0	GenEtd	FindCollision	VerifyCollision	UpdateEtd
CHCT	$4T_e$+T_m+$2T_p$	$1T_e$	$2T_e$+$1T_m$	$2T_e$+$1T_m$+$4T_p$	$3T_e$+$3T_m$
RCH [25]	$5T_e$+$2T_m$+$2T_p$	$2T_e$	$2T_e$+$2T_m$	$2T_e$+$1T_m$+$4T_p$	$8T_e$+$7T_m$
TCH [24]	$3T_e$+T_m	kT_e	(k+1)T_e+$3T_m$+$2T_p$	T_e+T_m+$2T_p$	N/A
ITCH [33]	$2T_e$+T_m	kT_e	kT_e+$2T_m$+ T_{inv} + $2T_p$	T_m+$2T_p$	N/A

[1] Denote T_e as group exponentiation; T_m as group multiplication; T_{inv} as the inverse operation of group element; T_p as bilinear pairing operation;

TABLE II
PROPERTIES OF DIFFERENT SCHEMES

Schemes	Properties		
	Trapdoor can be updated	Trapdoor can be updated flexibly	No key exposure problem
CHCT	√	√	√
RCH [25]	√	×	×
TCH [24]	×	×	×
ITCH [33]	×	×	×

VI. Performance Evaluations

In this section, we give the theoretical and experimental analysis of our new chameleon hash scheme.

A. Simulation Environment

We conducted real experiments to test the efficiency of the proposed chameleon hash scheme. The experiments are conducted on a laptop with the PBC library (version 0.5.14) and openssl library (version 1.1.1). The device configuration is 1.6-GHz 4-cores CPU and 4-GB RAM with Ubuntu 18.04 LTS (64 Bit) operating system.

B. Theoretical Analysis

We compare the dominant algorithms of CHCT with [24], [25], and [33] in Table I, where *GenCH*0 represents the whole process until the first chameleon hash is calculated. In [24] and [33], the bilinear pairing is required to check whether the params in a chameleon hash tuple can form the Diffie–Hellman tuple. Since the trapdoor in [24] and [33] is divided to k parts, the computation cost of the scheme will depend on param k. The comparison focuses on the number of complex operations. Table I shows that our chameleon hash scheme requires fewer operations in the group. Table II lists the properties satisfied by the above schemes, and it can be seen from the table that the scheme proposed in this article can satisfy all the desired properties.

C. Processing Performance

1) Computation Cost of Each Algorithm: We compare the computation cost of each algorithm with [25] in Fig. 3. The messages used in this experiment are generated randomly with the size of 1 kB. Each algorithm in the figure will be executed one time in sequence. Fig. 3 shows that the proposed scheme is more efficient compared with RCH [25].

2) Efficiency of Key Algorithms: In terms of the efficiency of chameleon hash scheme, we should focus on the computation cost of algorithm *FindCollision* and *VerifyCollision* since they will be executed frequently. *FindCollision*, *VerifyCollision* will be executed N times and their computation cost are combined in this experiment. The messages used in this experiment are generated randomly with the size of 256 Bytes. To prevent the trapdoor owner from abusing the trapdoor, TCH [24] and ITCH [33] divides the trapdoor into k parts and gives it to k trusted organizations. In this experiment, we set k to a small value ($k = 8$) since TCH and ITCH require plenty of exponential calculations, and the larger the k, the worse the performance of TCH and ITCH. Fig. 4 presents the computation cost of key algorithms with different times of collision. The efficiency of our scheme is similar with RCH [25] when N is small. When N is large, our scheme has a significant advantage in computation cost. The trapdoor in TCH and ITCH cannot be updated, and the abuse of trapdoor still exists. RCH realizes the periodic update of the trapdoor, however, the abuse of the trapdoor within the validity period is still unavoidable. Our solution does not allow the same trapdoor to be reused, which fundamentally solves the problem. From Fig. 4, we can conclude that our scheme adds strict restrictions on the use of trapdoor without undermining the performance of the scheme.

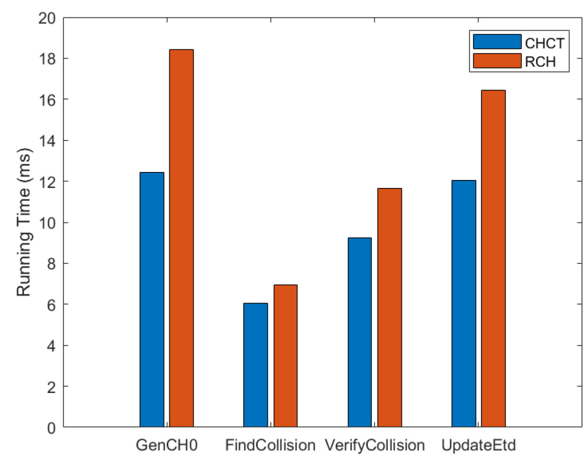
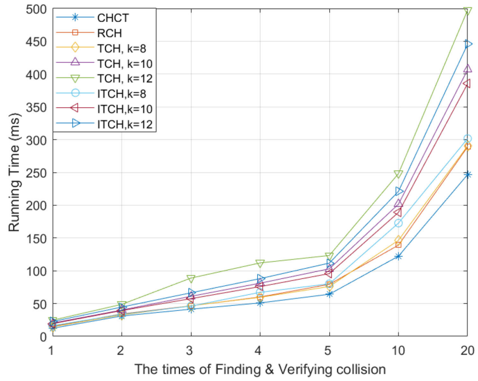
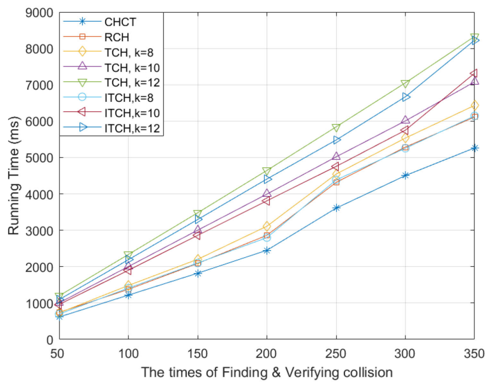


Fig. 3. Computation cost of different algorithms.

3) Stability and Compatibility: Chameleon hash is a special hash function and its efficiency should be independent

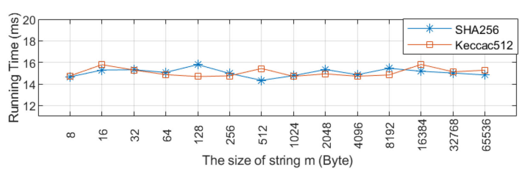


(a)

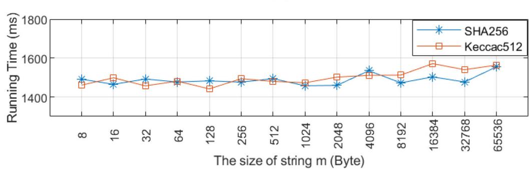


(b)

Fig. 4. Computation cost of key algorithms with different execution times. (a) Computation cost when N (Times) is small. (b) Computation cost when N (Times) is large.



(a)



(b)

Fig. 5. Stability for underlying hash function. (a) Computation cost when $N = 1$. (b) Computation cost when $N = 100$.

of the underlying hash function. We changed the underlying hash function from SHA-256 to Keccac-512 while the size of message m ranges from 8 to 65 536 Byte. In this experiment, we still focus on the computation cost of *FindCollision* and *VerifyCollision*. These algorithms will be executed N times and the total computation cost of each experiment was recorded. Fig. 5 shows that the computation overhead fluctuates slightly and our scheme is compatible with the different underlying hash algorithm.

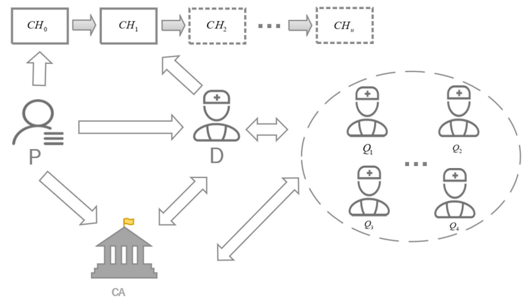


Fig. 6. Schematic of RMB.

VII. Instantiation of CHCT

In this section, we will briefly introduce how to employ CHCT to build RMB.

In the modern medical system, IoT device will be used to record the health information of patient, and the data generated from IoT device will form the electronic health record (EHR). Naturally, EHR has the requirement of modification since there will be privacy or erroneous data in the EHR. When blockchain is used to realize the decentralized storage of EHR, this requirement will conflict with the unmodifiable characteristics of blockchain, and CHCT can solve this problem well.

As is shown in Fig. 6, there will be three kinds of participants in RMB: 1) patient P; 2) doctor D; and 3) trusted entity CA. Note that CA records the public data (such as the trapdoor update parameter) and is not the centralized entity in RMB. CA participates in the blockchain as a miner and there can be several CAs in one area.

CA runs *CHInit* and broadcasts the system public parameter $param_{pub}$ at first. When patient P receives medical treatment from doctor D_1, they will negotiate a secret key key_1 to encrypt EHR. P uploads the encrypted data to the cloud and can access it through the unique access link. Then, P treats the access link as m and runs *CHGen* to calculate the chameleon hash tuple CH_0. After that, P runs *EtdShare* to generate the encrypted trapdoor parameter *EtdParam* and Rec_{CID}. Finally, the chameleon hash tuple CH_0, the commitment cmt and Rec_{CID} will be sent to CA via public channel and the encrypted trapdoor parameter *EtdParam* will be sent to D_1. CA will record cmt, Rec_{CID}, and anyone can query these parameters. CH_0 will be added to blockchain if it is valid.

If doctor D_1 wants to modify the EHR of P, he needs to provide the new medical data to other doctors for review. Specifically, D_1 will offer parameters to CA and CA will run *PointsGen*. If the number of doctors who agree to modify the EHR reaches the prescribed number in $param_{pub}$, CA and D_1 can run *EtdConsult* to get the trapdoor update parameter. Then, D_1 encrypts the raw medical data with key_1. The encrypted medical data will be uploaded to the cloud and D_1 can obtain an access link. Then, D_1 treats the access link as $m\prime$ and runs *FindCollision* to generate the new chameleon hash tuple. CA will verify whether the new chameleon hash tuple provided by D_1 forms a valid chameleon hash collision. After that, the request to modify the block will be broadcasted among miners. If P wants to revoke D_1's authority of modifying medical data, P can runs *EtdAbolish* to generate another trapdoor and chameleon hash tuple. This new chameleon hash tuple will be accepted by *CA* only if: 1) *CA* can use *CmtOpen* to verify *P*'s identity and 2) it is a valid chameleon hash collision. After that, the chameleon hash tuple provided by *P* will replace the counterpart in blockchain. *CA* and the other miners will reject the chameleon hash tuple generated by D_1 even if D_1 can still find a valid chameleon hash collision.

VIII. CONCLUSION

In this article, we proposed an efficient chameleon hash scheme for secure FL in IIoT. Different from the existing schemes, the trapdoor in our scheme can be abolished at any time. Besides, each trapdoor will be used once to avoid the key exposure problem fundamentally. Another interesting feature of our scheme is that everyone can supervise the update of each trapdoor without knowing any trapdoor, which puts a stronger constraint on the use of trapdoors. The security analysis and experiments show that our scheme is security and efficient. We also briefly introduced how to build the RMB with the proposed chameleon hash scheme.

REFERENCES

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015.

[2] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016.

[3] D. Minoli, K. Sohraby, and B. Occhiogrosso, "IoT considerations, requirements, and architectures for smart buildings–Energy optimization and next-generation building management systems," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 269–283, Feb. 2017.

[4] J.-S. Fu, Y. Liu, H.-C. Chao, B. K. Bhargava, and Z.-J. Zhang, "Secure data storage and searching for industrial IoT by integrating fog computing and cloud computing," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4519–4528, Oct. 2018.

[5] J. Wei, T. V. X. Phuong, and G. Yang, "An efficient privacy preserving message authentication scheme for Internet-of-Things," *IEEE Trans. Ind. Informat.*, vol. 17, no. 1, pp. 617–626, Jan. 2021.

[6] R. Li, T. Song, B. Mei, H. Li, X. Cheng, and L. Sun, "Blockchain for large-scale Internet of Things data storage and protection," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 762–771, Sep./Oct. 2019.

[7] M. A. Ferrag, O. Friha, L. Maglaras, H. Janicke, and L. Shu, "Federated deep learning for cyber security in the Internet of Things: Concepts, applications, and experimental analysis," *IEEE Access*, vol. 9, pp. 138509–138542, 2021.

[8] K. Yu *et al.*, "Securing critical infrastructures: Deep-learning-based threat detection in IIoT," *IEEE Commun. Mag.*, vol. 59, no. 10, pp. 76–82, Oct. 2021.

[9] K. Yu *et al.*, "A blockchain-based Shamir's threshold cryptography scheme for data protection in industrial Internet of Things settings," *IEEE Internet Things J.*, early access, Nov. 13, 2021, doi: 10.1109/JIOT.2021.3125190.

[10] H. Liu *et al.*, "Blockchain and fl for collaborative intrusion detection in vehicular edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 16, pp. 6073–6084, Jun. 2021.

[11] W. Liang, Y. Fan, K.-C. Li, D. Zhang, and J.-L. Gaudiot, "Secure data storage and recovery in industrial blockchain network environments," *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6543–6552, Oct. 2020.

[12] H.-T. Wu, Y. Zheng, B. Zhao, and J. Hu, "An anonymous reputation management system for mobile crowdsensing based on dual blockchain," *IEEE Internet Things J.*, early access, Sep. 21, 2021, doi: 10.1109/JIOT.2021.3113997.

[13] D. Wang and X. Zhang, "Secure ride-sharing services based on a consortium blockchain," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2976–2991, Feb. 2021.

[14] M. A. Ferrag and L. Shu, "The performance evaluation of blockchain-based security and privacy systems for the Internet of Things: A tutorial," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17236–17260, Dec. 2021.

[15] H. Li, D. Han, and M. Tang, "A privacy-preserving storage scheme for logistics data with assistance of blockchain," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4704–4720, Mar. 2022.

[16] Y. Lin, J. Li, S. Kimura, Y. Yang, Y. Ji, and Y. Cao, "Consortium blockchain-based public integrity verification in cloud storage for IoT," *IEEE Internet Things J.*, vol. 9, no. 5, pp. 3978–3987, Mar. 2022.

[17] C. Feng, B. Liu, K. Yu, S. K. Goudos, and S. Wan, "Blockchain-empowered decentralized horizontal federated learning for 5G-enabled uavs," *IEEE Trans. Ind. Informat.*, vol. 8, no. 5, pp. 3582–3592, May 2022.

[18] L. Tan, K. Yu, N. Shi, C. Yang, W. Wei, and H. Lu, "Towards secure and privacy-preserving data sharing for COVID-19 medical records: A blockchain-empowered approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 1, pp. 271–281, Jan.-Feb. 2022.

[19] J. Wei, G. Yang, and Y. Mu, "Comments on 'Accountable and privacy-enhanced access control in wireless sensor networks'," *IEEE Trans. Wireless Commun.*, vol. 15, no. 4, pp. 3097–3099, Apr. 2016.

[20] J. Wei, X. Wang, N. Li, G. Yang, and Y. Mu, "A privacy-preserving fog computing framework for vehicular crowdsensing networks," *IEEE Access*, vol. 6, pp. 43776–43784, 2018.

[21] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable blockchain–or–Rewriting history in bitcoin and friends," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, 2017, pp. 111–126.

[22] G. Yu *et al.*, "Enabling attribute revocation for fine-grained access control in blockchain-IoT systems," *IEEE Trans. Eng. Manag.*, vol. 67, no. 4, pp. 1213–1230, Nov. 2020.

[23] J. Xu, K. Xue, H. Tian, J. Hong, D. S. Wei, and P. Hong, "An identity management and authentication scheme based on redactable blockchain for mobile networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 6, pp. 6688–6698, Jun. 2020.

[24] K. Huang *et al.*, "Building redactable consortium blockchain for industrial Internet-of-Things," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3670–3679, Jun. 2019.

[25] K. Huang, X. Zhang, Y. Mu, F. Rezaeibagha, X. Du, and N. Guizani, "Achieving intelligent trust-layer for Internet-of-Things via self-redactable blockchain," *IEEE Trans. Ind. Informat.*, vol. 16, no. 4, pp. 2677–2686, Apr. 2020.

[26] H. Krawczyk and T. Rabin, "Chameleon hashing and signatures," in *Proc. NDSS*, 2000, pp. 143–154.

[27] G. Ateniese and B. de Medeiros, "Identity-based chameleon hash and applications," in *Proc. Int. Conf. Financial Cryptography*, 2004, pp. 164–180.

[28] G. Ateniese and B. de Medeiros, "On the key exposure problem in chameleon hashes," in *Proc. Int. Conf. Secur. Commun. Netw.*, 2004, pp. 165–179.

[29] X. Chen, F. Zhang, and K. Kim, "Chameleon hashing without key exposure," in *Proc. Int. Conf. Inf. Secur.*, 2004, pp. 87–98.

[30] W. Gao, X.-L. Wang, and D.-Q. Xie, "Chameleon hashes without key exposure based on factoring," *J. Comput. Sci. Technol.*, vol. 22, no. 1, pp. 109–113, 2007.

[31] X. Chen, H. Tian, F. Zhang, and Y. Ding, "Comments and improvements on key-exposure free chameleon hashing based on factoring," in *Proc. Int. Conf. Inf. Secur. Cryptol.*, 2010, pp. 415–426.

[32] X. Chen, F. Zhang, W. Susilo, H. Tian, J. Li, and K. Kim, "Identity-based chameleon hashing and signatures without key exposure," *Inf. Sci.*, vol. 265, pp. 198–210, May 2014.

[33] W. Gao, L. Chen, C. Rong, K. Liang, X. Zheng, and J. Yu, "Security analysis and improvement of a redactable consortium blockchain for industrial Internet-of-Things," *Comput. J.*, p. bxab080, Jun. 2021, doi: 10.1093/comjnl/bxab080.

[34] S. Ramesh and M. Govindarasu, "An efficient framework for privacy-preserving computations on encrypted IoT data," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8700–8708, Sep. 2020.

[35] X. Xiang, J. Cao, and W. Fan, "Scalable attestation protocol resilient to physical attacks for IoT environments," *IEEE Syst. J.*, vol. 15, no. 3, pp. 4566–4577, Sep. 2021.

[36] R. Guo, G. Yang, H. Shi, Y. Zhang, and D. Zheng, "O^3-R-CP-ABE: An efficient and revocable attribute-based encryption scheme in the cloud-assisted iomt system," *IEEE Internet Things J.*, vol. 8, no. 11, pp. 8949–8963, Jun. 2021.

[37] C. Peng, M. Luo, L. Li, K.-K. R. Choo, and D. He, "Efficient certificateless Online/Offline signature scheme for wireless body area networks," *IEEE Internet Things J.*, vol. 8, no. 18, pp. 14287–14298, Sep. 2021.

[38] T. Mizuide, A. Takayasu, and T. Takagi, "Tight reductions for Diffie-Hellman variants in the algebraic group model," in *Proc. Cryptograph. Track RSA Conf.*, 2019, pp. 169–188.

Jiannan Wei (Member, IEEE) received the M.S. degree in computer science from Zhengzhou University, Zhengzhou, China, in 2012, and the Ph.D. degree in information security from the School of Computing and Information Technology, University of Wollongong, Wollongong, NSW, Australia, in 2016.

She was previously an Academic Visitor with the Monash Blockchain Technology Center, Monash University, Melbourne, VIC, Australia, in 2019. She is currently an Assistant Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. She has authored publications, including papers in prestigious journal/conferences, such as the IEEE Transactions on Wireless Communications, IEEE Transactions on Industrial Informatics, IEEE Internet of Things Journal, *The Computer Journal*, TrustCom, ProvSec, and WOWMOM. Her research interests include applied cryptography, blockchain, and network security.

Qinchuan Zhu received the bachelor's degree from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2019, where he is currently pursuing the master's degree with the School of Computer Science and Engineering.

His research interests include blockchain and applied cryptography.

Qianmu Li (Member, IEEE) received the B.Sc. and Ph.D. degrees from Nanjing University of Science and Technology, Nanjing, China, in 2001 and 2005, respectively.

He is a Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology. His research interests include information security and data mining.

Prof. Li has won many scientific and technological awards, such as the China Network and Information Security Outstanding Talent Award and the Ministry of Education Science Award.

Laisen Nie (Member, IEEE) received the Ph.D. degree in communication and information system from Northeastern University, Shenyang, China, in 2016.

He is currently an Associate Professor with the Qingdao Research Institute of Northwestern Polytechnical University, Qingdao, China. His research interests include network measurement and cognitive networks.

Zhangyi Shen received the B.S. degree in information security from Hangzhou Dianzi University, Hangzhou, China, in 2015, and the M.S. and Ph.D. degrees in computer engineering from New Jersey Institute of Technology, Newark, NJ, USA, in 2016 and 2021, respectively.

He is currently an Instructor with the School of Cyberspace, Hangzhou Dianzi University. His research interests include multimedia security, information security, image forensics, and image anti-forensics.

Kim-Kwang Raymond Choo (Senior Member, IEEE) received the Ph.D. degree in information security from the Queensland University of Technology, Brisbane, QLD, Australia, in 2006.

He currently holds the Cloud Technology Endowed Professorship with The University of Texas at San Antonio, San Antonio, TX, USA.

Dr. Choo is a recipient of the 2019 IEEE Technical Committee on Scalable Computing Award for Excellence in Scalable Computing (Middle Career Researcher), and has received best paper awards from IEEE Systems Journal in 2021, IEEE Computer Society's Bio-Inspired Computing STC Outstanding Paper Award for 2021, IEEE DSC 2021, *IEEE Consumer Electronics Magazine* in 2020, *Journal of Network and Computer Applications* in 2020, *EURASIP Journal on Wireless Communications and Networking* in 2019, IEEE TrustCom 2018, and ESORICS 2015, as well as the Outstanding Editor Award for 2021 from Future Generation Computer Systems. He is the Founding Co-Editor-in-Chief of ACM Distributed Ledger Technologies: Research Practice, and the founding Chair of IEEE TEMS's Technical Committee on Blockchain and Distributed Ledger Technologies. He also serves as the Department Editor of IEEE Transactions on Engineering Management, an Associate Editor of IEEE Transactions on Dependable and Secure Computing and IEEE Transactions on Big Data, and the Technical Editor of *IEEE Network Magazine*. He is an ACM Distinguished Speaker and an IEEE Computer Society Distinguished Visitor from 2021 to 2023 and a Web of Science's Highly Cited Researcher.

Keping Yu (Member, IEEE) received the M.E. and Ph.D. degrees from the Graduate School of Global Information and Telecommunication Studies, Waseda University, Tokyo, Japan, in 2012 and 2016, respectively.

He was a Research Associate and a Junior Researcher with the Global Information and Telecommunication Institute, Waseda University from 2015 to 2019 and from 2019 to 2020, respectively, where he is currently a Researcher. He has authored more than 100 publications, including papers in prestigious journal/conferences, such as the IEEE Wireless Communications, *IEEE Communications Magazine*, *IEEE Network Magazine*, IEEE Internet of Things Journal, *Technological Forecasting and Social Change*, IEEE Transactions on Industrial Informatics, IEEE Transactions on Intelligent Transportation Systems, IEEE Transactions on Vehicular Technology, IEEE Journal of Biomedical and Health Informatics, *Therapeutic Recreation Journal*, IEEE Transactions on Communications, IEEE Transactions on Network and Service Management, *Transactions of the Institute of Measurement and Control*, IEEE Transactions on Network Science and Engineering, IEEE Transactions on Green Communications and Networking, CEM, IOTM, ICC, and GLOBECOM. He is an associate editor of several journals. His research interests include smart grids, information-centric networking, the Internet of Things, artificial intelligence, blockchain, and information security.

Dr. Yu received the IEEE Outstanding Leadership Award from IEEE BigDataSE 2021, the Best Paper Award from *IEEE Consumer Electronics Magazine* Award 2022 (first Place Winner), IEEE ICFTIC 2021, and ITU Kaleidoscope 2020.