An Efficiency-improved and Conditional Privacy-preserving Authentication Scheme Based on Merkle Hash Tree in MEC

Yan Zhang, Chunsheng Gu, Peizhong Shi, Zhengjun Jing, Bing Li, Weizhi Meng, *Senior Member, IEEE*

***Abstract*—Authentication is an important security issue for multi-access edge computing (MEC). However, the existing authentication schemes have not achieved a good balance between privacy preserving, efficiency, and low computation overhead on the device. To address this issue, we propose an efficiency-improved and privacy-preserving authentication scheme suitable for resource-constrained MEC devices. Our core idea is integrating the merkle hash tree (MHT) into the anonymous authentication scheme constructed by the blockchain and key derivation function (KDF) to improve efficiency. The MHT not only reduces the on-chain storage overhead brought by the increasing pseudo-public keys of KDF, but also utilizes few hash functions to achieve lightweight k-times accesses with the same edge server. Despite these advantages, managing pseudo-key pairs in the form of MHT leafs still brings efficiency and unlinkability problems. We construct the partially shuffled merkle hash tree to only shuffle leafs with in the device group, and combine with the KDF to update MHTs in a public manner by synchronizing pseudo-key pairs. Consequently, the efficiency of key update can be ensured. Moreover, a time-bound key derivation function based on physically unclonable function and BIP-32 is developed to provide immediate and permanent device revocation. Only the remaining valid pseudo-public keys of the revoked device will be recorded on the blockchain, which reveals no linkable information and avoids frequently reconstructing all the MHTs. We prove the authentication security and discuss other security properties. A proof-of-concept prototype was implemented to conduct experiments and comparative analysis to evaluate the performance.**

***Index Terms*—Multi-access edge computing, authentication, privacy preservation, merkle hash tree, blockchain, physically unclonable functions.**

I. Introduction

MULTI-ACCESS edge computing (MEC) has recently emerged as a promising architecture [1], which provides cloud computing services by deploying edge servers in the vicinity of users or various device within heterogeneous networks (e.g., WiFi, radio network, fixed access network, etc). The MEC constructs a high-bandwidth and low-latency working environment, as well as offers the context-aware computing ability [2]. As a result, innovative applications with stringent computation and communication demands have been spawned, such as smart manufacturing, health monitoring, intelligent transportation, making the MEC a key enabler to materialize Internet of Things (IoT) vision.

Chunsheng Gu (corresponding author) is with the School of Computer Engineering, Jiangsu University of Technology, Changzhou, Jiangsu, China. E-mail: guchunsheng@gmail.com.

Authentication is an important security issue for MEC [3]. Due to the openness of the network and device deployment, the authenticity of devices should be verified to ensure the security of communications and interactions with distributed edge servers. Furthermore, the MEC authentication also needs to consider properties of anonymity and efficiency. For anonymity, in sensitive applications, the identity information of the devices is closely related to the user privacy, such as driving records, working habits, and production capacity. As communicators are not fully trusted, they may maliciously leak real device identities to violate the user privacy. For efficiency, after successful authentication, devices often need to continuously access the same edge server for multiple times in scenarios of resource sharing [4], [5] and zero-trust architecture [6]. The verification of frequent accesses should not influence the low latency of MEC. Therefore, achieving anonymous and efficient authentication in the MEC is of great significance for improving the MEC security.

A. The Limitation of Current Studies

Anonymous authentication has already been the research focus in the security of MEC and IoT [3]. The existing anonymous authentication schemes can be divided into the following five categories, according to their involved technologies:

(1) Identity encryption: The device utilizes the public key of the edge server to anonymize the authentication request [7]. However, the curious edge server can still link different requests to the same device.

(2) Pseudo-identities: To support anonymity, the edge server pre-stores a set of devices' pseudo-identities or updates a new pseudo-identity during each request [8]. However, the real device identity will still be exposed to the server.

(3) Periodical pseudo-key pairs: The trusted authority periodically updates the on-chain devices' pseudo-public keys generated by the key derivation function [9], [10]. The device uses pseudo-key pairs to achieve authentication with conditional anonymity. However, the increasing number of pseudo-public keys will bring large on-chain storage overhead. Besides, relying on the TA to frequently update each device's pseudo-public key on the blockchain will lead to additional computation and communication overhead.

(4) Group or ring signature: The device constructs authentication requests based on the group [11] or ring signature [12], which provides the properties of conditional anonymity and unlinkability. However, the construction of group or ring

signature involves the time-consumption bilinear mapping operation or complex certificate management [13], which are all not suitable for resource-constrained devices.

(5) One-authentication-multiple-access: In the work [4], [5], after performing one successful authentication, the merkle hash tree (MHT) [4] or sequence-of-zero-knowledge proof [5] will be integrated to support devices efficiently accessing services for k times. However, the device also leverages the time-consumption group signature to achieve relaxed anonymous authentication to get the grant for multiple accesses.

Based on the above analysis, the existing anonymous authentication schemes have not achieved a good balance between privacy preserving, efficiency, and low computation overhead on the device in MEC. Hence, there is an urgent need to explore:

How to achieve an efficiency-improved and conditional privacy-preserving authentication scheme for resource-constrained MEC devices?

B. Our Motivation and Proposal

To achieve the above goal, our core idea is to introduce the merkle hash tree (MHT) into the anonymous authentication scheme constructed by the blockchain and key derivation function, which is proposed in Lin et al.'s work EBCPA [9]. For the sake of clarity, we summarize Lin et al.'s method as follows: The pseudo-key pairs generated by KDF is used to achieve anonymous authentication. Meanwhile, the synchronization feature of KDF effectively realizes the conditional privacy-preserving. The blockchain manages these pseudo-public keys in the form of transactions to better support the distributed environment. In this paper, we propose integrating MHT into their work to improve the efficiency by reducing the on-chain storage overhead and the computation overhead on the device side. On the one hand, the pseudo-public keys can be constructed as leafs of MHT. Only the root of MHT needs to be recorded on the blockchain on behalf of all the pseudo-public keys, thus greatly reducing the on-chain storage demand. On the other hand, in MEC applications that require continuous accesses, once being successfully authenticated, the device will be granted to utilize leafs and authentication path information of MHT to access the same edge server for multiple times. The computation overhead on the device side will keep low, as only few lightweight hash functions will be involved to finish each access.

Although introducing MHT has the potential to improve efficiency, managing pseudo-key pairs in the form of MHT leafs still brings efficiency and unlinkability problems to be further solved.

- The efficiency of key update: The devices' pseudo-public keys will be constructed as leafs of MHTs. The existence of each pseudo-public key can be proved by the leaf and its related authentication path information (API). To keep device unlinkability, the leafs from the same device should be shuffled. As a result, when KDF updates pseudo-key pairs, the TA needs to build secure channels with each device to distribute the API. The efficiency of key update will be affected.
- The balance of efficiency and unlinkability in key revocation: If it needs to revoke a part of pseudo-public keys in device revocation, there obviously exist two methods. The first reconstructs new merkle hash trees. The efficiency would be influenced. The second records pseudo-public pairs of the revoked device on the blockchain, which can be used to link to one certain device. The property of unlinkability could be broken.

To further solve the above problems, we propose the partially shuffled merkle hash tree and time-bound key derivation function to achieve an efficiency-improved and conditional privacy-preserving authentication scheme for resource-constrained MEC devices. Our contributions are summarized as follows:

(1) We propose integrating MHT into the anonymous authentication scheme constructed by the blockchain and key derivation function to improve efficiency. The MHT tree is leveraged to not only reduce the storage overhead of on-chain pseudo-public keys, but also support devices to perform k-times access with the same edge server, which only involves few hash functions.

(2) We introduce the group manager in the system model to construct the partially shuffled merkle hash tree. We only shuffle leafs within the device group, and make the group manager leverage the KDF to synchronize new pseudo-key pairs with the TA in a public manner. The key update only needs to build secure channels within the device group, needless of frequently involving in the TA. Therefore, the efficiency of key update would be ensured.

(3) We construct a time-bound key derivation function based on physically unclonable function (PUF) and BIP-32 to provide immediate and permanent device revocation. The immediate revocation will not break the unlinkability and avoid frequently reconstructing partially shuffled MHTs for all the devices' pseudo-public keys.

- Permanent revocation: The permanent revocation will only be performed at the beginning of each time period. The TA will remove the devices' pseudo-public keys from merkle hash trees of the new time period.
- Immediate revocation: The expire time is constructed as the key index of pseudo-key pairs, which are derived by PUFs-based reconstructed BIP-32. To revoke device immediately, only the leafs of unexpired pseudo-public keys will be recorded on the blochchain. No information that can be used to break unlinkability will be revealed.

(4) We prove the authentication security of our scheme and discuss its security features. A proof-of-concept prototype was implemented to conduct experiments and comparative analysis to evaluate the performance.

II. Related Work

In this section, we review anonymous authentication schemes in MEC and IoT applications.

To achieve the device identity anonymity, the studies [7], [15] utilized the public key of the edge server to send anonymous authentication requests. In work [8], the device pseudo-identities are pre-stored in the backend server and updated dynamically after each authentication. The methods of identity encryption and pseudo-identities only realize anonymity for attackers from the open channels, the curious edge servers can still get the real device identity or link different authentication to one certain device.

To support conditional privacy preservation, Lin et al. proposed authentication schemes [9], [10] in vehicular ad hoc networks (VANETS) by using blockchain to periodically update traceable pseudo-key pairs generated by KDF. The vehicle utilized pseudo-key pairs to construct signatures for anonymous message communications. Only the trusted authority is allowed to use the root key of these pseudo-key pairs to trace the real device identity. Zhang et at. proposed a dual blockchain-assisted conditional privacy-preserving authentication framework and protocol [14]. The consortium blockchain loads a pseudo-public key table, and the private blockchain maintains a private information table. Only the private blockchain node can quickly determine the real device identity by decrypting the mask value. In above mentioned studies, the increasing pseudo-public keys would bring significant on-chain storage overhead. The TA or the private blockchain node needs to be frequently involved in updating pseudo-key pairs. The system efficiency would be influenced.

There also exist anonymous authentication schemes constructed by group signature or ring signature. Huang et al. [11] constructed an efficient group signature scheme for data package authentication. Luo et al. [12] introduced ring signature to construct authentication protocol for VANETs. The group or ring schemes realized by the time-consuming bilinear pairing will put heavy computation overhead on the device side. Buser et al. constructed a post-quantum group signature scheme based on one-time signature (OTS) and MHT to reduce the computation overhead on the device side [13], which can be implemented on resource-constrained devices. OTS public keys will be built as leafs of MTH, and then shuffled to achieve anonymity and unlinkability. Their construction of OTS only needs few hash operations. However, the group signature constructed by OTS and MHT is not suitable for the distributed authentication in MEC, as different edge servers need to synchronize the used leafs. The management of device certificates or keys is complex.

The one-authentication-multiple-access framework has been proposed to restrict the access times. In scenarios of IoT resource sharing, the group signature was combined with MHT to achieve efficient and anonymous authentication [4]. To secure communications in space information network, periodic k-times anonymous authentication also has been proposed in [16] to allow users to authenticate satellite users to ground stations at most k times. In cloud computing, the sequence-of-zero-knowledge-proof mechanism is utilized to design k-times anonymous pay-as-you-go authentication. The features of flexible access controllability, anonymity, and public accountability are all supported. However, the existing anonymous k-times authentication studies still relies on the heavy bilinear pairing to grant multiple accesses. The requirement of low-computation overhead for the resource-constrained device has not been satisfied.

Therefore, there still lacks an anonymous authentication scheme in distributed MEC systems to simultaneously achieve the issues of privacy preservation, efficiency, and low-computation overhead on the device.

III. System model

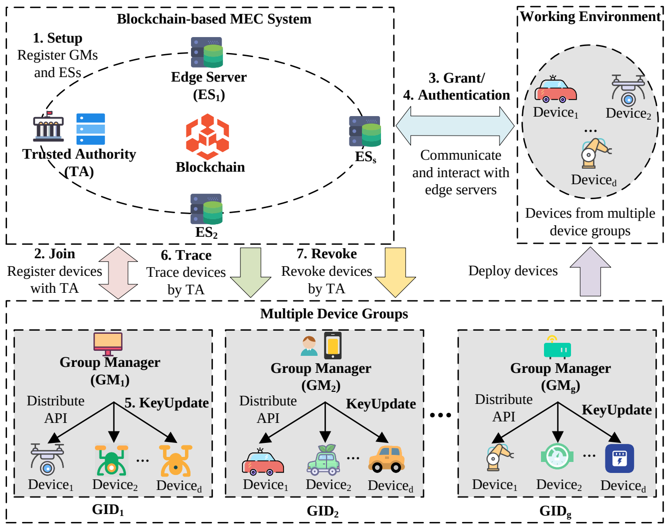

Fig. 1. System model.

A. System Model

The system model is depicted in Fig. 1. There are mainly five entities jointly performing our authentication scheme.

(1) Trusted Authority (TA): The TA is modeled as an authority that is trusted by all the entities and has enough resources. The edge servers and group managers are registered by the TA. Besides, the group manager will interact with the TA to enroll the devices belonging to its group. The TA setups the blockchain network and has the authority to manage the blockchain ledger.

(2) Edge Server (ES): Edge servers have decent computation and storage resources to provide MEC applications and services. They will also participate in the blockchain network to maintain the blockchain ledger by the consensus mechanism.

(3) Group Manager (GM): The GM is introduced into our model to mainly perform the key update. For different MEC applications, GMs can be gateways, mobile phones, personal digital assistants, etc. The GM will periodically update pseudo-key pairs for devices in the group. The communication channels between the GM and devices can be protected by the multi-factor authentication, which can adaptively select appropriate factors according to the type of the GM.

(4) Device: Devices need to communicate and interact with distributed edge servers. The device will first be authenticated by the edge server, and then continuously access the same edge server for MEC services or resources. Note that each device will be inserted with the PUF circuit.

(5) Blockchain: The TA and edge servers run a consortium blockchain network. The blockchain utilizes transactions to manage root nodes and revoked leafs of merkle hash trees, which together provide the existence proof for devices'

pseudo-public keys. The TA and edge servers install smart contracts to manage the blockchain ledger.

B. Syntax of Our Authentication Scheme

Our authentication scheme is composed of five protocols: **Setup**, **Join**, **Grant**, **Auth**, **KeyUpate**, and two algorithms: **Revoke**, **Trace**.

(1) **Setup**: In this protocol, the TA first publishes public parameters, and then registers edge servers and GMs. In addition, the TA and edge servers will run the blockchain network and install the smart contract.

(2) **Join**: The TA, GMs and devices jointly perform this protocol through secure channels. Devices in each group first generate root key pairs and real identities. Then, the GM registers root public keys and identities into the TA on behalf of devices. Once receiving the synchronization information (SI) from the TA, the GM generates the authentication path information (API) of leafs (time-bound pseudo-public keys). Last, the API and key index of pseudo-public keys will be securely distributed to relevant devices by the GM.

(3) **Grant**: The device utilizes its time-bound pseudo-key pairs to authenticate with distributed edge servers. The pseudo-key pair will only be valid in a certain time period, such as 10 minutes. After authentication, the edge server will grant k-times accesses to legitimate devices.

(4) **Auth**: After being authorized in **Grant**, the device will be allowed to continuously access the same edge server for k times.

(5) **KeyUpdate**: The TA, GMs and devices will jointly perform this protocol to update all the devices' pseudo-key pairs. The TA will record the root nodes of newly built MHTs on the blockchain. Meanwhile, each GM will locally synchronize these MHTs. Once receiving the key update request, the GM will secretly distribute the API of leafs to the corresponding device.

(6) **Trace**: The TA utilizes this algorithm to trace the real identities of devices, according to the messages transmitted in **Grant** and **Auth** protocols.

(7) **Revoke**: The TA uses this algorithm to revoke devices. The root nodes of revoked devices will be published on the blockchain. Then, the TA will not generate new MHTs for these devices in the next time period of pseudo-key pairs.

C. Threat Model and Goals

The threat model defines the knowledge and abilities of the adversary in our system.

(1) The adversary's abilities are defined as follows:

- The adversary controls the public channel to modify, replay, eavesdrop, intercept the transmitted data.
- The adversary cannot modify the recorded transactions or break the consensus mechanism. The consortium blockchain is assumed to be reliable and secure.

(2) The adversary's knowledge is described as follows:

- The adversary can access to the local storage of devices by side channel attacks or malware attacks. However, the adversary cannot derive secret keys from the device's running memory, which can be ensured by using technologies such as trusted execution environments, hardware security module, etc.
- The adversary cannot obtain secret keys of the TA, edge servers, and GMs. Their identities and public keys will be made public.
- The adversary cannot get the device root key derived from the PUF response, as the inserted PUF circuit is unclonable, and the generated PUF response is random and unpredictable.

(3) Under the threat of the above adversaries, our scheme must achieve the following goals:

- **Authentication security**: The adversary cannot use an unregister device or forge a registered device to pass the **Grant** protocol. In addition, the adversary cannot utilize an ungrant device to pass the **Auth** protocol.
- **Conditional anonymity**: The device's real identity will not be exposed. Only the TA or GM is allowed to trace real device identities to regulate the system, such as revoking devices with malicious behaviours.
- **Periodical unlinkability** Given two grant requests that are randomly selected from different time periods, the adversary cannot link these grant requests to the same device. In other words, only the grant requests singed by the same pseudo-key pair in each time period can be linked to the same device, as defined in [9]. However, the device's granted k-times accesses with the same server are also linkable, as defined in [4].
- **Resistance to other attacks**: Our scheme is required to resist common attacks, including malicious impersonation, replay attack, physical and cloning attacks.

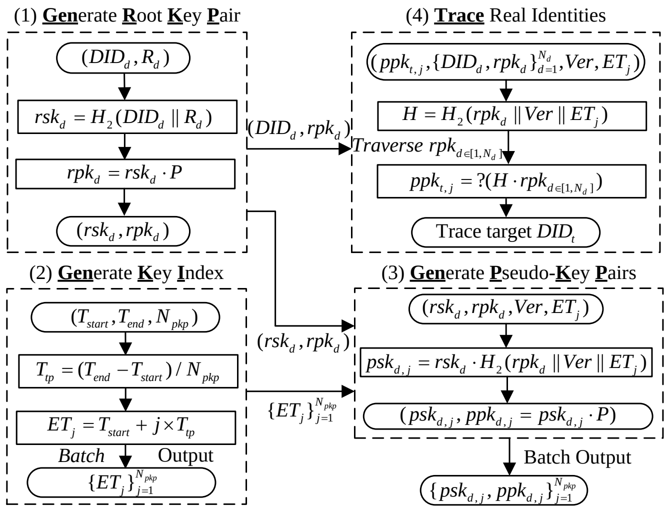
(1) Generate Root Key Pair
(DID_d, R_d)
$rsk_d = H_2(DID_d \| R_d)$
$rpk_d = rsk_d \cdot P$
(rsk_d, rpk_d)
(DID_d, rpk_d)
(4) Trace Real Identities
$(ppk_{t,j}, \{DID_d, rpk_d\}_{d=1}^{N_d}, Ver, ET_j)$
$H = H_2(rpk_d \| Ver \| ET_j)$
Traverse $rpk_{d\in[1,N_d]}$
$ppk_{t,j} = ?(H \cdot rpk_{d\in[1,N_d]})$
Trace target DID_t
(2) Generate Key Index
$(T_{start}, T_{end}, N_{pkp})$
$T_{tp} = (T_{end} - T_{start}) / N_{pkp}$
$ET_j = T_{start} + j \times T_{tp}$
Batch Output
$\{ET_j\}_{j=1}^{N_{pkp}}$
(rsk_d, rpk_d)
$\{ET_j\}_{j=1}^{N_{pkp}}$
(3) Generate Pseudo-Key Pairs
$(rsk_d, rpk_d, Ver, ET_j)$
$psk_{d,j} = rsk_d \cdot H_2(rpk_d \| Ver \| ET_j)$
$(psk_{d,j}, ppk_{d,j} = psk_{d,j} \cdot P)$
Batch Output
$\{psk_{d,j}, ppk_{d,j}\}_{j=1}^{N_{pkp}}$

Fig. 2. Time-bound KDF based on BIP-32 and PUFs.

D. Our Proposed Time-bound Key Derivation Function based on BIP-32 and PUFs

In our scheme, the trusted authority periodically updates pseudo-public keys for devices to achieve anonymous authentication (e.g., pre-submitting 240 public keys for one device for an interval of two days [9]). We propose the key derivation function based on BIP-32 and PUFs. Relying on KDF to update leafs of MHTs can ensure the efficiency of updating pseudo-key pairs, as there is no need to build secure channels between devices and the TA. In addition, the features of our reconstructed KDF can be concluded from two aspects: (1) To avoid breaking unlinkability when revoking pseudo-key pairs, we use the expire time ET_j as the index to generate time-bound pseudo-key pairs, since there is no need to revoke expired keys. (2) To enhance the device security, the device root key pairs (rsk_d, rpk_d) are generated by the PUF circuit.

In this paper, a cyclic group $\mathbb{G}$ on elliptic curve $E(\mathbb{F}_p)$ with a big prime p, a generator $P \in \mathbb{G}$, and the order q will be chosen. We also select two hash functions $H_1 : \{0,1\}^* \rightarrow \{0,1\}^{l_1}$, $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q$, where l_1 is the bit length of hash value. Based on these definitions, our reconstructed KDF contains the following algorithms, as shown in Fig. 2.

(1) $\mathbf{KDF.GenRKP}(DID_d, R_d) \rightarrow (rsk_d, rpk_d)$. This algorithm inputs device identity DID_d and the PUF response R_d derived from the PUF circuit. The root secret key $rsk_d = H_2(DID_d||R_d)$ is generated to compute the root public key $rpk_d = rsk_d \cdot P$.

(2) $\mathbf{KDF.GenKI}(T_{start}, T_{end}, N_{pkp}) \rightarrow \{ET_j\}_{j=1}^{N_{pkp}}$. This algorithm inputs the start time T_{start}, end time T_{end}, and the number of pseudo-key pairs N_{pkp} of each time period. The time period $T_{tp} = (T_{end} - T_{start})/N_{pkp}$ of each pseudo-key pair is used to generate the expire time ET_j as the key index. A set of $\{ET_j\}_{j=1}^{N_{pkp}}$ will be output to index the the derived pseudo-key pairs in our key derivation function.

(3) $\mathbf{KDF.GenPKP}(rsk_d, rpk_d, Ver, ET_j) \rightarrow \{psk_{d,j}, ppk_{d,j}\}$. This algorithm inputs the device root secret key rsk_d, Ver, and the expire time ET_j. Then, the j-th pseudo-key pair is computed as $psk_{d,j} = rsk_d \cdot H_2(rpk_d||Ver||ET_j)$ and $ppk_{d,j} = psk_{d,j}P$. In each Ver, N_{pkp} pseudo-key pairs $\{psk_{d,j}, ppk_{d,j}\}_{j=1}^{N_{pkp}}$ will be batch generated.

(4) $\mathbf{KDF.Trace}(ppk_{t,j}, Ver, ET_j, \{DID_d, rpk_d\}_{d=1}^{N_d}) \rightarrow DID_t$. The TA utilizes this algorithm to trace the real device identity DID_t from the pseudo-public key $ppk_{t,j}$. The TA will traverse $\{DID_d, rpk_d\}_{d=1}^{N_d}$ to check whether $ppk_{t,j} \overset{?}{=} H \cdot rpk_{d \in [N_d]}$, where $H = H_2(rpk_d||Ver||ET_j)$. Last, this algorithm outputs DID_t.

E. Our Construction of Partially Shuffled Merkle Hash Tree

We construct the partially shuffled merkle hash tree to achieve anonymity by hiding the relationships between devices' pseudo-public keys and MHTs' root nodes. Moreover, the efficiency of updating pseudo-key pairs can be ensured by the integration of our key derivation function, which does not need to build secure channels between devices and the TA. As shown in Fig. 3, the construction of partially shuffled merkle hash tree contains four algorithms as follows.

(1) $\mathbf{MHT.SL}(\{\{ppk_{d,j}, ET_j\}_{j=1}^{N_{pkp}}\}_{d=1}^{N_d}) \rightarrow \{leaf_l\}_{l=1}^{N_d N_{pkp}}$. This algorithm inputs $(N_d N_{pkp})$ pseudo-public keys of N_d devices. Each device is assigned N_{pkp} pseudo-key pairs. Leafs are computed as $\{\{H_1(ET_j||ppk_{d,j})\}_{j=1}^{N_{pkp}}\}_{d=1}^{N_d}$. After that, we will sort these leafs by the hash value, and output the shuffled leafs as $\{leaf_l\}_{l=1}^{N_d N_{pkp}}$.

(2) $\mathbf{MHT.BPST}(H_{mht}, \{leaf_l\}_{l=1}^{N_d N_{pkp}}) \rightarrow \{root_m, MHT_m\}_{m=1}^{N_{mht_g}}$. This algorithm inputs shuffled leafs $\{leaf_l\}_{l=1}^{N_d N_{pkp}}$ of GID_g and the height H_{mht} of each MHT. As building only one MHT will bring a very long authentication path for each leaf, we construct different smaller MHTs with the height of H_{mht} to reduce the burden on the device's storage and transmission of APIs. This algorithm first checks whether $N_{mht_g} = (N_d N_{pkp})/2^{H_{mht}} \in \mathbb{N}_+$. If not, abort the algorithm. Otherwise, all the leafs will be used to build N_{mht_g} different MHTs, which are denoted as $\{MHT_m\}_{m=1}^{N_{mht_g}}$ and have N_{mht_g} root nodes. When being expanding to N_g groups, this algorithm outputs all groups' root nodes and MHTs as $\{\{root_{g,m}, MHT_{g,m}\}_{m=1}^{N_{mht_g}}\}_{g=1}^{N_g}$. Note that root nodes $\{root_{m'}\}_{m'=1}^{\sum_{g=1}^{N_g} N_{mht_g}}$ will be recorded on the blockchain.

(3) $\mathbf{MHT.BTT}(N_{tmp}, H_{mht}, \{leaf_l\}_{l=1}^{(N'_d N'_{pkp})}) \rightarrow \{root'_m, MHT'_m\}_{m=1}^{N'_{mht}}$. If the time of executing $\mathbf{KeyUpdate}$ has not arrived, the TA will leverage this algorithm to temporarily add N'_d devices and their leafs $\{leaf_l\}_{l=1}^{(N'_d N'_{pkp})}$, where N'_{pkp} is the number of the remaining valid pseudo-key pairs during this time period. This algorithm requires to build at least N_{tmp} MHTs to keep the unlinkability when adding new devices. First, we will add t random numbers that are included in $\mathbb{Z}_{2^{l_1}}$ as additional leafs, which can build N'_{mht} complete temporary MHTs with the input $(N'_d N'_{pkp})$ leafs. The N'_{mht} should satisfy: $N'_{mht} = (N'_d N'_{pkp} + t)/2^{mht} \in \mathbb{N}_+, N'_{mht} \geq N_{tmp}$. Next, these $(N'_d N'_{pkp} + t)$ leafs will be shuffled by the hash value to get $\{leaf_l\}_{l=1}^{(N'_d N'_{pkp})+t}$. After that, $\mathbf{MHT.BPST}(H_{mht}, \{leaf_l\}_{l=1}^{(N'_d N'_{pkp}+t)})$ will be taken to construct N'_{mht} MHTs and root nodes as $\{root'_m, MHT'_m\}_{m=1}^{N'_{mht}}$. Note that the newly generated root nodes $\{root'_m\}_{m=1}^{N'_{mht}}$ will be recorded on the blockchain.

(4) $\mathbf{MHT.GenAPI}(H_{mht}, \{leaf_l\}_{l=1}^{N_l}) \rightarrow \{API_l\}_{l=1}^{N_l}$. Assume that $N_l = N_{mht} \times 2^{H_{mht}}$, and $N_{mht} \in \mathbb{N}_+$. This algorithm will use $\mathbf{MHT.BPST}(H_{mht}, \{leaf_l\}_{l=1}^{N_l})$ to reconstruct all N_{mht} MHTs and then compute the authentication path information API for every leaf node. For example, as shown in Fig. 3, the green node, orange node, and yellow node are the authentication path $path$ for the first left red leaf. Therefore, the API_l of the l-th leaf is defined as $\{leaf_l : path_l\}$.

F. Smart Contract

The smart contract manages the transactions in the form of a key-value state database to provide five functions: $\mathbf{SC.Create}(key, value)$, $\mathbf{SC.Update}(key, value)$, $\mathbf{SC.Add}(key, value)$, $\mathbf{SC.Delete}(key)$, and $(value) \leftarrow \mathbf{SC.Query}(key)$. As shown in Fig. 3, these functions maintain two key-value data structures on the blockchain as follows.

The key of the first data structure is the identity of root node list ID_{rn}, and the value is the on-chain root node

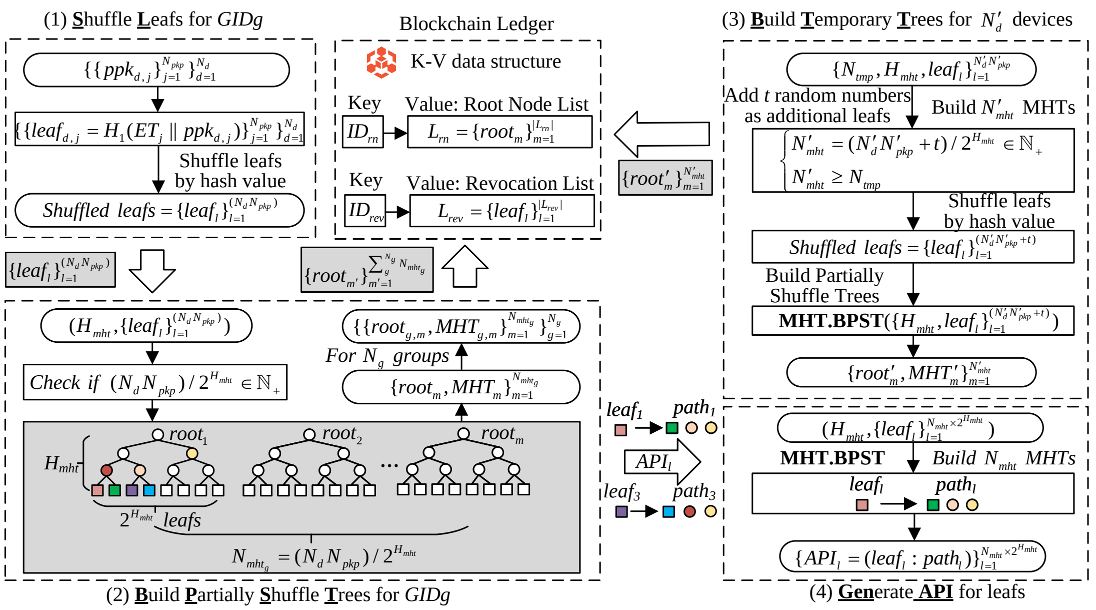
(1) Shuffle Leafs for GIDg
Shuffle leafs by hash value
Blockchain Ledger
K-V data structure
Key
Value: Root Node List
Value: Revocation List
(3) Build Temporary Trees for N'_d devices
Add t random numbers as additional leafs
Build N'_{mht} MHTs
Shuffle leafs by hash value
Build Partially Shuffle Trees
Check if $(N_d N_{pkp})/2^{H_{mht}} \in \mathbb{N}_+$
For N_g groups
$2^{H_{mht}}$ leafs
$N_{mht_g} = (N_d N_{pkp})/2^{H_{mht}}$
MHT.BPST
Build N_{mht} MHTs
(2) Build Partially Shuffle Trees for GIDg
(4) Generate API for leafs

Fig. 3. The construction of partially shuffled MHT.

list $L_{rn} = \{root_m\}_{m=1}^{|L_{rn}|}$. The TA can register the root node list by $\mathbf{SC.Create}(ID_{rn}, L_{rn})$. Moreover, the TA also can construct new root node list L'_{rn} to update the on-chain root node list $\mathbf{SC.Update}(ID_{rn}, L'_{rn})$. To immediately add devices, root nodes of temporary MHTs will be added by the TA to the existing on-chain root node list using $\mathbf{SC.Add}(ID_{rn}, \{root'_m\}_{m=1,2,\dots})$. Edge servers are allowed to get the on-chain root node list $(L_{rn}) \leftarrow \mathbf{SC.Query}(ID_{rn})$.

The key of the second data structure is the identity of leafs revocation list ID_{rev}, and the value is the on-chain leafs revocation list $L_{rev} = \{leaf_l\}_{l=1}^{|L_{rev}|}$ generated by the remaining valid pseudo-public keys of revoked devices. For example, to immediately revoke a device DID_d that still has $(N_{pkp} - r + 1)$ valid pseudo-key pairs, the TA will record the corresponding leafs of these pseudo-public keys in L_{rev} by performing $\mathbf{SC.Add}(ID_{rev}, L_{rev} = \{leaf_{d,j}\}_{j=r}^{N_{pkp}})$. Moreover, the on-chain revocation list can be deleted by $\mathbf{SC.Delete}(ID_{rev})$. The edge server can also query the leafs revocation list from the blockchain ledger $L_{rev} \leftarrow \mathbf{SC.Query}(ID_{rev})$.

G. PUF and Fuzzy Extractor

The PUF circuit PUF_d inputs one challenge C_d and outputs one correlated response R_d, which generates the physically secure secret key for device DID_d [17]. However, there may exist noises in the derived PUF response. To support the noisy environment, we use the fuzzy extractor to register the PUF response R_d and get the helper data $hd_d \leftarrow \mathbf{FE.GEN}(R_d)$. Once getting the R_d^* with noises, the correct PUF response can be reproduced by $R_d \leftarrow \mathbf{FE.REP}(R_d^*, hd_d)$.

IV. Our Proposed Authentication Scheme

A. Setup

First, the TA publishes public parameters $pp = (p, q, P, \mathbb{G}, H_1, H_2)$, which have already been defined in section III-D. Then, the secret/public keys of the TA are computed as $(tsk \in \mathbb{Z}_q, tpk = tsk \cdot P \in \mathbb{G})$.

Second, the s-th edge server generates its identity SID_s, the secret key $ssk_s \in \mathbb{Z}_q$ and public key $spk_s = ssk_s \cdot P$. The zero-knowledge proof is used to register $\{SID_s, spk_s\}_{s=1}^{N_s}$ in the TA. Assume that the public keys and identities of edge servers will be known to all the entities. Moreover, the TA and edge server will run the blockchain network together and install the smart contract.

Third, similarly to the edge server, the g-th GM also generates its group identity and key pairs (GID_g, gsk_g, gpk_g). The group identity and public key will be registered and made public. According to the type of the GM, the gsk_g can be derived from one or more factors including the user biometrics, password, hardware circuit, smart card, etc.

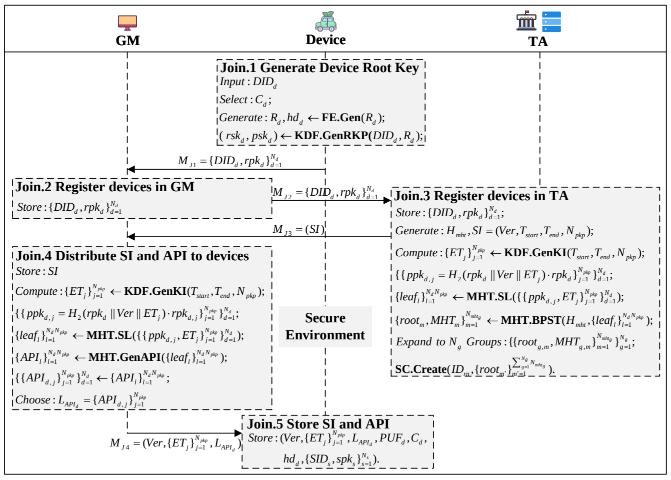
GM
Device
TA
Join.1 Generate Device Root Key
Join.2 Register devices in GM
Join.3 Register devices in TA
Join.4 Distribute SI and API to devices
Secure Environment
Join.5 Store SI and API

Fig. 4. Join protocol.

B. Join

The GM will register devices' key materials in the TA and distribute the authentication path information to devices in a secure environment. As is shown in Fig.4, the TA, devices and their GM will perform **Join** protocol as the following steps.

(1) **Join.1 Generate Device Root Key Pair**: This step inputs the device identity DID_d and randomly selects the PUF challenge C_d to first generate the PUF response R_d using the PUF circuit PUF_d. Then, the fuzzy extractor is utilized to derive the helper data $(hd_d) \leftarrow \mathbf{FE.Gen}(R_d)$. Next, the device's root secret/public keys are derived as $(rsk_d, rpk_d) \leftarrow \mathbf{KDF.GenRKP}(DID_d, R_d)$. The message $M_{J1} = (DID_d, rpk_d)$ will be sent to the GM.

(2) **Join.2 Register devices in GM**: The GM receives and stores the identities and root public keys of N_d devices $\{DID_d, rpk_d\}_{d=1}^{N_d}$ in the local storage. Then, these key materials will be constructed as M_{J2} and sent to the TA.

(3) **Join.3 Register devices in TA**: Once receiving M_{J2}, the TA locally stores $\{DID_d, rpk_d\}_{d=1}^{N_d}$. Afterward, the synchronization information $SI = (Ver, T_{start}, T_{end}, N_{pkp})$ and the height of MHT H_{mht} will be generated, where Ver is the version of pseudo-key pairs, T_{start} and T_{end} are the start and end time, and N_{pkp} is the number of pseudo-key pairs. Then, the TA computes the indexes of pseudo-key pairs as $\{ET_j\}_{j=1}^{N_{pkp}} \leftarrow \mathbf{KDF.GenKI}(T_{start}, T_{end}, N_{pkp})$ and N_d devices' pseudo-public keys of the first version as $\{\{ppk_{d,j} = H_2(rpk_d||Ver||ET_j) \cdot rpk_d\}_{j=1}^{N_{pkp}}\}_{d=1}^{N_d}$. Next, the shuffled leafs are generated as $\{leaf_l\}_{l=1}^{N_d N_{pkp}} \leftarrow \mathbf{MHT.SL}(\{\{ppk_{d,j}, ET_j\}_{j=1}^{N_{pkp}}\}_{d=1}^{N_d})$, which will further be used to build the partially shuffled MHTs $\{root_m, MHT_m\}_{m=1}^{N_{mht_g}} \leftarrow \mathbf{MHT.BPST}(H_{mht}, \{leaf_l\}_{l=1}^{N_d N_{pkp}})$ for the group GID_g, where N_{mht_g} is the number of MHTs in each group. Once collecting all the MHTs of N_g groups, the TA constructs $\sum_{g=1}^{N_g} N_{mht_g}$ MHTs and their root nodes as $\{\{root_{g,m}, MHT_{g,m}\}_{m=1}^{N_{mht_g}}\}_{g=1}^{N_g}$ Last, the TA utilizes the smart contract $\mathbf{SC.Create}(ID_{rn}, \{root_{m'}\}_{m'=1}^{\sum_{g=1}^{N_g} N_{mht_g}})$ to build the on-chain root node list L_{rn}, and also distributes $M_{J3} = SI$ to the GM. All the GMs share the same SI.

(4) **Join.4 Distribute SI and API to devices**: Once receiving SI from the TA, the GM first stores SI and synchronizes the pseudo-public keys as $\{\{ppk_{d,j} = H_2(rpk_d||Ver||ET_j) \cdot rpk_d\}_{j=1}^{N_{pkp}}\}_{d=1}^{N_d}$ by taking the same operations in **Join.3**. Next, the leafs are generated as $\{leaf_l\}_{l=1}^{N_d N_{pkp}} \leftarrow \mathbf{MHT.SL}(\{\{ppk_{d,j}, ET_j\}_{j=1}^{N_{pkp}}\}_{d=1}^{N_d})$, and then the corresponding API will be computed by $\{API_l\}_{l=1}^{N_d N_{pkp}} \leftarrow \mathbf{MHT.GenAPI}(H_{mht}, \{leaf_l\}_{l=1}^{N_d N_{pkp}})$. The GM classifies $\{API_l\}_{l=1}^{N_d N_{pkp}}$ into $\{\{API_{d,j}\}_{j=1}^{N_{pkp}}\}_{d=1}^{N_d}$ according to the shuffle order, and then builds the API list $L_{API_d} = \{API_{d,j}\}_{j=1}^{N_{pkp}}$ for each device DID_d. The Ver, $\{ET_j\}_{j=1}^{N_{pkp}}$ and L_{API_d} will be distributed to each device.

(5) **Join.5 Store SI and API**: Each device stores Ver, $\{ET_j\}_{j=1}^{N_{pkp}}$ and the API list L_{API_d} of this version. The PUF challenge C_d, helper data hd_d, and servers' identities and public keys are all stored in the devices's storage.

Remark: If N'_d devices are needed to immediately join the system during the time period, the TA will collect their device's real identities $\{(DID_d, rpk_d)\}_{d=1}^{N'_d}$ as **Join.3**, and then build the temporary MHTs $\{root'_m, MHT'_m\}_{m=1}^{N'_{mht}} \leftarrow \mathbf{MHT.BTT}(N_{tmp}, H_{mht}, \{leaf_l\}_{l=1}^{(N'_d N'_{pkp})})$, where N'_{pkp} is the number of remaining valid pseudo-key pairs in this time period. The new root nodes $\{root'_m\}_{m=1}^{N'_{mht}}$ will be added to the on-chain root node list $\mathbf{SC.Add}(ID_{rn}, \{root'_m\}_{m'=1}^{N'_{mht}})$. The information of API and SI will be distributed to added devices.

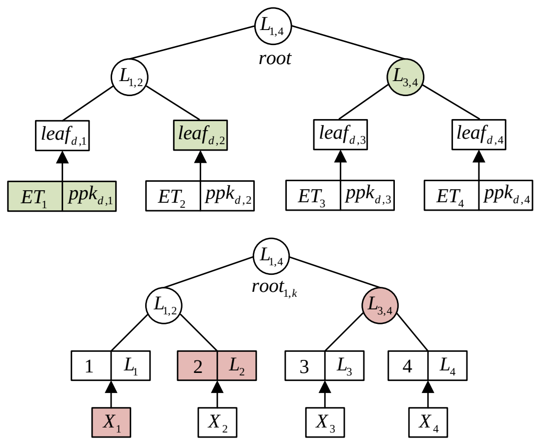

Fig. 5. The construction of two kinds of MHTs: (a) The first partially shuffled MHT with the *root* proves the validity of pseudo-public keys. For example, the $path_{d,1}$ proves the validity of $leaf_{d,1} = H_1(ET_1||ppk_{d,1})$, where $path_{d,1} = (leaf_{d,2}, L_{3,4})$; (2) The second MHT $MHT_{1,k}$ with the $root_{1,k}$ proves the validity of k-times access. Each $X_i = H_1(R_d||N_i^R)$ and $L_i = H_1(X_i||i)$ will only be used for once. For example, the $path_1$ for the leaf node $(1||L_1)$ is $path_1 = (2||L_2, L_{3,4})$. In this construction, the algorithm that **b**uilds **m**erkle **h**ash **t**ree with k leafs is defined as $(root_{1,k}, MHT_{1,k}) \leftarrow \mathbf{BMHT}(\{leaf_i\}_{i=1}^{k})$. The algorithm that **c**omputes the **r**oot of **MHT** using leaf node and the correlated path is defined as $root_{1,k} \leftarrow \mathbf{CRMHT}(leaf_i, path_i)$.

C. Grant

The device authenticates with any one of the edge servers to request the grant for k-times accesses. The authenticity of the device can be verified by the Schnorr signature constructed by time-bound pseudo-key pairs. The **Grant** protocol will be performed in a public channel, as demonstrated in Fig. 6.

(1) **Grant.1 Generate MHT for *k*-times Access**: The device DID_d inputs C_d to get the PUF response R_d^* with noises and reproduces the correct $R_d = \mathbf{FE.REP}(R_d^*, hd_d)$ using the fuzzy extractor. Next, the device's root key pair is computed as $(rsk_d, rpk_d) \leftarrow \mathbf{KDF.GenRKP}(DID_d, R_d)$. Then, the device generates $k = 2^v$ ($v \in \mathbb{N}_+$) random elements $\{X_i = H_1(R_d||N_i^R)\}_{i=1}^{k}$ using the PUF response R_d, where N_i^R is the random number. As shown in the lower part of Fig. 5, a new MHT $MHT_{1,k}$ will be built using k leafs $\{(i||L_i = H_1(X_i||i))\}_{i=1}^{k}$ to request k-times accesses with the edge server SID_s. The root node of $MHT_{1,k}$ is denoted as $root_{1,k}$.

(2) **Grant.2 Construct Signaure for Request**: The device first gets the expire time list $\{ET_j\}_{j=1}^{N_{pkp}}$ from the local

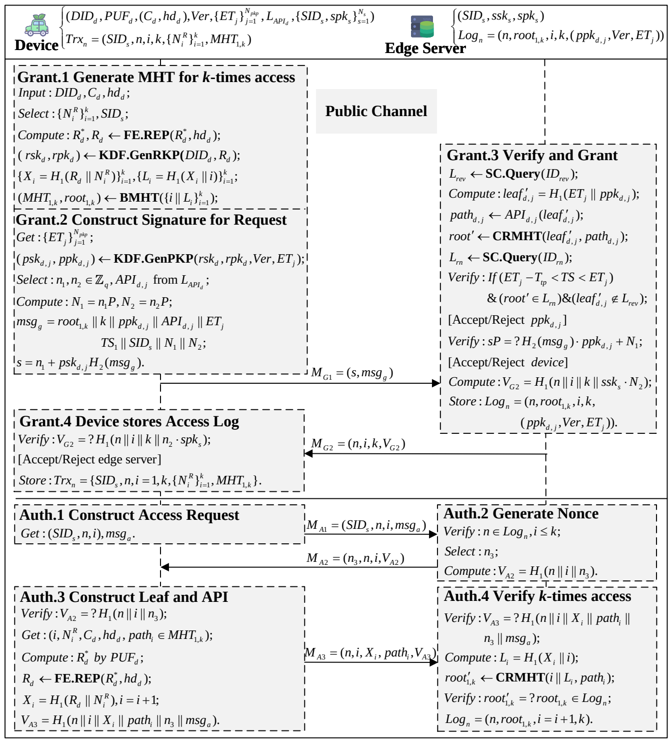
Device
Edge Server
Public Channel
Grant.1 Generate MHT for k-times access
Grant.2 Construct Signature for Request
Grant.3 Verify and Grant
Grant.4 Device stores Access Log
$M_{G1} = (s, msg_g)$
$M_{G2} = (n, i, k, V_{G2})$
Auth.1 Construct Access Request
Auth.2 Generate Nonce
Auth.3 Construct Leaf and API
Auth.4 Verify k-times access
$M_{A1} = (SID_s, n, i, msg_a)$
$M_{A2} = (n_3, n, i, V_{A2})$
$M_{A3} = (n, i, X_i, path_i, V_{A3})$

Fig. 6. Grant and authentication protocol.

storage as the indexes of pseudo-key pairs. Next, the root secret key will be further utilized to generate the time-bound pseudo-key pair as $(psk_{d,j}, ppk_{d,j}) = \mathbf{KDF.GenPKP}(rsk_d, rpk_d, Ver, ET_j)$, which is valid until the expire time ET_j. Then, the device chooses random numbers $n_1, n_2 \in \mathbb{Z}_q$ and selects $API_{d,j}$ from L_{API_d} according to $ppk_{d,j}$. After that, $N_1 = n_1 \cdot P$ and $N_2 = n_1 \cdot P$ are computed to construct the message $msg_g = root_{1,k}||k||ppk_{d,j}||API_{d,j}||ET_j||TS_1||SID_s||N_1||N_2$, where the TS_1 is the timestamp and $API_{d,j}$ is the API of $H_1(ET_j||ppk_{d,j})$. The device uses the $psk_{d,j}$ to sign the msg_g as $s = n_1 + psk_{d,j} \cdot H_2(msg_g)$. The Schnorr signature result is denoted as $M_{G1} = (s, msg_g)$. Last, the device sends M_{G1} to the edge server.

(3) **Grant.3 Verify and Grant Request**: Once receiving (s, msg_g), the edge server computes the current leaf as $leaf'_{d,j} = H_1(ET_j||ppk_{d,j})$. Then, the edge server queries the leafs revocation list $L_{rev} \leftarrow \mathbf{SC.Query}(ID_{rev})$ to verify whether the $leaf'_{d,j}$ has been revoked. If not, the edge server gets $path_{d,j}$ from $API_{d,j} = \{leaf_{d,j} : path_{d,j}\}$ inserted in the msg_g. The $leaf_{d,j}$ and $path_{d,j}$ can be further used to compute the root node $root'$. Next, the smart contract is used to query $L_{rn} = \{root_m\}_{m=1}^{\sum_{g=1}^{N_g} N_{mht_g}} \leftarrow \mathbf{SC.Query}(ID_{rn})$ to check whether the root node has been recorded in the on-chain root node list. If the ET_j has not expired, the $root'$ is included in L_{rn}, and $leaf'_{d,j}$ is not in L_{rev}, it means that the $ppk_{d,j}$ is valid. Afterward, the edge server verifies if $sP \overset{?}{=} H_2(msg_g) \cdot ppk_{d,j} + N_1$. If the equation holds, the authenticity of the device can be verified. To grant k-times accesses for the device, the edge server records the elements $Log_n = (n, root_{1,k}, i = 1, k, (ppk_{d,j}, Ver, ET_j))$ as the n-th grant log in the local storage, and returns the message $M_{G2} = (n, i = 1, k, V_{G2})$ to the device, where n is the index of Log_n and $V_{G2} = H_1(n||i||k||ssk_s \cdot N_2)$.

(4) **Grant.4 Device Stores Access Log**: After verifying $V_{G2} =? H_1(n||i||k||n_2 \cdot spk_s)$, the device locally stores $Trx_n = (SID_s, n, i = 1, k, \{N_i^R\}_{i=1}^k, MHT_{1,k})$ as the n-th k-times access transcript, where $MHT_{1,k}$ is the merkle hash tree and $\{N_i^R\}_{i=1}^k$ are random numbers.

D. Authentication

After being granted, the device is allowed to access the same edge server for k times through a public channel. The **Auth** protocol is depicted in Fig.6.

(1) **Auth.1 Construct Access Request**: The device obtains (SID_s, n, i) from the Trx_n and sends $M_{A1} = (SID_s, n, i, msg_a)$ to the edge server SID_s.

(2) **Auth.2 Generate Nonce**: The edge server checks whether $n \in Log_n$ and $i \leq k$. Then, the Nonce n_3 will be randomly selected to compute $V_{A2} = H_1(n||i||n_3)$. Last, the edge server returns $M_{A2} = (n_3, n, i, V_{A2})$ to device.

(3) **Auth.3 Construct Leaf and API**: The device verifies the validity of the received M_{A2} by checking whether $V_{A2} \overset{?}{=} H_1(n||i||n_3)$. If the equation holds, the device reads $(i, N_i^R, C_d, hd_d, path_i)$ from the local storage, where $path_i$ is obtained from the stored $MHT_{1,k}$. Then, the device derives the PUF response R_d as the step **Join.1**, to compute $X_i = H_1(R_d||N_i^R)$ and construct $(i||L_i = H_1(X_i||i))$ as the leaf node, as shown in the lower part of Fig. 5. The message $M_{A3} = (n, i, X_i, path_i, V_{A3})$ will be sent to the edge server, where $V_{A3} = H_1(n||i||X_i||path_i||n_3||msg_a)$. Last, the device updates the index in Trx_n as $i = i + 1$.

(4) **Auth.4 Verify *k*-times Access**: Upon verifying the validity of V_{A3}, the edge server finds the n-th access log to check whether the order i is correct and $i \leq k$. If so, the edge server utilizes (i, X_i) to compute the leaf $i||L_i = H_1(X_i||i)$, which can be further used to calcualte the root node $root'_{1,k}$ with $path_i$. If the root node $root_{1,k}$ stored in Log_n equals the computed $root'_{1,k}$, the device's access will be allowed. Last, the edge server updates the i in the access log Log_n as $i = i + 1$.

E. Key Update

Once the pseudo-key pairs have expired, the GM will generate new MHTs and distribute the API to each device in the group through secure channels.

(1) **KeyUpdate.1 Update Root Nodes List**: When the pseudo-key pairs of current version are about to expire, the TA will generate new synchronization information SI^* and $\{ET_j^*\}_{j=1}^{N_{pkp}^*} \leftarrow \mathbf{KDF.GenKI}(T_{start}^*, T_{end}^*, N_{pkp}^*)$. By performing the steps in **Join.3**, new pseudo-public keys $\{\{ppk_{d,j}^* = H_2(rpk_d||Ver^*||ET_j^*) \cdot rpk_{d,j}\}_{j=1}^{N_{ppk}^*}\}_{d=1}^{N_d^*}$ are computed by new elements $(Ver^*, ET_j^*, N_g^*, N_{mht_g}^*)$

to construct new partially shuffled MHTs and their root nodes $\{root^{*}_{m'}, MHT^{*}_{m'}\}_{m'=1}^{|L_{rn}|}$, , where $|L_{rn}| = \sum_{g=1}^{N_g^*} N^{*}_{mht_g}$. Next, the TA uses the smart contract $\mathbf{SC}.\mathbf{Update}(ID_{rn}, \{root^{*}_{m'}\}_{m'=1}^{|L_{rn}|})$ to update the new root node list on the blockchain. In addition, the on-chain revocation list will be set empty by $\mathbf{SC}.\mathbf{Delete}(ID_{rev})$.

(2) $\mathbf{KeyUpdate.2}$ **Request New API**: Once the device DID_d finds that all the pseudo-key pairs have expired, the device will request the GM to update the API.

(3) $\mathbf{KeyUpdate.3}$ **Distribute New API**: Upon receiving the request, the GM synchronously generates pseudo-public keys and leafs as the step $\mathbf{KeyUpdate.1}$ to obtain the new API list $L^{*}_{API_d} = \{leaf^{*}_{d,j} : path^{*}_{d,j}\}_{j=1}^{N^*_{pkp}}$ of DID_d. The GM will distribute $(Ver^*, \{ET^*_j\}_{j=1}^{N^*_{pkp}}, L^{*}_{API_d})$ to DID_d and store other API lists.

(4) $\mathbf{KeyUpdate.4}$ **Store New SI and API**: Similar to $\mathbf{Join.5}$, the device stores Ver^*, $\{ET^*_j\}_{j=1}^{N^*_{pkp}}$ and $L^{*}_{API_d}$.

$\mathbf{Remark.1}$: In $\mathbf{KeyUpdate}$ protocol, the GM will build secure channels with devices in its group, as each device has gpk_g and the GM can derive pseudo-public keys of each device. We omit the description of how to achieve authentication and key agreement (AKA) between the GM and devices.

$\mathbf{Remark.2}$: Usually, the time period of pseudo-key pairs $T_{tp} = T_{end} - T_{start}$ remains unchanged. The TA and GMs will use T_{tp} to increase T_{start} and T_{end} in each version by themselves. There is no need to update the *SI* in $\mathbf{KeyUpdate}$ protocol. If the synchronization information needs to be updated, the TA will broadcast the new SI^* to all GMs.

F. Trace

The TA and each GM will be allowed to trace real identities of devices with malicious behaviours. The tracing algorithm is performed as follows.

$\mathbf{Trace}(ppk_{t,j}, Ver, ET_j, \{DID_d, rpk_d\}_{d=1}^{N_d}) \rightarrow (DID_t)$. The TA or GM collects elements $(ppk_{t,j}, Ver, ET_j)$ from the grant log Log_n or M_{GI}. These elements are kept by the edge server in step $\mathbf{Grant.3}$ to grant the device k-times accesses. Then, the algorithm $\mathbf{KDF}.\mathbf{Trace}(ppk_{t,j}, Ver, ET_j, \{DID_d, rpk_d\}_{d=1}^{N_d})$ will be executed to get the real identity DID_t of the malicious device.

G. Revoke

Our dynamic device revocation is composed of two algorithms. The TA first utilizes the $\mathbf{Revoke.1}$ algorithm to immediately revoke the devices from the system. Then, the TA leverages the $\mathbf{Revoke.2}$ to revoke devices by removing their pseudo-public pairs from MHTs of the next version during the $\mathbf{KeyUpdate}$.

(1) $\mathbf{Revoke.1}$ **Immediate Revocation**: Once obtaining the device real identity DID_t, the TA uses rpk_d of DID_t to compute the remaining valid leafs $\{leaf_{d,j}\}_{j=r}^{N_{pkp}}$. Then, these leafs will be recorded in the on-chain revocation list by using $\mathbf{SC}.\mathbf{Add}(ID_{rev}, \{leaf_{d,j}\}_{j=r}^{N_{pkp}})$.

(2) $\mathbf{Revoke.2}$ **Permanent Revocation**: To permanently revoke the device DID_t, the TA will not generate leafs $\{H_1(ET_j||ppk_{t,j})\}_{j=1}^{N_{pkp}}$ for DID_t in $\mathbf{KeyUpdate}$. It means that $\{ppk_{t,j}\}_{j=1}^{N_{pkp}}$ of new version cannot pass the verification of the on-chain L_{rn} in $\mathbf{Grant.3}$.

V. Security Analysis

A. Security Model

First, we introduce the oracles used in our proof as follows. The challenger $\mathcal{C}$ invokes Join oracle $\mathcal{O}_{Join}$, Grant oracle $\mathcal{O}_{Grant}$, Auth oracle $\mathcal{O}_{Auth}$, Key update oracle $\mathcal{O}_{KeyUpdate}$, Revoke oracle $\mathcal{O}_{Revoke}$ on behalf of the device DID_d, the GM GID_g, or the TA.

Next, we define the feature of authentication security as follows. We denote the probability of the adversary using an unregistered device or forging a registered device to pass the verification of the edge server in $\mathbf{Grant}$ as $Pr[E_1]$, and the probability of the adversary using an ungranted device to pass the verification of the edger server in $\mathbf{Auth}$ as $Pr[E_2]$. Thus, the advantage of the probabilistic polynomial time (PPT) adversary $\mathcal{A}$ breaking the authentication security can be defined as $Adv_{\mathcal{A}}^{Auth\text{-}Sec} = Pr[E_1] + Pr[E_2]$.

Definition 1. (Authentication Security). Our proposed scheme achieves authentication security, if $Adv_{\mathcal{A}}^{Auth\text{-}Sec}$ for PPT $\mathcal{A}$ is negligible.

B. Security Proof

Lemma 1. No adversary can use an unregistered device or forge a registered device to pass the verification of the edge server in $\mathbf{Grant}$ protocol, if the hash function H_1 is collision-resistant and the Elliptic Curve Discrete Logarithm (ECDL) hardness assumption holds.

Proof. Assume that the adversary $\mathcal{A}$ can pass the verification of the edge server in $\mathbf{Grant}$ with non-negligible probability ϵ. We will construct a challenge $\mathcal{C}$ to break the collision-resistant feature of the hash function H_1 as follows:

- *Setup*: The challenger $\mathcal{C}$ performs $\mathbf{Setup}$ to publish public parameters, and register edge servers and group managers. Their identities and key pairs will be kept by $\mathcal{C}$ to answer $\mathcal{A}$'s queries.
- *Query Phase*: The adversary $\mathcal{A}$ performs the *Hash query*, *PUF query*, $\mathcal{O}_{Join}$, $\mathcal{O}_{Grant}$, $\mathcal{O}_{KeyUpdate}$, $\mathcal{O}_{Revoke}$.
  - *Hash query*: $\mathcal{A}$ uses x to ask this query. $\mathcal{C}$ returns $H_1(x)$ and adds $(x, H_1(x))$ to the hash list L_H.
  - *PUF query*: $\mathcal{A}$ uses DID_d and C_d to ask this query. Then, $\mathcal{C}$ selects a random value as R_d, and returns R_d. The tuple (DID_d, C_d, R_d) will be added to L_{puf}.
  - $\mathcal{O}_{Join}$: After $\mathcal{C}$ honestly answers $\mathcal{A}$'s queries according to $\mathbf{Join}$, $\mathcal{C}$ adds each device's (DID_d, C_d, R_d) to the PUF list L_{puf}, and each device's key pair (DID_d, rsk_d, rpk_d) to the device key pair list L_{kp}. Besides, $\mathcal{C}$ generates $\{leaf_l\}_{l=1}^{N_{leaf}}$, $N_{leaf} = N_d N_{pkp}$ for each group. Each $leaf_l$ has a correlated $API_l = \{leaf_l : path_l\}$, which maps to a certain $API_{d,j} = \{leaf_{d,j} : path_{d,j}\}$. $\mathcal{C}$ records each $ppk_{d,j}$ and $API_{d,j}$ in the g-th *API* list L^{g}_{API} for each group GID_g. We use $L_{API} = \{L^{g}_{API}\}_{g=1}^{N_g}$ to denote the API list for all groups. The newest *SI* will be added to the list L_{SI}.

In addition, $\mathcal{C}$ keeps a on-chain root node list L_{rn} by taking the steps in **Join.3**.

- $\mathcal{O}_{Grant}$: $\mathcal{C}$ honestly answers $\mathcal{A}$'s queries, according to **Grant** protocol and L_{puf}, L_{kp}, L_{SI}, L_{rn}, and L_{rev}, where L_{rev} is built by $\mathcal{C}$ in $\mathcal{O}_{Revoke}$.
- $\mathcal{O}_{KeyUpdate}$: After $\mathcal{C}$ honestly answers $\mathcal{A}$'s queries according to **KeyUpdate** protocol, $\mathcal{C}$ updates L_{SI} and the API list $L_{API} = \{L_{API}^{g}\}_{g=1}^{N_g}$ for all N_g groups.
- $\mathcal{O}_{Revoke}$: To answer $\mathcal{A}$'s query of revoking DID_t, $\mathcal{C}$ adds remaining valid leafs of DID_t to L_{rev}, and removes the key pair of DID_t from L_{kp} and L_{puf}.

- *Challenge Phase*: $\mathcal{A}$ selects a DID_d to ask $\mathcal{O}_{Grant}$. First, $\mathcal{A}$ asks *PUF query* to get the PUF response from L_{puf}. If $DID_d \notin L_{puf}$, $\mathcal{C}$ returns `null`. Else if $DID_d \in L_{puf}$ and $DID_d \notin L_{kp}$, $\mathcal{C}$ returns (C_d, R_d). Otherwise, $\mathcal{C}$ returns $\perp$. Then, $\mathcal{A}$ is allowed to query the L_{API} and L_{SI} to get every $ppk_{d,j}$, $API_l = (leaf_l : path_l)$ and the *SI*. After that, let $\mathcal{A}$ continue to participate in **Grant** protocol.
- *Output*: $\mathcal{A}$ outpus Schnorr signature $\sigma = (s, msg)$, where $msg = root_{1,k}||k||ppk^*||API^*||ET_j||TS_1||SID_s||N_1||N_2$ and $API^* = (leaf^* : path^*)$. $\mathcal{A}$ wins the security game if: (1) $API^* = (leaf^* : path^*)$ is considered valid by the edge server. (2) The signature σ is verified.

Finally, $\mathcal{A}$ wins the game. We will show how to break the collision-resistant feature of H_1. We set $path[i,j]$ as the sub-path of $path[h,0]$ from the i-depth node up to the j-depth node, where $i,j \in [0,h], i \geq j$ and h denotes the height of MHT (the depth of leafs is h and the depth of the root is 0). Note that $path[i]$ is the i-depth node of $path[i,j]$, and $leaf_{[i,j]}$ is the leaf corresponding to $path[i,j]$ in our partially shuffled MHT. The $API^l = (leaf^l_{[h,0]}, path^l[h,0]) \in L_{API}$ is the l-th *API* belonging to L_{API}. There exist two cases of $\mathcal{A}$ wining the security game.

- *Case 1*: In this case, $\mathcal{C}$ finds that σ is valid, and $\mathcal{A}$ forges a valid path $path^*$ for $ppk^* \notin L_{API}$. It means that the $root^*$ of $path^*$ in our partially shuffled MHT satisfies: $root^* = root^l \in L_{rn}$, where $root^l$ is the root of $path^l \in L_{API}$. As $ppk^* \neq ppk_{d,j}$, $leaf^*_{[h,0]} = H_1(ET_j||ppk^*)$ and $leaf^d_{[h,0]} = H_1(ET_j||ppk_{d,j})$, there exist the following two conditions. We assume that the advantage of $\mathcal{A}$ wining in these two conditions is respectively ϵ_1 and ϵ_2.
  - *Condition 1*: If $leaf^*_{[h,0]}$=$leaf^d_{[h,0]}$, the hash collision of H_1 can be directly found by these two leafs with the probability ϵ_1.
  - *Condition 2*: If $leaf^*_{[h,0]} \neq leaf^d_{[h,0]}$, by $root^* = root^l$, there must exists $i \in [0, h-1]$ such that $leaf^*_{[i,0]} = leaf^d_{[i,0]}$ and $leaf^*_{[i+1,0]} \neq leaf^d_{[i+1,0]}$. The leafs $leaf^*_{[i,0]}$ and $leaf^d_{[i,0]}$ can be respectively computed as $leaf^*_{[i,0]} = H_1(leaf^*_{[i+1,0]}||path^*[i+1])$ and $leaf^d_{[i,0]} = H_1(leaf^d_{[i+1,0]}||path^l[i+1])$. Hence, we can find the collision of H_1 with the probability ϵ_2, according to the H_1's hash value $leaf^*_{[i,0]} = leaf^d_{[i,0]}$.

  Therefore, the advantage of $\mathcal{A}$ finding the collision of H_1 in this case is $\epsilon_1 + \epsilon_2$.
- *Case 2*: $\mathcal{A}$ forges a valid signature for *msg*, which can be verified by any $ppk_{d,j} \in L_{API}$. It means that $\mathcal{A}$ breaks the Schnorr signature. Since the security of Schnorr signature can be reduced to the hardness assumption of ECDL problem by the forking lemma as proved in [18], the advantage ϵ_3 of $\mathcal{A}$ is negligible.

Based on the above analysis, the advantage $\epsilon = \epsilon_1 + \epsilon_2 + \epsilon_3$ of $\mathcal{A}$ wining the security game is non-negligible. Since ϵ_3 is negligible, the advantage $\epsilon_1 + \epsilon_2$ of breaking the collision-resistant feature of H_1 is non-negligible. It contradicts to the security assumption of H_1. *Lemma* 1 is proved. □

Lemma 2. No adversary can use an ungranted device to pass the verification of the edge server in **Auth**, if the hash function H_1 is collision-resistant and the PUF is secure.

Proof. Assume that there exists an ungranted device passing the verification of the edge server in **Auth** with non-negligible probability ϵ. We will construct a challenger $\mathcal{C}$ to break the collision-resistant feature of the hash function H_1 as follows:

- *Setup*: $\mathcal{C}$ performs **Setup** as *Lemma* 1.
- *Query Phase*: $\mathcal{A}$ performs the *Hash query*, $\mathcal{O}_{Join}$, $\mathcal{O}_{Grant}$, $\mathcal{O}_{KeyUpdate}$, and $\mathcal{O}_{Revoke}$ as *Lemma* 1. We only explain the main differences and the additional $\mathcal{O}_{Auth}$.
  - $\mathcal{O}_{Grant}$: The difference is that $\mathcal{C}$ will maintain a access log list L_{AL} for each edge server, which contains the access log for n-the grant $Log_n = (n, root_{1,k}, i, k, (ppk_{d,j}, Ver, ET_j))$. $\mathcal{C}$ also maintains a access transcript list L_{Trx} to record the access transcript $Trx_n = (SID_s, n, i, k, \{N_i^R\}_{i=1}^{k}, MHT_{1,k})$ constructed by each granted device.
  - $\mathcal{O}_{Auth}$: $\mathcal{C}$ honestly answers $\mathcal{A}$'s queries according to **Auth** protocol. $\mathcal{C}$ maintains a API history list L_{API} to record each $API_n^i = (n, i, X_i, path_i)$ and a random number list L_N to store each N_i^R.
- *Challenge Phase*: $\mathcal{A}$ selects DID_d to perform $\mathcal{O}_{Auth}$. If $DID_d \in L_{kp}$, $\mathcal{C}$ returns $\perp$. Else if $DID_d \in L_{puf}$, $\mathcal{C}$ returns (C_d, R_d). Otherwise, returns `null`. $\mathcal{A}$ can also query L_{API} to get history API_n^i, query L_{Trx} to get Trx_n, query L_{AL} to get Log_n, and query L_N to get N_i^R.
- *Output*: $\mathcal{A}$ outputs $(n, i, X_i^*, path_i^*, V_{A3})$, and wins the game if: $(n, i, X_i^*, path_i^*, V_{A3})$ can pass the verification.

Finally, once $\mathcal{A}$ accesses the edge server successfully and wins the security game, we will show how to break the collision-resistant feature of H_1. Assume that the advantage of $\mathcal{A}$ wining is ϵ. $\mathcal{A}$ forges the valid i-th $leaf^*_{i[h,0]}$ and the correlated $path_i^*[h,0]$. It means that the $root^*_{1,k}$ corresponding to $path_i^*[h,0]$ that belongs to the MHT for k-times accesses will satisfy: $root^*_{1,k} = root_{1,k}$, where $root_{1,k}$ is the root of $(leaf_{i[h,0]}, path_i[h,0]) \in L_{API}$. Two leafs $leaf^*_{i[h,0]}$ and $leaf_{i[h,0]}$ can be respectively computed as $i||L_i^* = H_1(X_i^*||i)$ and $i||L_i = H_1(X_i||i)$, where $X_i = H_1(R_d||N_i^R)$ and $X_i^* = H_1(R_d{}^*||N_i^R)$. Since the PUF is secure, $\mathcal{A}$ cannot obtain the correct R_d, then we get $R_d \neq R_d{}^*$. As a result, there also exist two cases of $\mathcal{A}$ wining the game. We can prove that if $\mathcal{A}$ wins, $\mathcal{C}$ would find the hash collision of H_1 with the probability ϵ. The proof procedure is similar to the two cases in *Lemma* 1. To be concise, we omit detailed descriptions.

However, the advantage of $\mathcal{A}$ wining the security game ϵ is non-negligible. It contradicts to the security assumption of H_1. *Lemma* 2 is proved. □

Theorem 1. Our proposed scheme achieves authentication security, if the hash function H_1 is collision-resistant, the ECDL hardness assumption holds, and the PUF is secure.

Proof. Based on the proof in *Lemma* 1 and *Lemma* 2, no PPT adversary can use an unregistered device or forge a registered device to pass the verification of the edge server in **Grant** protocol, or use an ungranted device to pass the verification of the edge server in **Auth** protocol. □

C. Security Properties

(1) Conditional anonymity: The device in our scheme periodically utilizes pseudo-key pairs to authenticate with the edge server in **Grant**. On the one hand, these pseudo-key pairs hide the information of the device's root public key rpk_d as well as the information about the real device identity. On the other hand, only the device's GM or the TA can use Log_n or M_{GI} to trace the real identity $DID_t \leftarrow \mathbf{Trace}(ppk_{t,j}, Ver, ET_j, \{DID_d, pk_d\}_{d=1}^{N_d})$.

(2) Periodical unlinkability: In our scheme, the grant requests achieve periodical unlinkability, as each pseudo-key pair will be used by its device in one certain time interval and have an expire time ET_j. The j-th pseudo-key pair of DID_d is computed as $ppk_{d,j} = rsk_d \cdot H_2(rpk_d||Ver||ET_j) \cdot P$. It is hard to link $ppk_{d,j}$ and $ppk_{d,j+1}$ to DID_d.

In addition, the devices from the same group will use N_{mht_g} fixed on-chain root nodes to get the grant. The adversary can only link grant requests to one certain group, according to leafs and API from the group's specific root nodes. As MHTs of these root nodes are constructed by shuffled leafs $leaf_{d,j} = H_1(ET_j||ppk_{d,j})$, it is different for the adversary from public channels or the semi-honest edge server to link two pseudo-public keys in different time intervals to the same device, according to leafs and API.

Furthermore, the device revocation and addition will not reveal linkable information. For revocation, there is no need to revoke expire pseudo-key pairs. The adversary cannot link previous grant requests to the revoked device, as mentioned above. Recoding unused leafs on the blockchain also will not break the unlinkability. For addition, leafs of newly added devices will be aggregated and then shuffled, as explained in **MHT**.**BTT** and the remark of **Join** protocol.

The feature of periodic unlinkability provided in our scheme has the same privacy-preserving ability with EBCPA [9] and BCPPA [10]. However, the subsequent k-times access with the edge server will be linkable, as mentioned in the definition of the relaxed anonymity in work [4].

(3) Resistant to malicious impersonation: As has been proved in Theorem 1, the adversary cannot use an unregistered device or forge a registered device to get the k-times access grant. In addition, the adversary also cannot utilize an ungranted device to access the edge server. Hence, our scheme effectively resists the malicious impersonation.

(4) Resistant to replay attack: We add the steps **Auth**.**1** and **Auth**.**2** to realize an additional interaction to resist replay attack. The random number mechanism inserts n_3 to bind each access request. Therefore, the adversary cannot directly replay M_{A3} to pass the verification.

(5) Resistant to physical and cloning attacks: Assume that devices are lost or stolen and the adversary can access the device's local storage using physical attacks. However, the PUF circuit is unclonable, and the PUF response is random and unpredictable. The adversary cannot derive the correct PUF response R_d to generate rsk_d or $X_i = H_1(R_d||N_i^R)$ to pass the verification of the edge server in **Grant** or **Auth**.

VI. Performance Evaluation

A. Experimental Settings

A proof-of-concept system was constructed by implementing the XOR-APUFs circuit on Xilinx Virtex-5 FPGA and the blockchain network using Hyperledger fabric. All the involved entities were constructed by Java. To collect the data at 128-bit security level, the secp256r1 curve and Type F pairing with the 256-bit order, and the SHA-256 hash function were selected to implement cryptographic operations.

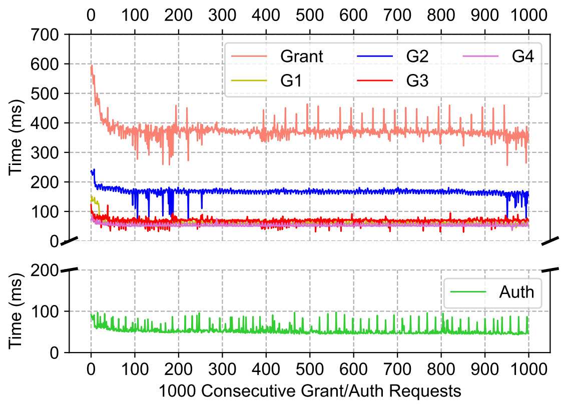

Fig. 7. Actual running time of 1000 consecutive Grant/Auth requests.

B. Efficiency

We first proposed 1000 consecutive distinct **Grant**/**Auth** requests to evaluate the actual running time. The height of two kinds of MHTs is all set as 7. It means that each MHT has 128 leafs. Besides, the number of pseudo-key pairs of the device in each time period is N_{pkp} (=128).

As is shown in Fig.7, it costs around 373 ms to finish the **Grant** protocol, where the step **Grant**.**2** takes up most of the time. Because the device needs to perform several scalar multiplication on $\mathbb{G}$ in **KDF**.**GenPKP**($\approx$ 53.42 ms) and the Schnorr signature. The cryptographic operations involved in our scheme are not computationally intensive for the edge server. The most time-consumption operation is to query the smart contract T_{sc}($\approx$ 29.48 ms) twice to obtain the root node list and the leafs revocation list. The **Auth** protocol only needs to cost around 53 ms. Because the device only performs few hash functions and the PUF operation. Besides, algorithms of **BMHT** for the device ($\approx$ 2.88 ms) and **CRMHT** for the edge server ($\approx$ 0.08 ms) are all efficient.

Then, the average time consumption of different protocols and algorithms is depicted in Fig. 8. It takes about 10,241 ms to register 10 devices. The time of **Grant** seems a bit

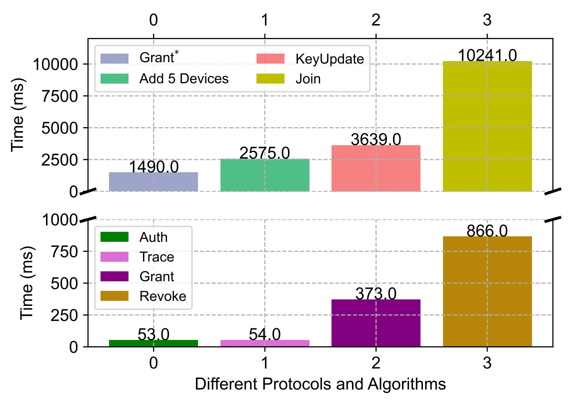

Fig. 8. Average time consumption of different protocols and algorithms.

long, as the TA first needs to generate root nodes (≈ 2651 ms) and the GM then has to build API **Gen.API** (≈ 4813 ms) for group's devices. The time of adding 5 devices is 2575 ms, and updating pseudo-key pairs **KeyUpdate** in each time period for 15 devices costs about 3639 ms. When pseudo-key pairs are expired, it takes the device 1490 ms (**Grant***) in all to retrieve valid API from the GM and get the grant of the k-times access from the edge server. The time of revoking one device is 866 ms (**Revoke**), which includes the time of invoking the smart contract (327 ms). In **Trace**, the TA takes 54 ms to get the real identity from 15 devices.

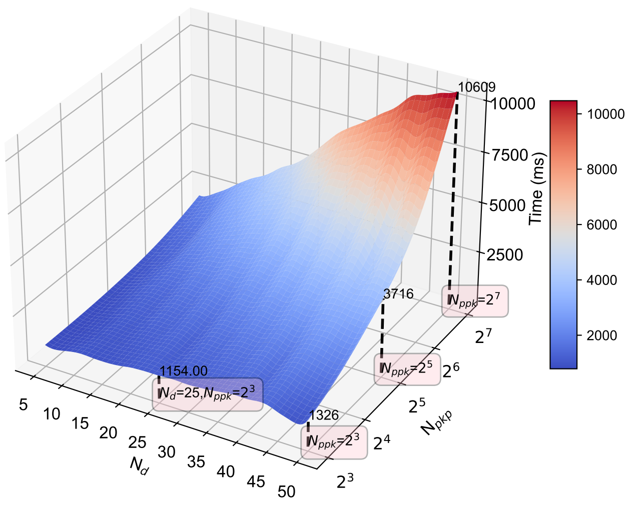

Fig. 9. The actual running time of **KeyUpdate** at different settings. N_d is the device number, and N_{pkp} is the number of pseudo-key pairs.

Next, we choose different device number N_d and pseudo-key pairs number N_{pkp} to evaluate the running time of **KeyUpdate** at different settings (N_d, N_{pkp}). As is shown in Fig. 9, under the settings of (25, 2^3), (50, 2^7), (50, 2^5), and (50, 2^7), the time of updating is respectively 1154 ms, 1326 ms, 3716 ms, and 10,609 ms. It is indicated that: (a) N_d and N_{pkp} all have the positive influence on the time cost. (b) The time has an obvious step-wise structure with the growth of N_{pkp}. To balance the efficiency and the privacy-preserving ability, the N_{pkp} should be properly set according to the privacy-preserving demands of different MEC services.

In conclusion, the actual running time of our scheme is efficient under proper settings.

C. Comparison of Computation Overhead

We compare the computation overhead of work with relevant studies [9], [10], [11], [4] by counting the most time-consumption cryptographic operations. The average running time of these operations is shown at the bottom of Table I.

In **Grant**, the device takes about 1.05 $\times 2^t$+383.4 ms in our work and costs 0.7 $\times 2^t$+19,629 ms in [4] . If the height ($H_{mht} = t$) of MHTs is reasonable (e.g., t=7), the computation overhead of our work is much less than [4], as performing four bilinear pairing required in [4] is very heavy for MEC devices. Similarly, the computation overhead (0.01$\times t$+71.78 ms) on the edge server of our work is much less than [4] (996.55 ms). In **Auth**, the relevant studies [9], [10], [11] cost the device 183.14 ms, 182.32 ms, and 771.04 ms respectively. In [4], the device directly transmits the leaf node and API and takes the least overhead, since no cryptographic operations are performed. To additionally resist the replay attack, the Nonce n_3 is inserted in our scheme, which brings three more hash operations than [4]. For the edge server, the work [9], [10], [11], [4] will spend 38.05 ms, 46.66 ms, 401.23 ms, and 0.01$\times t$ ms during the authentication. The work [4] and our work take much less computation overhead on the edge server. The overhead of our work (0.01$\times t$+0.03 ms) is very close to [4] and the difference is only three hash operations.

Compared with the state-of-the-art studies, although our work needs one **Grant** before authentication, the computation of subsequent k-times accesses in **Auth** is lightweight and definitely improves the authentication efficiency. Moreover, our **Grant** is also efficient and involves no time-consumption operation. Obviously, the computation overhead of our scheme has the performance advantage in MEC applications, which need continuous accesses with the same edge server.

D. Comparison of Communication Overhead

The communication overhead is measured by the bit length of transmitted messages. The bit length of each element in messages is shown at the bottom of Table II. In **Grant**, the communication overhead of [4] is 3$|\mathbb{G}_1|$+$|\mathbb{G}_2|$+7$|H|$ (=4352 bits) and our work is 7$|M_1|$+$|\mathbb{Z}_q|$+3$|\mathbb{G}|$+(t+2)$|H|$ (=2528+256$\times t$ bits). If $t \leq 7$, our work takes less overhead. In **Auth**, schemes [9], [10], [11] respectively take 2848 bits, 1440 bits, and 5376 bits. The communication overhead of our work (1120+256 $\times t$ ms) and [4] (256 $\times t$ ms) is linear to the height of MHTs. If $t \leq 7$, the overhead of our work is less than [2] and [11], and is only 64 bits more than [10].

Therefore, the brought communication overhead is also efficient and practical under the proper setting of the MHT height. According to the above comparison, it is recommended to set the height of MHT as 7.

TABLE I
COMPARISONS OF COMPUTATION OVERHEAD AT 128-BIT SECURITY LEVEL WITH t-HEIGHT MHTS

	Grant		**Auth**	
	Device	Edge Server	Device	Edge Server
EBCPA [9]	\	\	$3T_{\mathbb{G}^{sm}}+2T_{\mathbb{G}^{pa}}+T_h+2T_{mm}+T_{sc} \approx 183.14$ ms	$2T_{\mathbb{G}^{sm}}+4T_{\mathbb{G}^{pa}}+T_h+T_{sc} \approx 38.05$ ms
BCPPA [10]	\	\	$3T_{\mathbb{G}^{sm}}+T_{mi}+T_{mm}+T_H+T_{sc} \approx 182.32$ ms	$4T_{\mathbb{G}^{sm}}+3T_{\mathbb{G}^{pa}}+2T_{mi}+4T_{mm}+T_h$ $+T_{sc} \approx 46.66$ ms
Huang et al. [11]	\	\	$8T_{\mathbb{G}_1^{sm}} \approx 771.04$ ms	$T_{\mathbb{G}_T^{pa}}+3T_{bp} \approx 401.23$ ms
Zhang et al. [4]	$6T_{\mathbb{G}_1^{sm}}+2T_{\mathbb{G}_2^{sm}}+4T_{\mathbb{G}_T^{sm}}$ $+T_{enc}+2T_{\mathbb{G}_1^{pa}}+3T_{\mathbb{G}_T^{pa}}$ $+(2^{t+1}+1)T_h+4T_{bp}$ $\approx 0.7\times 2^t+19{,}629$ ms	$T_{dec}+6T_{bp}+4T_{\mathbb{G}_1^{sm}}+2T_{\mathbb{G}_2^{sm}}$ $+5T_{\mathbb{G}_T^{sm}}+2T_{\mathbb{G}_1^{pa}}+2T_{\mathbb{G}_2^{pa}}$ $+5T_{\mathbb{G}_T^{pa}}+2T_h$ ≈ 996.55 ms	\	$tT_h \approx 0.01 \times t$ ms
Our scheme	$T_{puf}+T_{rep}+5T_{\mathbb{G}^{sm}}+3(2^t+1)T_h$ $\approx 1.05\times 2^t+383.4$ ms	$(t+3)T_h+3T_{\mathbb{G}^{sm}}+T_{\mathbb{G}^{pa}}+2T_{sc}$ $\approx 0.01 \times t+71.78$ ms	$T_{puf}+T_{rep}+3T_h \approx 129.45$ ms	$(t+3)T_h \approx 0.01 \times t+0.03$ ms

*In secp256r1, the computation time on $\mathbb{G}$ includes: scalar multiplication time $T_{\mathbb{G}^{sm}}$ and point addition time $T_{\mathbb{G}^{pa}}$. In Type F pairing, the computation time on $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ includes: scalar multiplication time ($T_{\mathbb{G}_1^{sm}}$, $T_{\mathbb{G}_2^{sm}}$, $T_{\mathbb{G}_T^{sm}}$), point addition time ($T_{\mathbb{G}_1^{pa}}$, $T_{\mathbb{G}_2^{pa}}$, $T_{\mathbb{G}_T^{pa}}$), bilinear pairing time T_{bp}. We also denote the time of other cryptographic operations as follows: T_h denotes the hash function time, T_{mm} and T_{mi} are the modular multiplication and inversion time in $\mathbb{Z}_q$. T_{puf} and T_{rep} are the generation time of the PUF response. T_{enc} and T_{dec} are the AES encryption and decryption time. T_{sc} is the querying smart contract time.
*For devices, $T_{\mathbb{G}^{sm}}$(≈50.79 ms), $T_{\mathbb{G}^{pa}}$(≈0.26 ms), $T_{\mathbb{G}_1^{sm}}$(≈96.38 ms), $T_{\mathbb{G}_2^{sm}}$(≈197.65 ms), $T_{\mathbb{G}_T^{sm}}$(≈860.57 ms), $T_{\mathbb{G}_1^{pa}}$(≈0.41 ms), $T_{\mathbb{G}_2^{pa}}$(≈0.64 ms), $T_{\mathbb{G}_T^{pa}}$(≈3.87 ms), T_{bp}(≈3799.63 ms), T_h(≈0.35 ms), T_{mm}(≈0.21 ms), T_{mi}(≈1.41 ms), T_{puf}(≈0.028 ms), T_{rep}(≈128.37 ms), T_{enc}(≈2.03 ms), T_{dec}(≈1.20 ms), T_{sc}(≈29.48 ms). For edge servers, $T_{\mathbb{G}^{sm}}$(≈4.26 ms), $T_{\mathbb{G}^{pa}}$(≈0.01 ms), $T_{\mathbb{G}_1^{sm}}$(≈5.78 ms), $T_{\mathbb{G}_2^{sm}}$(≈9.73 ms), $T_{\mathbb{G}_T^{sm}}$(≈30.2 ms), $T_{\mathbb{G}_1^{pa}}$(≈0.01 ms), $T_{\mathbb{G}_2^{pa}}$(≈0.03 ms), $T_{\mathbb{G}_T^{pa}}$(≈0.13 ms), T_{bp}(≈133.7 ms), T_h(≈0.01 ms), T_{mm}(≈0.006 ms), T_{mi}(≈0.04 ms), T_{enc}(≈0.06 ms), T_{dec}(≈0.02 ms), T_{sc}(≈29.48 ms).

TABLE II
COMPARISONS OF COMMUNICATION OVERHEAD AT 128-BIT SECURITY LEVEL WITH t-HEIGHT MHTS

	Grant	**Auth**
[9]	\	$4\lvert\mathbb{G}\rvert+3\lvert\mathbb{Z}_q\rvert+\lvert M_1\rvert = 2848$ bits
[10]	\	$3\lvert M_2\rvert+4\lvert\mathbb{Z}_q\rvert+\lvert M_1\rvert = 1440$ bits
[11]	\	$7\lvert\mathbb{Z}_q\rvert+7\lvert\mathbb{G}\rvert = 5376$ bits
[4]	$3\lvert\mathbb{G}_1\rvert+\lvert\mathbb{G}_2\rvert+7\lvert\mathbb{Z}_q\rvert = 4352$ bits	$7\lvert H\rvert = 256 \times t$ bits
Our Scheme	$7\lvert M_1\rvert+\lvert\mathbb{Z}_q\rvert+3\lvert\mathbb{G}\rvert+(t+2)\lvert H\rvert$ =(2528+256×t) bits	$7\lvert M_1\rvert+\lvert M_2\rvert+(t+3)\lvert H\rvert$ =(1120+256×t) bits

The length of the hash function is $\lvert H\rvert$(=256 bits). The length of elements on $\mathbb{Z}_q$, $\mathbb{G}$, $\mathbb{G}_1$, $\mathbb{G}_2$ is $\lvert\mathbb{Z}_q\rvert$(=256 bits), $\lvert\mathbb{G}\rvert$(=512 bits), $\lvert\mathbb{G}_1\rvert$(=512 bits), and $\lvert\mathbb{G}_2\rvert$(=1024 bits). $\lvert M_1\rvert$=(32 bits) denotes the length of n, i, k, TS, SID. $\lvert M_2\rvert$(=128 bits) is the length of the Nonce n_3 and the transaction identity.

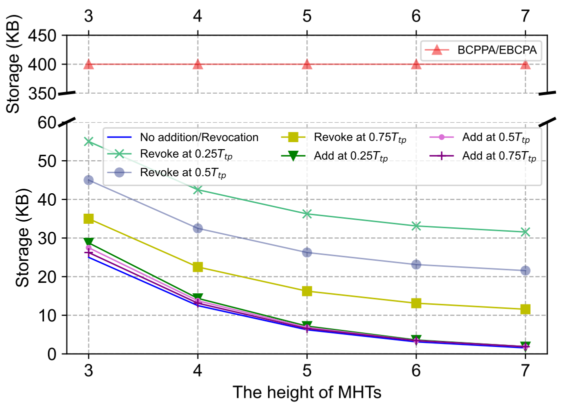

Fig. 10. The on-chain storage overhead brought by **KeyUpdate** in each time period (T_{tp}) at settings of N_d=50, N_{pkp}=128, N_{tmp}=10.

E. Comparison of Storage Overhead

As is demonstrated in Fig. 10, we first compare the on-chain storage overhead with [9], [10]. The device number N_d, the pseudo-key pairs number N_{pkp}, and N_{tmp} in **MHT.BTT** are set as 50, 128, and 10. The studies [9], [10] will bring 400 KB on-chain storage overhead in each time period. Our work greatly reduces this storage size by multiples of 2^t, where t is the height of MHTs. Because pseudo-public keys are constructed as leafs, and recorded on the blockchain in the form of root nodes. Although adding/revoking devices during the time period will increase the on-chain storage size by recording revoked leafs or new root nodes on the blockchain, the brought overhead is relatively small and will not influence the satisfactory on-chain storage performance of our work.

Then, we evaluate the storage overhead on the resource-constrained MEC device. The storage brought to the device is (9.98N_{trx}+0.06N_s+8.5) KB, where N_s is the edge server number and N_{trx} is the number of logs granted by different edge servers. The storage overhead on the device is not very lightweight, but is still can be accepted by MEC devices. The lightweight k-times accesses will sacrifice a small portion of the device's storage resources.

In conclusion, our work greatly improve the efficiency by reducing the on-chain storage overhead. Moreover, the storage brought to resource-constrained MEC devices is reasonable.

VII. CONCLUSION AND FUTURE WORK

We proposed an efficiency-improved and conditional privacy-preserving authentication scheme based on MHT in MEC. We prove the authentication security and discuss other security features and functionalities. The experiment evaluation shows that it costs around 373 ms to get the grant of k-times accesses from distributed edge server and only speeds 53 ms to finish the authentication with the same edge server. It also takes 3639 ms for key update, 2574 ms for device addition, 866 ms for device revocation, 54 ms for device tracing. Although the device joining and the distribution of API will take a bit long time, the execution frequency of these operations is low. The actual running time of our scheme would be efficient under proper settings. The comparisons indicate that it is recommended to set the height of MHT as 7, and our work's computation and communication overhead are satisfactory and suitable for MEC services. Moreover, our scheme greatly reduces the on-chain storage overhead.

Although our authentication scheme effectively improves the efficiency, it still remains challenging to adapt to real-world MEC applications with dynamic system architectures. In the future, we will further improve the flexibility by specially designing a authentication protocol tailored for MEC devices to build secure channels with GMs or TAs in different trust domains. In addition, to better fit lightweight MEC devices, we plan to replace the elliptic curve cryptography performed on the device side with only the hash function.

References

[1] P. Cruz, N. Achir, and A. C. Viana, "On the edge of the deployment: A survey on multi-access edge computing," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–34, 2022.

[2] L. Kong, J. Tan, J. Huang, G. Chen, S. Wang, X. Jin, P. Zeng, M. Khan, and S. K. Das, "Edge-computing-driven internet of things: A survey," *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–41, 2022.

[3] P. Ranaweera, A. D. Jurcut, and M. Liyanage, "Survey on multi-access edge computing security and privacy," *IEEE Communications Surveys and Tutorials*, vol. 23, no. 2, pp. 1078–1124, 2021.

[4] Q. Zhang, J. Wu, H. Zhong, D. He, and J. Cui, "Efficient anonymous authentication based on physically unclonable function in industrial internet of things," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 233–247, 2023.

[5] J. Huang, W. Susilo, F. Guo, G. Wu, Z. Zhao, and Q. Huang, "An anonymous authentication system for pay-as-you-go cloud computing**," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 1280–1291, 2022.

[6] P. Fu, J. Wu, X. Lin, and A. Shen, "Ztei: Zero-trust and edge intelligence empowered continuous authentication for satellite networks," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 2376–2381.

[7] Y. Zhang, B. Li, B. Liu, Y. Hu, and H. Zheng, "A privacy-aware pufs-based multiserver authentication protocol in cloud-edge iot systems using blockchain," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13 958–13 974, 2021.

[8] P. Gope, J. Lee, and T. Q. S. Quek, "Lightweight and practical anonymous authentication protocol for rfid systems using physically unclonable functions," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2831–2843, 2018.

[9] C. Lin, X. Huang, and D. He, "Ebcpa: Efficient blockchain-based conditional privacy-preserving authentication for vanets," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 3, pp. 1818–1832, 2022.

[10] C. Lin, D. He, X. Huang, N. Kumar, and K.-K. R. Choo, "Bcppa: A blockchain-based conditional privacy-preserving authentication protocol for vehicular ad hoc networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 12, pp. 7408–7420, 2021.

[11] H. Huang, Y. Wu, F. Xiao, and R. Malekian, "An efficient signature scheme based on mobile edge computing in the ndn-iot environment," *IEEE Transactions on Computational Social Systems*, vol. 8, no. 5, pp. 1108–1120, 2021.

[12] M. Luo and Y. Zhou, "An efficient conditional privacy-preserving authentication protocol based on generalized ring signcryption for vanets," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 9, pp. 10 001–10 015, 2022.

[13] M. Buser, J. K. Liu, R. Steinfeld, A. Sakzad, and S.-F. Sun, "Dgm: Ad ynamic and revocable group merkle signature," in *Computer Security–ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I 24*. Springer, 2019, pp. 194–214.

[14] J. Zhang, Y. Jiang, J. Cui, D. He, I. Bolodurina, and H. Zhong, "Dbcpa: Dual blockchain-assisted conditional privacy-preserving authentication framework and protocol for vehicular ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 2, pp. 1127–1141, 2022.

[15] J. Wang, L. Wu, K.-K. R. Choo, and D. He, "Blockchain-based anonymous authentication with key management for smart grid edge computing infrastructure," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1984–1992, 2020.

[16] Y. Yang, W. Xue, J. Sun, G. Yang, Y. Li, H. Hwa Pang, and R. H. Deng, "Pkt-sin: A secure communication protocol for space information networks with periodic k-time anonymous authentication," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 6097–6112, 2024.

[17] Y. Zhang, B. Li, B. Liu, and J. Chang, "Building puf as a service: Distributed authentication and recoverable data sharing with multidimensional crps security protection," *IEEE Internet of Things Journal*, vol. 11, no. 10, pp. 17 301–17 316, 2024.

[18] D. Pointcheval and J. Stern, "Security arguments for digital signatures and blind signatures," *Journal of cryptology*, vol. 13, pp. 361–396, 2000.