Building PUF as a Service: Distributed Authentication and Recoverable Data Sharing with Multidimensional CRPs Security Protection

Yan Zhang, Bing Li, Bo Liu, *Senior Member, IEEE*, Jinke Chang

***Abstract*—Physically Unclonable Functions (PUFs) have emerged as hardware fingerprints for IoT devices in the form of challenge-response pairs (CRPs). This mapping behaviour is regarded as a physically secure primitive, activating mechanisms of authentication and data protection. However, multidimensional security threats to CRPs, including impersonation attacks, availability attacks, machine learning attacks, and single point failure, impede the applications of PUFs technology. To simultaneously solve these threats, this paper not only leverages Shamir secret sharing (SSS) to provide comprehensive CRPs protection, but also integrates blockchain to address trust issues of synchronization, supervision, and deployment brought by the SSS system. Specifically, we first propose a security-enhanced and reliable CRPs management method. This method leverages SSS and its homomorphic addition feature to protect CRPs storage, sharing, and backup processes. Meanwhile, blockchain is involved in the SSS system to synchronize CRPs and supervise sharing behaviours. Then, a PUF-as-a-service (PaaS) framework is constructed, which utilizes blockchain to trace the change of the SSS system and integrate different PUFs-based security mechanisms. Once deployed in PaaS, users can always utilize transactions to build secure on-chain channels with SSS system and employ the PUF service. Based on our CRPs management method and PaaS framework, we successfully constructed PUFs-based distributed authentication and recoverable data sharing with multidimensional CRPs protection. The security proof and discussions of our scheme are also provided. Moreover, a proof-of-concept prototype was implemented to conduct experimental evaluations and comparative analysis. The results and additional discussions demonstrate that our work is efficient, practical, and suitable for IoT deployment.**

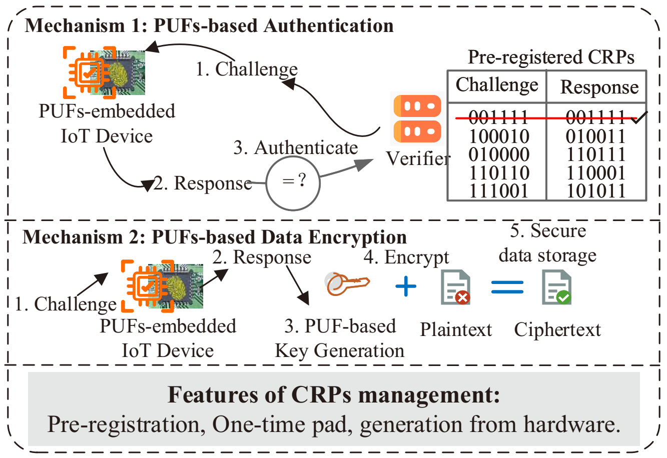
***Index Terms*—Physically Unclonable Functions, blockchain, Shamir secret sharing, Internet of Things (IoT), authentication and data sharing, CRPs security protection, security and privacy.**

I. Introduction

PHYSICALLY Unclonable Functions (PUFs) have recently emerged as a promising and critical security technology for Internet of Things (IoT) [1]. In general, a PUF physical system is embedded in IoT devices to present a challenge-responses behaviour $\gamma : \{0,1\}^n \to \{0,1\}^m$. A set of n-bit challenges are mapped into m-bit responses. As PUFs are instanced by deriving micro-scale or nano-scale manufacturing violations of integrated circuits, this circuit-level mapping behaviour is efficient to operate, but hard to predict and difficult to clone [2].

Due to these circuit-derived, instance-specific, and unclonable properties, PUFs essentially serve as hardware fingerprints for IoT devices. There is no need for PUFs-embedded IoT devices to store secret keys in non-volatile memory, thus, effectively resisting physical and cloning attacks [3]. Consequently, PUFs are widely regarded as hardware primitives to activate various IoT security mechanisms [1], [4], [5], [6].

Authentication and data encryption are two main PUFs-based security mechanisms for IoT. As illustrated in Fig. 1, CRPs will be pre-registered in the verifier through a secure channel. Subsequently, leveraging PUFs as physical identities, IoT devices (e.g., vehicles, UAVs, smart meters) can establish authentication and secure communications with verifiers (e.g., road units, base stations, edge servers) in an efficient and secure way. Moreover, PUFs-based key generation could be seamlessly integrated with symmetric encryption [7], [8] or data compression [9] algorithms to safeguard the device's sensitive information. Therefore, it is promising to integrate PUFs into different IoT scenarios to enhance system security.


Mechanism 1: PUFs-based Authentication
1. Challenge
PUFs-embedded IoT Device
2. Response
=?
3. Authenticate
Verifier
Pre-registered CRPs
Mechanism 2: PUFs-based Data Encryption
1. Challenge
PUFs-embedded IoT Device
2. Response
3. PUF-based Key Generation
4. Encrypt
Plaintext
5. Secure data storage
Ciphertext
Features of CRPs management:
Pre-registration, One-time pad, generation from hardware.

Challenge	Response
~~001111~~	~~001111~~ ✓
100010	010011
010000	110111
110110	110001
111001	101011

Fig. 1. PUFs-based security mechanisms and features of CRPs management.

However, managing CRPs has features of pre-registration,

Yan Zhang (Corresponding Author) is with the School of Computer Engineering, Jiangsu University of Technology, Changzhou, Jiangsu, China. E-mail: yan.zhang@jsut.edu.cn.

Bing Li is with the School of Cyber Science and Engineering, Southeast University, Nanjing, Jiangsu, China. E-mail: bernie_seu@seu.edu.cn.

Bo Liu is with the School of Computer Science, University of Technology Sydney, Ultimo, NSW, AU. E-mail: Bo.Liu@uts.edu.au.

Jinke Chang is with the UCL Faculty of Medical Sciences, University College London, London, UK. E-mail: jinke.chang@ucl.ac.uk.

This work was supported by National Natural Science Foundation of China 62174150.

one-time pad, generation from hardware, multidimensional security threats will be brought to CRPs

(1) Impersonation attack: The security of CRPs storage remains an open problem for PUFs-based authentication [10], [11]. Exposed CRPs can be directly exploited to impersonate IoT devices. What is worse, when implementing PUFs-based authentication in distributed environments with multiple verifiers, the attack surface will inevitably expand [10]. As CRPs storage in verifiers may not always be secure, preventing impersonation attacks becomes challenging.

(2) Availability attack: To avoid storing CRPs in multiple verifiers, current studies rely on trusted authorities [4], [5], [6], [10], [12], [13], [14], [15] to provide strong protection for CRPs storage. However, involving the trusted authority in the authentication process will lead to the availability threats, such as denial-of-service attacks.

(3) Machine learning attack: Apart from the leakage of CRPs storage, if attackers, either from public channels or curious verifiers, collect enough raw CRPs during the authentication process, it is possible to model the PUF circuit using machine learning-based methods [16].

(4) Single point failure: The secret keys for PUFs-based data encryption are derived from CRPs [8], [9]. If the embedded PUF circuit is broken, lost, or seriously aging, devices cannot recover the origin encryption keys as well as the protected information. Even though CRPs may be backed up by the trusted authority, the threat of single point failure still exists.

Therefore, we carry out our research to answer the following question:

Is it possible to simultaneously solve these multidimensional security threats existing in different PUFs-based security mechanisms?

The Shamir secret sharing (SSS) [17] technology is highly suitable for providing comprehensive CRPs protection in PUFs-based authentication and data protection. PUF responses can be split by (n-t)-SSS into n slices and distributed to n different slice providers (SPs), avoiding storing CRPs in verifiers. As long as the number of revealed slices remains below the threshold (t), the information-theoretic security of CRPs storage could be ensured. More importantly, this protection method relies on distributed SPs to share CRPs on behalf of the trusted authority. The impersonation and availability attacks could be resisted at the same time.

During the authentication, PUF responses will be reconstructed by verifiers using SSS system. However, exposing even a limited number of CRPs still leads to machine learning-based attacks [16]. To mitigate this threat, we propose leveraging Nonce N to randomize PUF response $R = (R_1||R_2)$ into the PUF secret $(N \cdot R_1 + R_2)$. In each request, SPs will perform homomorphic additions on SSS slices of R_1, R_2 to compute PUF secret slices, which are then shared with verifiers. The verifier will reconstruct PUF secret slices to obtain the randomized PUF response, allowing authentication of the IoT device without revealing real CRPs. This approach ensures that CRPs are available but invisible.

Moreover, by collecting no less than t response slices, the PUFs-based encryption keys will be successfully reconstructed. The single point failure could be resisted.

However, introducing SSS system to support authentication and data encryption raises trust concerns related to CRPs synchronization, SPs supervision, and user deployment. First, slices are pre-registered in SPs, and each slice will be used only once. The slice usage order should be synchronized among SPs [11] to resist the desynchronization attack. Second, slice providers may not be accountable for their behaviours. It is necessary to supervise the whole sharing process. Third, slice users should locate devices' SPs and build secure channels with SPs to retrieve PUF secret slices. Additional registrations are required to adapt to the change of SSS system, such as SSS parameters and SPs' certificates. It makes user deployment complicated and inconvenient.

To address these trust issues, we turn to blockchain technology [18]. As the blockchain has features of decentralization, tamper-proofing, and distributed consistency, trust can be built between slice users and SSS system, as well as within SSS system itself. The blockchain consensus mechanism can be used to synchronize the latest slice usage order among SPs and trace the change of SSS system for slice users. In addition, if the PUF secret slices are forwarded to slice users in the form of transactions, the entire reconstruction process will be tamper-proofing, non-repudiable and accountable.

Therefore, our work contributes to combining SSS with blockchain to first simultaneously solve multidimensional security threats to CRPs existing in PUFs-based authentication and data protection. Not only SSS is leveraged to provide comprehensive CRPs protection, but also the blockchain is integrated to address trust issues of CRPs synchronization, SPs supervision, and user deployment brought by the SSS system. The major contributions are summarized as follows:

(1) We propose a security-enhanced and reliable CRPs management method. SSS and its homomorphic addition feature are leveraged to provide strong protection for CRPs storage, sharing, and backup. Meanwhile, the blockchain is integrated to develop a reliable sharing mode, which synchronizes CRPs usage order and supervises SPs' sharing behaviours.

(2) We design a PUF-as-a-Service (PaaS) framework with flexible user deployment. The blockchain is leveraged to trace the change of SSS system and integrate different PUFs-based mechanisms. Any one of the users deployed in PaaS can always build secure on-chain channels with slice providers and employ the PUF service through transactions, eliminating the need for additional registrations.

(3) Based on our proposed CRPs management method and PaaS framework, we successfully construct PUFs-based distributed authentication and recoverable data sharing with multidimensional CRPs security protection.

(4) A comprehensive security analysis of authentication and data sharing is given. Furthermore, a proof-of-concept prototype was implemented to conduct experimental evaluation and comparative analysis. In-depth and extensive discussions are included to show the promising prospect of our scheme.

II. Related Work

In this part, we review relevant PUFs-based authentication and data encryption studies.

A. PUFs-based Authentication for IoT

PUFs-based authentication has become an important security mechanism for IoT scenarios. However, CRPs storage in verifiers will not always be secure, especially when expanding authentication to distributed environments.

To avoid storing CRPs explicitly in multiple verifiers, the existing studies rely on the trusted authority to protect CRPs storage. Chatterjee et al. [10] impressively proposed an authentication protocol by combining identity-based encryption with hash functions to build a secure mapping mechanism. The security association provider provides mapped CRPs to support multi-verifier authentication. In work [14], CRPs are also protected by identity-based encryption and stored in a cloud server to help different verifiers authenticate smart meters. The complex cryptographic operations are delegated to the verifier to balance security and efficiency. Aman et al. utilized PUFs to build privacy-preserving and scalable authentication between roadside units and vehicles [4]. The trusted authority was involved to specially protect vehicles' CRPs and significantly reduce the communication overhead. For Internet of Drones [5] and smart grid [6], Gope et al. constructed a service provider to update devices' CRPs during each authentication request. Moreover, a dynamic identity management mechanism was also designed for devices to achieve anonymous authentication. In work [15], an anonymous authentication scheme 3PAA was proposed to grant or revoke the user access dynamically. The group member leveraged Pedersen Commitment to securely insert CRPs into access logs. When applying 3PAA to distributed environments, multi-registration should be performed to distribute access logs to multiple application providers. In general, relying on the trusted authority to protect CRPs storage in distributed environments still suffers from availability attacks. The security of CRPs storage and sharing should all be achieved.

To resist machine learning attacks, most of the studies would not expose real CRPs during the authentication process. Qureshi et al. [12] designed a lightweight masking function to obfuscate CRPs, eliminating the need for hash functions. Besides, their approach prevented devices from unauthorized access, which completely inhibits any model-building and side-channel analysis attack. The research [16] proposed lightweight Strong-PUFs-based authentication by transmitting shuffled PUF responses. The real response will be reconstructed by the verifier using SSS to finish the authentication.

In general, when expanding PUFs-based authentication to multiple verifiers, the existing studies have not simultaneously solved impersonation and availability attacks. Meanwhile, the resistance to the machine learning attack should also be considered.

B. PUFs-based Data Protection

PUFs-based data encryption has been utilized to protect sensitive information. Barbareschi et al. [7] introduced the Pseudo-PUF architecture, integrating PUFs key derivation with symmetric encryption modules. This approach not only adapts to advanced security primitives but is also suitable for the cost and resource demands of IoT devices. The work [9] developed a joint compression and encryption scheme to secure data storage in edge servers. The storage overhead is reduced by the Huffman compression algorithm. PUF responses are employed to mutate the Huffman tree for enhanced data protection. Dai et al. [8] built two symmetric searchable encryption protocols to protect outsourced data. The threat of memory leakage will be mitigated through PUFs-based key storage. The work [19] also realized the PUFs-based key generation and management in an FPGA-based blockchain system. Transactions will be signed by PUFs key and generated in an isolated and secure manner. However, the issue of recoverable CRPs storage has not been seriously considered. Existing research lacks proper CRPs backup mechanisms, with CRPs either not backed up or simply stored in a trusted authority. Once single point failure happens, the encrypted data will be lost.

In conclusion, there is an urgent need to simultaneously solve multidimensional security threats present in PUFs-based authentication and data protection.

III. System Model and Preliminaries

A. Shamir Secret Sharing

To provide comprehensive protection for CRPs, SSS is leveraged to divide the PUF response into SSS slices. In addition, homomorphic additions on response slices facilitate the sharing of the PUF secret. The definitions and properties of (n-t)-SSS are discussed as follows.

(1) Secret Split: The dealer selects n receivers and sets the threshold as t. A (t-1)-degree polynomial is initialized as $f(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \cdots + a_1 x + s$, $(s, a_1, a_2, \cdots, a_{t-1} \in \mathbb{F}_p)$, where s is the secret, p is a chosen large prime. Then, the secret s will be split into n slices $\{(ID_i, Slice_i = f(ID_\mathrm{i}))\}_{i=1}^{n}$. The secret split algorithm is denoted as $\{Slice_i\}_{i=1}^{n} = SS(n, t, p, s)$.

(2) Secret Reconstruct: After aggregating at least $(l \geq t)$ slices, s can be reconstructed by the Lagrange interpolation formula as below:

$$s = f(0) = \sum_{i=1}^{l} Slice_i \prod_{j=1, j \neq i}^{l} \frac{-ID_j}{ID_i - ID_j}. \tag{1}$$

The secret reconstruction algorithm is defined as $s = SSS\_RC(p, \{Slice_i\}_{i=1}^{l})$.

(3) Additional homomorphism: This property implies that the addition of slices will result in the addition of the shared secrets and the threshold t remains unchanged. In other words, the secret $(s_1 + s_2)$ will be reconstructed by $\{Slice_{1,i} + Slice_{2,i}\}_{i=1}^{t}$.

B. Zero-knowledge Proof of Knowledge

In this paper, Zero-knowledge Proof of Knowledge (ZKPoK) [20] is used by slice providers (SPs) to authenticate slice users and build on-chain secure channels with Diffie-Hellman key exchange. In general, the notation of ZKPoK is expressed as $ZKP\ \{(a) : A = a \cdot P\}$ [21]. The prover is allowed to prove the knowledge of a by generating a cryptographic proof based on statement $A = a \cdot P$. If the proof is verified, the knowledge of a could be proved, without revealing its actual value.

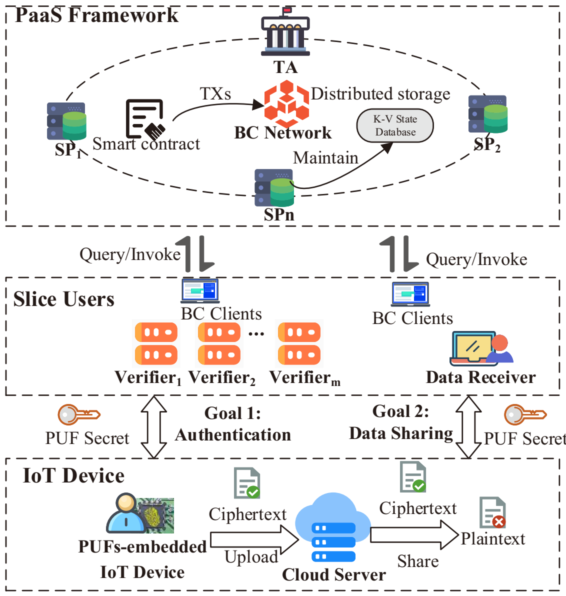


Fig. 2. System model.

C. System Model

Our system aims to provide authentication and data sharing for smart grids [6], intelligent transportation systems [4], Internet of Drones [5], Internet of Videos [10], etc. We abstract these practical scenarios into our system model, as shown in Fig. 2. The proposed three-layer architecture consists of PaaS framework, slice users, and IoT devices.

(1) PaaS Framework: This framework is composed of the trusted authority, slice providers, and the blockchain.

• Trusted Authority (TA): The TA initializes the system, sets up the blockchain, and completes the registration for SPs, slice users, and IoT devices. Besides, TA would not be involved in the authentication and data sharing processes.

• Slice Providers (SPs): SPs act as the participators of (t-n)-SSS and the blockchain consensus. They also make computations on response slices and transmit encrypted PUF secret slices through the blockchain network. In addition, appropriate blockchain clients will be chosen to invoke/query the smart contract or monitor transactions on behalf of SPs.

• Blockchain: To adapt to IoT environments, the consortium blockchain will be chosen to efficiently send and validate transactions. The transactions are validated and recorded by SPs to maintain a key-value state database.

(2) Slice Users: Slice users could be edge servers, base stations, gateways, etc. They will be deployed in PaaS after registering their public keys in the blockchain. Registered users can employ the PUF service to authenticate IoT devices or receive shared IoT data. Blockchain clients are also installed on them to interact with the blockchain network.

(3) IoT Devices: IoT devices are defined as smart devices and machines with constrained resources, including vehicles, UAVs, smart meters, etc. These devices usually need to be authenticated by multiple verifiers. Meanwhile, they may store sensitive data or share them through public channels. Note that each IoT device is equipped with a secure PUF circuit to provide physically secure hardware fingerprints.

D. Threat Model

The capabilities of the adversary in our system model are modeled as follows:

(1) Channel: The communication channel is constructed by the widely-accepted Dolve-Yao (DY) threat model. The adversary is assumed to fully control the channel.

(2) IoT Devices: Devices are under the threat of physical and cloning attacks. The adversary has the ability to derive secret keys stored in devices' local non-volatile memory [3].

(3) Slice Users: Slice users will protect their private keys properly to ensure the security of their blockchain accounts. However, they are modeled to be semi-honest, attempting to obtain real CRPs to model the PUF circuit.

(4) Slice Providers: SPs are also considered semi-honest, ensuring the validity of the PUF secret slices they provide. However, they will try their best to retrieve privacy information. Additionally, SPs will securely keep their private keys.

(5) Threats to (t-n)-SSS: The adversary may corrupt the storage of less than t SPs to steal or damage PUF response slices. The number of collusive SPs will also be less than t. In addition, there always exist at least t semi-honest SPs participating in the reconstruction phase.

(6) Blockchain: The entire consortium blockchain is assumed to be secure, trustworthy, tamper-proof, and reliable. Only transactions proposed by authorized participants and registered users will be validated and recorded.

IV. CRPs Management and PaaS Framework

In this section, we describe our proposed CRPs management method and PaaS framework in Fig. 3. The CRPs management method comprises slice-based CRPs storage and blockchain-based reliable sharing mode.

A. Slices-based CRPs storage

We take one CRP as an example to illustrate the storage method. The SSS system is mainly used to ensure the security of CRPs storage and backup.

(1) Divide response: Each $|R|$ bit-length PUF response is divided into two $|R|/2$ bit-length slices $\{Slice_1, Slice_2\}$. Then, the secret split algorithm $SS(\cdot)$ of (t, n)-SSS is leveraged to respectively split $\{Slice_1, Slice_2\}$ into a sub-slices set $\{Slice_{1,i}, Slice_{2,i}\}_{i=1}^{n}$.

(2) Distribute and store sub-slices: The ith sub-slices, along with their related challenge C_i and helper data HD_i are distributed to SP_i in secure channels. Each SP_i retains these elements in its local storage.

(3) Register IoT devices: To register IoT devices, the TA invokes the smart contract, recording the blockchain addresses of all SPs and the initial (C_1, HD_1).

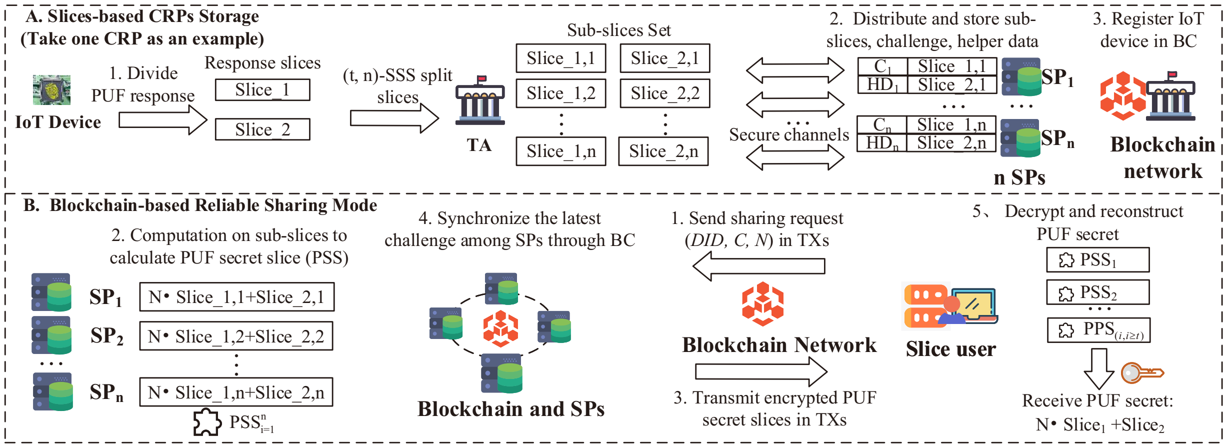

A. Slices-based CRPs Storage
(Take one CRP as an example)
IoT Device
1. Divide PUF response
Response slices
Slice_1
Slice_2
(t, n)-SSS split slices
TA
Sub-slices Set
Slice_1,1
Slice_1,2
Slice_1,n
Slice_2,1
Slice_2,2
Slice_2,n
Secure channels
2. Distribute and store sub-slices, challenge, helper data
C1 Slice_1,1
HD1 Slice_2,1
Cn Slice_1,n
HDn Slice_2,n
SP1
SPn
n SPs
3. Register IoT device in BC
Blockchain network
B. Blockchain-based Reliable Sharing Mode
2. Computation on sub-slices to calculate PUF secret slice (PSS)
SP1 N• Slice_1,1+Slice_2,1
SP2 N• Slice_1,2+Slice_2,2
SPn N• Slice_1,n+Slice_2,n
PSS n i=1
4. Synchronize the latest challenge among SPs through BC
Blockchain and SPs
1. Send sharing request (DID, C, N) in TXs
Blockchain Network
3. Transmit encrypted PUF secret slices in TXs
Slice user
5、Decrypt and reconstruct PUF secret
PSS1
PSS2
PPS(i,i≥t)
Receive PUF secret:
N• Slice1 +Slice2

Fig. 3. CRPs management method with multidimensional security protection.

B. Blockchain-based Reliable Sharing Mode

In this mode, SPs share the PUF secret $(N \cdot Slice_1 + Slice_2)$ slices to users through on-chain channels. The blockchain is utilized to synchronize the slice usage order among SPs. In addition, PUF secret slices are transmitted in the form of transactions to supervise SP's behaviours. The reliable slice sharing proceeds as follows:

(1) Send sharing request: The slice user initiates a sharing request by querying the latest challenge *C* of device *DID* from the blockchain. Then, the nonce *N* will be chosen and transmitted to SPs along with *(DID,C)* through transactions.

(2) Computation on sub-slices to calculate PSSs: Upon receiving *(DID, C, N)*, each SP_i will use its stored sub-slices $\{Slice_{1,i}, Slice_{2,i}\}$ to homomorphically compute PUF secret slice, which is denoted as $PSS_i = N \cdot Slice_{1,i} + Slice_{2,i}$.

(3) Transmit Encrypted PSSs: SPs utilize slice user's public key to encrypt the $\{PSS_i\}_{i=1}^{n}$. The ciphertext will be transmitted to users in the form of transactions to supervise SPs' sharing behaviours.

(4) Synchronize the latest PUF challenge: The latest (C_{x+1}, HD_{x+1}) will be synchronized among SPs through the blockchain consensus mechanism.

(5) Decrypt and reconstruct: The slice user utilizes its private key to decrypt and obtain $\{PSS_i\}_{i=1}^{n}$. Then, the reconstruction algorithm $SSS\_RC(p, \{PSS_i\}_{i=1}^{n})$ is executed to reconstruct the complete PUF secret.

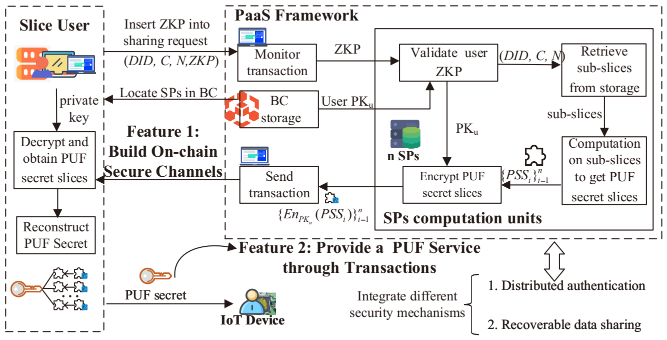

Slice User
Insert ZKP into sharing request
(DID, C, N,ZKP)
PaaS Framework
Monitor transaction
ZKP
Validate user ZKP
(DID, C, N)
Retrieve sub-slices from storage
Locate SPs in BC
private key
BC storage
User PKu
sub-slices
PKu
n SPs
Feature 1: Build On-chain Secure Channels
Decrypt and obtain PUF secret slices
Send transaction
Encrypt PUF secret slices
{PSS}n i=1
Computation on sub-slices to get PUF secret slices
{EnPKu(PSSi)}n i=1
SPs computation units
Reconstruct PUF Secret
Feature 2: Provide a PUF Service through Transactions
PUF secret
IoT Device
Integrate different security mechanisms
1. Distributed authentication
2. Recoverable data sharing

Fig. 4. Proposed PaaS framework.

C. PaaS Framework

As is shown in Fig. 4, the PaaS framework is constructed to achieve flexible user deployment in SSS system and provide different security mechanisms for distributed users. The important features are summarized as follows:

(1) Build on-chain secure channels: The blockchain is leveraged to trace the change of SSS system. The slice user will locate SPs by querying the blockchain and insert his ZKP in sharing request. Then, SPs will query the user on-chain public key to validate user's ZKP. In addition, the user public key would be utilized by SPs to encrypt PUF secret slices and transmit them in the form of transactions. Once deployed in the blockchain, users can always build secure on-chain channels with SPs to share the PUF secret.

(2) Integrate different PUFs-based security mechanisms: The PUF secret can be leveraged by slice users to activate both authentication and data sharing. It means that any one of the slice users can always send transactions to employ the PUF service, needless of performing additional registrations to accommodate the change of SSS parameters and SPs' certificates.

V. Distributed Authentication and Recoverable Data Sharing

In this section, we construct PUFs-based distributed authentication and recoverable data sharing based on our proposed PaaS framework and CRPs management method.

A. Initialization

Our scheme is initialized as follows:

TA first selects a cyclic group $\mathbb{G}$ with prime order q on elliptic curve $E_p(a,b)$, where $a, b \in$ finite field $\mathbb{F}_p$. Then, a generator $P \in \mathbb{G}$ and three hash functions $H_1 : \{0,1\}^* \rightarrow \{0,1\}^{l_1}, H_2 : \{0,1\}^* \rightarrow \{0,1\}^{l_2}, H_3 : \{0,1\}^* \rightarrow \mathbb{Z}_q$ are chosen. To set up SSS, TA chooses, n, t, and another $\mathbb{F}_{p'}$ with prime $p' > |R|$. $|R|$ denotes the bit-length of the shared PUF response. Last, public parameters $\{n, t, q, p, p', P, \mathbb{G}, H_{1\sim3}\}$ are made public.

B. Deployment of Blockchain and Smart Contract

The TA sets up the blockchain network and deploys the smart contract. The blockchain clients will be installed to query/invoke the smart contract, as well as monitor transactions. Recorded transactions will be secondly used to maintain a key-value state database, which supports the efficient query by blockchain addresses. Five different functions of the smart contract are provided:

(1) *DevReg* $(H_1(\mathit{DID}), \{H_1(\mathit{IDSP}_i)\}_{i=1}^n, (C_1, \mathit{HD}_1))$. TA invokes this function to register IoT devices. The blockchain addresses of all SPs and the initial (C_1, HD_1) used for authentication will be recorded.

(2) *UsrReg* $(H_1(\mathit{VID}), pkv)$. Slice users invoke this function *UsrReg* $(H_1(\mathit{VID}), pkv)$ to create their blockchain account. The public key pkv of the user will be written in the blockchain.

(3) *ZKPoK* $(H_1(\mathit{VID}), N, H_1(\mathit{DID}), C_x, \pi_{i=1}^n)$. Slice user invoke this function to distribute ZKPoK cryptographic proofs $\pi_{i=1}^n$ and challenge to SPs in the form of transactions.

(4) *TXEPSS* $(H_1(\mathit{VID}), N, \{\mathit{EPSS}_i, \delta_i \cdot P\}_{i=1}^n)$. SPs transmit the $\{\mathit{EPSS}_i, \delta_i \cdot P\}_{i=1}^n$ encrypted PUF secret slices and Nonce N to slice users through the blockchain.

(5) *Update* $(H_1(\mathit{DID}), (C_{x+1}, \mathit{HD}_{x+1}))$. SPs invoke this function to synchronize the latest$(C_{x+1}, \mathit{HD}_{x+1})$ among SPs for the subsequent authentication request.

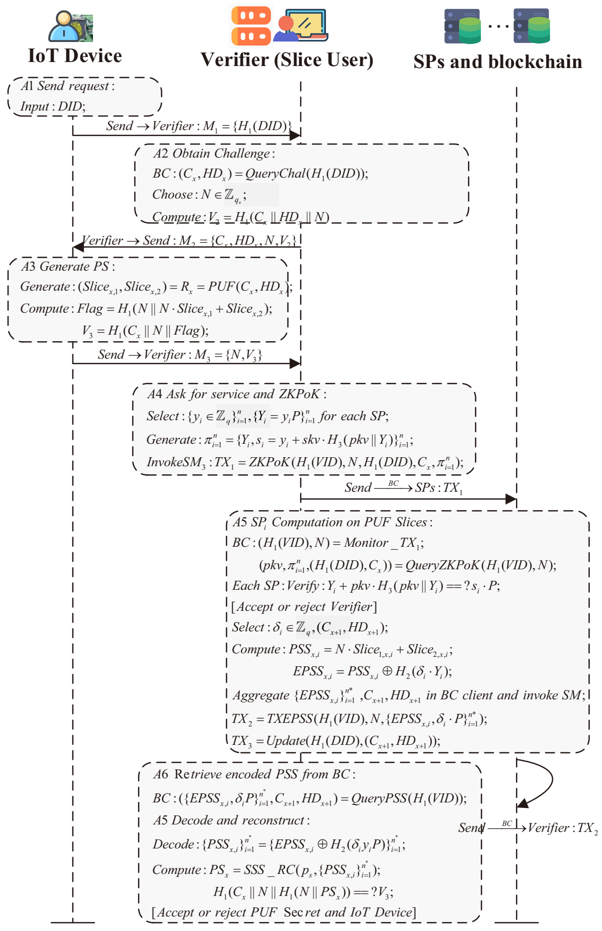

IoT Device
Verifier (Slice User)
SPs and blockchain
A1 Send request :
Input : DID;
Send → Verifier : $M_1 = \{H_1(DID)\}$
A2 Obtain Challenge :
BC : $(C_x, HD_x) = QueryChal(H_1(DID))$;
Choose : $N \in \mathbb{Z}_{q_s}$;
Compute : $V_2 = H_1(C_x \| HD_x \| N)$
Verifier → Send : $M_2 = \{C_x, HD_x, N, V_2\}$
A3 Generate PS :
Generate : $(Slice_{x,1}, Slice_{x,2}) = R_x = PUF(C_x, HD_x)$;
Compute : $Flag = H_1(N \| N \cdot Slice_{x,1} + Slice_{x,2})$;
$V_3 = H_1(C_x \| N \| Flag)$;
Send → Verifier : $M_3 = \{N, V_3\}$
A4 Ask for service and ZKPoK :
Select : $\{y_i \in \mathbb{Z}_q\}_{i=1}^n, \{Y_i = y_i P\}_{i=1}^n$ for each SP;
Generate : $\pi_{i=1}^n = \{Y_i, s_i = y_i + skv \cdot H_3(pkv \| Y_i)\}_{i=1}^n$;
InvokeSM$_3$: $TX_1 = ZKPoK(H_1(VID), N, H_1(DID), C_x, \pi_{i=1}^n)$;
Send $\xrightarrow{BC}$ SPs : TX_1
A5 SP_i Computation on PUF Slices :
BC : $(H_1(VID), N) = Monitor\_TX_1$;
$(pkv, \pi_{i=1}^n, (H_1(DID), C_x)) = QueryZKPoK(H_1(VID), N)$;
Each SP : Verify : $Y_i + pkv \cdot H_3(pkv \| Y_i) == ?\, s_i \cdot P$;
[Accept or reject Verifier]
Select : $\delta_i \in \mathbb{Z}_q, (C_{x+1}, HD_{x+1})$;
Compute : $PSS_{x,i} = N \cdot Slice_{1,x,i} + Slice_{2,x,i}$;
$EPSS_{x,i} = PSS_{x,i} \oplus H_2(\delta_i \cdot Y_i)$;
Aggregate $\{EPSS_{x,i}\}_{i=1}^{n^*}, C_{x+1}, HD_{x+1}$ in BC client and invoke SM;
$TX_2 = TXEPSS(H_1(VID), N, \{EPSS_{x,i}, \delta_i \cdot P\}_{i=1}^{n^*})$;
$TX_3 = Update(H_1(DID), (C_{x+1}, HD_{x+1}))$;
A6 Retrieve encoded PSS from BC :
BC : $(\{EPSS_{x,i}, \delta_i P\}_{i=1}^{n^*}, C_{x+1}, HD_{x+1}) = QueryPSS(H_1(VID))$;
A5 Decode and reconstruct :
Decode : $\{PSS_{x,i}\}_{i=1}^{n^*} = \{EPSS_{x,i} \oplus H_2(\delta_i y_i P)\}_{i=1}^{n^*}$;
Compute : $PS_x = SSS\_RC(p_s, \{PSS_{x,i}\}_{i=1}^{n^*})$;
$H_1(C_x \| N \| H_1(N \| PS_x)) == ?\, V_3$;
[Accept or reject PUF Secret and IoT Device]
Send $\xrightarrow{BC}$ Verifier : TX_2

Fig. 5. PUFs-based distributed authentication.

C. Registration

In this phase, IoT devices and slice users will be registered. According to the slice-based storage method in section IV, we first explain the device registration phase.

Step R.1: The IoT device *DID* first prepares and divides k CRPs $\{C_j, R_j, \mathit{HD}_j\}_{j=1}^k$.

Step R.2: The PUF response slices are split into $\{C_j, \mathit{HD}_j, R_j = \{\mathit{Slice}_{1,j,i}, \mathit{Slice}_{2,j,i}\}_{i=1}^n\}_{j=1}^k$ sub-slice set. The IoT device distributes the sub-slice set to TA.

Step R.3: TA stores these sub-slices on n SPs and then invokes *DevReg* function to build device's blockchain account and initialize (C_1, HD_1). Note that TA will not keep sub-slices to avoid CRPs leakage.

Moreover, the sub-slices set will be managed according to the following rules.

Rule 1: The first k_0 CRPs activate authentication service. The device will be authenticated by $(N \cdot \mathit{Slice}_1 + \mathit{Slice}_2)$, which is together computed by SPs. Once providing PUF secret slices, SPs will remove the used challenge, helper data, and sub-slices from their local storage. Meanwhile, *Update* function will be invoked to synchronize the new $(C_{x+1}, \mathit{HD}_{x+1})$.

Rule 2: The remaining $(k - k_0)$ CRPs enable data sharing by using $(N \cdot \mathit{Slice}_1 + \mathit{Slice}_2)$ to generate PUFs key. To make PUFs key recoverable, SPs will store these sub-slices, challenges, helper data, and Nonce N in their local storage.

Rule 3: The TA periodically registers slices to avoid the storage overhead on SPs as well as adapt to the change of SSS parameters.

To register distributed slice users, the key pair $\{skv \in \mathbb{Z}_q, pkv = skv \cdot P \in \mathbb{G}\}$ and $\mathit{VID} = H_1(pkv)$ should be first generated. Then, the user blockchain address $H_1(\mathit{VID})$ will be calculated to invoke *UsrReg* function and create the blockchain account for the slice user. If $H_1(\mathit{VID})$ already exists, the slice user has to re-generate the key pair.

D. Unidirectional Authentication

In this phase, we construct distributed unidirectional authentication. Any one of the registered users could employ the PUF service to authenticate the IoT device. The authentication phase is discussed as follows and depicted in Fig. 5.

Step A.1: The IoT device *DID* sends request M_1 to verifier.

Sender $\rightarrow$ Verifier: $M_1 = \{H_1(\mathit{DID})\}$.

Step A.2: Once receiving M_1, the verifier queries the blockchain address $H_1(\mathit{DID})$ to get the device's PUF challenge and helper data (C_x, HD_x). More importantly, Nonce N will be chosen by the verifier and inserted into the returned message. The random number N not only helps to resist replay attack, but also randomizes each PUF response into the PUF secret, instead of exposing real CRPs. After computing the hash value $V_2 = H_1(C_x || \mathit{HD}_x || N)$, the verifier constructs and returns M_2 to the IoT device.

Verifier $\rightarrow$ Sender: $M_2 = \{C_x, \mathit{HD}_x, N, V_2\}$.

Step A.3: Once receiving (C_x, HD_x), the PUF instance is challenged to output R_x. Then, R_x will be further divided into $(\mathit{Slice}_{x,1}, \mathit{Slice}_{x,2})$. Afterward, the hash function is used to calculate $Flag = H_1(N||N \cdot Slice_{x,1} + Slice_{x,2})$ and $V_3 = H_1(C_x||N||Flag)$. Last, the message M_3 is sent to the verifier.

Sender $\rightarrow$ Verifier: $M_3 = \{V_3\}$.

Step A.4: Upon receiving M_3, the verifier randomly selects n elements $\{y_i \in \mathbb{Z}_q\}_{i=1}^n$ and computes $\{Y_i = y_i \cdot P\}_{i=1}^n$. The Fiat-Shamir heuristic is applied to generate n proofs $\pi_{i=1}^n = \{Y_i, s_i = y_i + skv \cdot H_3(pkv||Y_i)\}_{i=1}^n$ for ZKP $\{(skv) : pkv = skv \cdot P\}$, where (skv, pkv) is the verifier's key pair. Last, the verifier will invoke the function $ZKPoK$ $\{H_1(VID), N, H_1(DID), C_x, \pi_{i=1}^n\}$ to distribute cryptographic proofs $\pi_{i=1}^n$ to n SPs in the form of transactions.

Verifier $\xrightarrow{BC}$ SPs:
$TX_1 = ZKPoK(H_1(VID), N, H_1(DID), C_x, \pi_{i=1}^n)$.

Step A.5: Once the transaction TX_1 is recorded, the blockchain client of SPs will monitor the on-chain event to obtain $H_1(VID)$ and N. The public keys, proofs, and challenge $(pkv, \pi_{i=1}^n, H_1(DID), C_x) = Query\_Ledger(H_1(VID), N)$ will be further queried from the blockchain. Then, each SP_i verifies the authenticity of the verifier by checking whether $Y_i + H_3(pkv||Y_i) \cdot pkv = s_i \cdot P$. If so, SP_i accepts the verifier and continues to compute the PUF secret slice (PSS).

SPs will find PUF sub-slices $(Slice_{1,x,i}, Slice_{2,x,i})$ according to $H_1(DID)$ and C_x. Then, each PUF secret slice is calculated as $PSS_{x,i} = (N \cdot Slice_{1,x,i} + Slice_{2,x,i})$. To encrypt each slice, SP_i chooses a random $\delta_i \in \mathbb{Z}_q$ to compute $\delta_i \cdot Y_i$. The $PSS_{x,i}$ will be encrypted into $EPSS_{x,i} = PSS_{x,i} \oplus H_2(\delta_i \cdot Y_i)$. After that, (C_{x+1}, HD_{x+1}) is selected in the sequence of the index. Each SP_i will transmit $\{EPSS_{x,i}, (\delta_i \cdot P)\}$, (C_{x+1}, HD_{x+1}) to the blockchain client. The $TXEPSS$ function will be invoked to insert n^* encrypted PSSs into transaction TX_2. At last, the $Update$ function will also be utilized to synchronize new (C_{x+1}, HD_{x+1}) among SPs. Note that each SP will remove the used sub-slice from the local storage.

$\{SP_i\}_{i=1}^{n*} \xrightarrow{BC}$ Verifier:
$TX_2 = TXEPSS(H_1(VID), N, \{EPSS_{x,i}, \delta_i P\}_{i=1}^{n^*})$
$TX_3 = Update(H_1(VID), (C_{x+1}, HD_{x+1}))$

Step A.6: When noticing TX_2, the verifier queries $\{EPSS_{x,i}, \delta_i \cdot P\}_{i=1}^{n^*} = Query\_Ledger(H_1(VID))$ from the blockchain. Then, $(y_i \cdot \delta_i \cdot P)$ is computed to decrypt and get $\{PSS_{x,i}\}_{i=1}^{n^*} = \{EPSS_{x,i} \oplus H_2(y_i \cdot \delta_i \cdot P)\}_{i=1}^{n^*}$. That is to say, the PUF secret could be reconstructed by $PS_x = SSS\_RC(p, \{PSS_{x,i}\}_{i=1}^{n^*})$. Finally, the verifier checks whether $H_1(C_x||N||H_1(N||PS_x)) = V_3$. If so, the IoT device DID could be authenticated.

E. Mutual Authentication

If mutual authentication (MA) is required, simple changes can be directly added to the unidirectional authentication process. The way of authenticating the IoT device is the same. We only discuss the main differences.

In step A.1, the additional element $N_2 = n_2 \cdot P \in \mathbb{G}$ will be generated and inserted into M_1. In step A.2, $X = N_2 \cdot skv$ will be calculated by the verifier to build $V_2 = H_1(C_x||HD_x||H_1(VID||X))$. Then, $M_2 = \{C_x, HD_x, H_1(VID), V_2\}$ will be sent back to the IoT device. Once receiving M_2, the IoT device uses its blockchain client to get the public key of the verifier pkv. The queried pkv could be used to check whether V_2 is valid. If so, the verifier VID is authenticated.

Note that our mutual authentication is not suitable for very lightweight IoT nodes, as both the operations of ECC cryptography and blockchain query will be involved to authenticate multiple verifiers.

Remark: Our work only ensures identity anonymity for devices and verifiers by transmitting their blockchain addresses in the public channel. Different messages sent by the same requestor could be linked. If unlinkability is required in a particular scenario, there exist two ways to improve our work. The first one is to store one-time pseudo identities of IoT devices in SPs. Each pseudo identity will be used for one request, as work [6] did. Another one stores certificates of verifiers in the IoT device in advance [25]. The transmitted identity will be encrypted by the shared element computed by public key cryptography. However, anonymity and unlinkability are out of the scope of this paper. We omit detailed discussions.

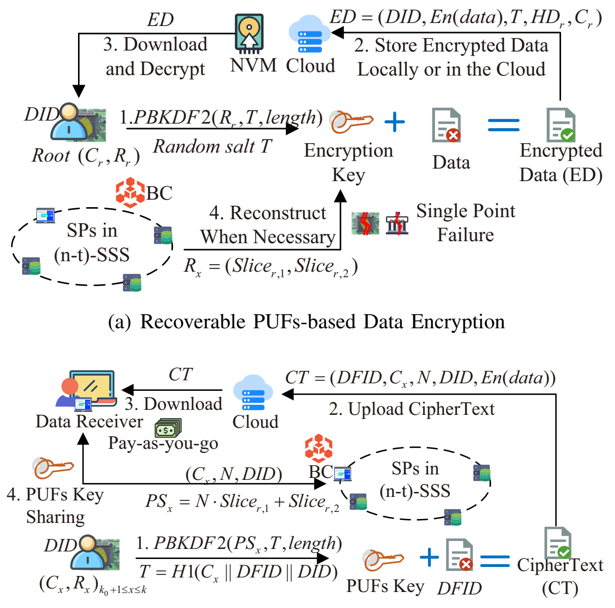

ED
3. Download and Decrypt
NVM
Cloud
ED = (DID, En(data), T, HD_r, C_r)
2. Store Encrypted Data Locally or in the Cloud
DID
Root (C_r, R_r)
1.PBKDF2(R_r, T, length)
Random salt T
Encryption Key
Data
Encrypted Data (ED)
BC
SPs in (n-t)-SSS
4. Reconstruct When Necessary
R_x = (Slice_r,1, Slice_r,2)
Single Point Failure
(a) Recoverable PUFs-based Data Encryption
CT
3. Download
Data Receiver
Cloud
CT = (DFID, C_x, N, DID, En(data))
2. Upload CipherText
Pay-as-you-go
(C_x, N, DID)
BC
SPs in (n-t)-SSS
4. PUFs Key Sharing
PS_x = N · Slice_r,1 + Slice_r,2
DID
(C_x, R_x)
1. PBKDF2(PS_x, T, length)
T = H1(C_x || DFID || DID)
PUFs Key
DFID
CipherText (CT)
(b) Data Sharing with dynamic PUFs key

Fig. 6. Data encryption and sharing.

F. Recoverable PUFs-based Data Encryption

As is shown in Fig. 6 (a), PUF-based data encryption is used by the IoT device to protect its sensitive information. The encryption key generated by the PUF circuit is recoverable.

(1) Generate encryption key: The IoT device will choose one CRP (C_r, R_r, HD_r) from $(k - k_0)$ CRPs and a salt-value T as the encryption root. Then, Password-Based Key Derivation Function 2 ($PBKDF2$) is used to generate the encryption key as Key=$PBKDF2$ $(PKCS\#5, R_r, T, key\ length)$.

(2) Store encrypted data (ED): The advanced symmetric encryption primitives, such as AES, will use the encryption key to protect sensitive data. The elements (C_r, HD_r, T) will

be kept with the encrypted data. The encrypted data will be stored locally or in the public cloud.

(3) Download and decrypt data: After downloading encrypted data, the IoT device can use the PUF response R_r to generate the decryption key with the salt value T .

(4) Reconstruct encryption key: If the PUF circuit and CRPs backup suffer from the single point failure, the TA has the authority to request sub-slices of R_r from SPs. The R_r can be reconstructed by SSS to recover the encryption key.

However, this data encryption method is not suitable for data sharing. First, it is unsafe to directly share the encryption root with other users. Second, as mentioned in [26], if public key cryptography is used to transmit the generated encryption key, the device should be always online and additional overhead will be brought.

G. Data Sharing with Dynamic PUFs Key

We propose a dynamic PUFs key generation method to support data sharing.

(1) Generate dynamic PUFs key: Assume each data file has an open identity *DFID*. IoT device *DID* will choose (C_x, HD_x) to calculate the unique encryption key for each data file. The salt value is generated by $T = H_1(C_x||DFID||DID)$. In addition, the PUF secret $PS_x = (N \cdot Slice_{x,1} + Slice_{x,2})$ is set as the seed of *PBKDF2*. The dynamic PUFs key is computed as *key=PBKDF2* $(PKCS\#5, PS_x, T, key\ length)$.

(2) Upload ciphertext (*CT*): The data file *DFID* will be encrypted into *En(data)* using the PUFs key. The *DFID*=$(DFID, C_x, DID, N, En(data))$ will be uploaded to the cloud for data sharing.

(3) Download ciphertext: The data receiver downloads the ciphertext from the cloud.

(4) PUFs key sharing (*PKS*): The data receiver uses (C_x, DID, N) to request slices from SPs. After taking the same steps $(A_4 \sim A_6)$ mentioned in the authentication phase, the data receiver will reconstruct the PUF secret (PS_x). Then, the PUFs key can be computed to decrypt the data *De(data)* and get the data file of *DFID*. After processing the initial key sharing, SPs will store the Nonce N with sub-slices, C_x, and helper data to verify future sharing requests.

VI. Security Analysis

In this section, we analyze our authentication and data sharing scheme by performing security proof and security discussions.

A. Provable Security and Security Reduction

The provable security is a classic and widely accepted way of performing security proof. The security reduction is an important security analysis method of provable security and has been used in many relevant studies [22], [23]. Hence, we use provable security and security reduction to make security proof for our authentication.

The provable security is based on complexity theory and reduces the security of protocols or algorithms to some good axioms or difficult problems. If the security reduction process is correct, and these axioms are correct or the mathematical problems (e.g., discrete logarithm problem, Diffie-Hellman problem) are indeed difficult to solve, the protocol/algorithm will be proved to be computationally secure. It is important to note that the provable security does not directly prove the security of the cryptographic scheme, but rather perform the security reduction to protocol/algorithm.

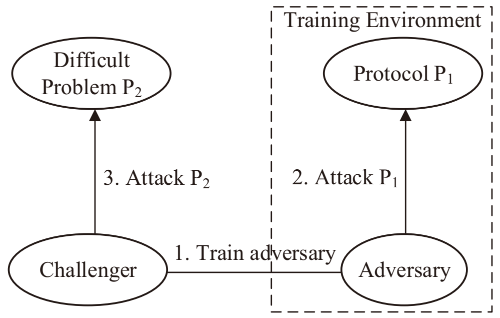


Fig. 7. Framework of security reduction.

As shown in Fig. 7, the challenger $\mathcal{C}$ first trains the adversary $\mathcal{A}$ to attack protocol P_1. $\mathcal{A}$'s behaviours in our training environment are modeled according to the threat model. If $\mathcal{A}$ is trained to break P_1, $\mathcal{C}$ will use $\mathcal{A}$'s attack ability to break difficult problem P_2. However, P_2 is assumed to be difficult to solve based on the computational complexity. Therefore, the security of P_1 can be proven by contradiction.

B. Security Model

The security model that models the adversary's behaviours in training is constructed as an interactive game between the challenge $\mathcal{C}$ and the adversary $\mathcal{A}$. $\mathcal{C}$ and $\mathcal{A}$ are both modeled by a Probabilistic Polynomial Time (P.P.T) Turing machine. The $\prod_{\Omega}^{l}$ is defined as an instance l of a participant Ω, where Ω represents IoT devices, verifiers, or SPs. In this game, $\mathcal{A}$'s behaviours are set as different queries and $\mathcal{C}$ answers them according to the real protocol as follows to perform the training process.

- $h_i(m_j)$*-Query*: When $\mathcal{A}$ asks a hash query on message m_j, $\mathcal{C}$ returns with a random element r_j and records (m_j, r_j) in the hash list L_{h_i}.
- *Extract* (DID_i)*-Query*: When $\mathcal{A}$ asks this query, $\mathcal{C}$ returns the identity DID_i and PUF instance PUF_i, and record them in the PUF instance list L_p.
- *Send* $(\prod_{\Omega}^{l}, m)$*-Query*: In this query, $\mathcal{A}$ sends m to participant Ω of instance l. $\mathcal{C}$ will make operations according to the actual protocol and return results to $\mathcal{A}$.
- *RevealPUFSecret* $(\prod_{\Omega}^{l})$*-Query*: If $\prod_{\Omega}^{l}$ is accepted, $\mathcal{C}$ returns the PUF secret to $\mathcal{A}$. Otherwise, $\perp$ will be returned.
- *ExtractPUFSlices* (DID_i, C_i)*-Query*: In this query, $\mathcal{C}$ returns less than t PUF slices correlated to C_i, where t is the threshold of (t-n)-SSS.
- *BC* (ID_i)*-Query*: In this query, $\mathcal{C}$ returns the information recorded in the blockchain account ID_i.

After a set of queries has been asked in finite time, the adversary $\mathcal{A}$ tries to break the security of our mutual authentication (MA). $\mathcal{A}$ will impersonate an IoT device to violate the verifier's authentication, or forge a verifier to pass the verification of the IoT device.

Next, we introduce the definitions of underlying hard problems and security assumptions used in security proof.

Definition 1 (Decisional Uniqueness Problem (DUP) for PUF [10], [15], [24]): Given a secure PUF instance, with only an n-bit output, a challenge C_i, and an n-bit string $z \in \{0,1\}^n$, this *DUP* problem is to decide whether z is a random string or a valid PUF response $R_i = \mathit{PUF}(C_i)$ correlated to C_i.

Definition 2 (Computational Diffie-Hellman (CDH) Problem: Given elements $(P, a \cdot P, b \cdot P \in \mathbb{G})$, this *CDH* problem aims to compute $a \cdot b \cdot P \in \mathbb{G}$.

Definition 3 (CDH and DUP Assumptions): Assume that *DUP* and *CDH* problems are hard. The P.P.T adversary $\mathcal{A}$ can solve these problems with negligible probability.

C. Security Proof

Lemma 1: No P.P.T adversary $\mathcal{A}$ can forge an IoT device DID_t to violate the verifier's authentication with non-negligible probability.

Proof: Assume that $\mathcal{A}$ can impersonate a device DID_t to pass the verifier's authentication with non-negligible probability ε_1. In this lemma, we will present how challenge $\mathcal{C}$ solves *DUP* problem with non-negligible probability.

Given an instance $(\mathit{DID}_t, \mathit{PUF}_t, C_x, z)$ of *DUP* problem, $\mathcal{C}$ first initializes the system and publishes public parameters. Then, $\mathcal{C}$ simulates the authentication scheme by answering $\mathcal{A}'s$ queries as follows.

$h_i(m_j)$-Query: $\mathcal{C}$ keeps a set of lists $\{L_{h_i}\}_{i=1}^{3}$ as the tuple (m_j, r_j). If (m_j, r_j) exists in L_{h_i}, $\mathcal{C}$ returns r_j. Otherwise, r_j is generated by the random oracle and sent back to $\mathcal{A}$. And, the tuple (m_j, r_j) is inserted to L_{h_i}.

Extract (DID_i)-Query: $\mathcal{C}$ maintains a list of IoT devices' PUF instances $L_p = (\mathit{DID}_i, \mathit{PUF}_i)$, which will be initialized as empty. Each PUF instance presents a unique challenge-response behaviour. If DID_i exits in L_p, the instance PUF_i will be returned to $\mathcal{A}$. Else, C proceeds as follows.

- If $\mathit{DID}_i = \mathit{DID}_t$, $\mathcal{C}$ returns $(\mathit{DID}_t, \perp)$ to $\mathcal{A}$.
- If $\mathit{DID}_i \neq \mathit{DID}_t$, $\mathcal{C}$ constructs a unique PUF_i instance and returns $(\mathit{DID}_t, \mathit{PUF}_i)$ to $\mathcal{A}$, and updates L_p.

BC ($H_1(\mathit{DID}_i)$)-Query: $\mathcal{C}$ maintains a CRP list L_{crp} for each device, which is initialized as empty. $\mathcal{C}$ will randomly choose C_1 to challenge PUF_i and generate $(C_1, \mathit{HD}_1, R_1)$ to initialize L_{crp}. The L_{crp} will be updated once during each authentication request. In this query, $\mathcal{C}$ returns (C, HD) to $\mathcal{A}$.

BC ($H_1(\mathit{VID}_i)$)-Query: $\mathcal{C}$ will maintain a public key list L_k for all verifiers. If VID_i exists, $\mathcal{C}$ sends the public key pkv_i to $\mathcal{A}$. Otherwise, $\mathcal{C}$ generates (pkv_i, skv_i) and updates L_k. In this query, pkv_i will be sent to $\mathcal{A}$.

RevealPUFSecret ($\prod_{\Omega}^{l}$)-Query: $\mathcal{C}$ will find C_x, N, DID_i used in instance l. Then, the PUF instance PUF_i in L_p will be used to generate R_x. Last, $\mathcal{C}$ splits $R_x = R_{x,1} || R_{x,2}$ to calculate the PUF secret $(NR_{x,1} + R_{x,2})$ and returns it to $\mathcal{A}$.

ExtractPUFSlices (DID_i, C_i)-Query: In this query, $\mathcal{C}$ finds the according $R_{x,1}$ and splits it into n slices. Only less than t slices will be returned to $\mathcal{A}$.

Send ($\prod_{\mathit{DID}_i}^{l}$)-Query: If $\mathit{DID}_i = \mathit{DID}_t$, $\mathcal{C}$ aborts the game. As $\mathcal{C}$ does not know the real PUF instance of DID_t, the game should be aborted to ensure the correctness of this simulation. Otherwise, $\mathcal{C}$ answers the query as follows.

- When $\mathcal{A}$ sends ($\prod_{\mathit{DID}_i}^{l}$, *'start'*), $\mathcal{C}$ sends $M_1 = \{H_1(\mathit{DID}_i)\}$ to $\mathcal{A}$. If mutual authentication is required, $\mathcal{C}$ will choose $N_2 = n_2 \cdot P$ to compute and insert N_2 into M_1.
- When $\mathcal{A}$ sends ($\prod_{\mathit{VID}_i}^{l}$, M_1), $\mathcal{C}$ first retrieves (C_x, HD_x) by asking *BC ($H_1(\mathit{DID}_i)$)-Query*. Then, $\mathcal{C}$ chooses nonce N to construct $M_2 = \{C_x, \mathit{HD}_x, N, V_2\}$ will be returned to $\mathcal{A}$. If mutual authentication is required, $X = N_2 \cdot skv$ is computed and inserted into V_2 with $H_1(\mathit{VID}_i)$.
- When $\mathcal{A}$ sends ($\prod_{\mathit{DID}_i}^{l}$, M_2), $\mathcal{C}$ first checks whether $\mathit{DID}_i = \mathit{DID}_t$. Then, $\mathcal{C}$ retrieves n_2 and N generated in *($\prod_{\mathit{DID}_i}^{l}$, 'start')-Query*. Then, the public key pkv of the verifier is obtained by *BC ($H_1(\mathit{VID}_i)$)-Query* to verify whether M_2 is valid. If not, $\mathcal{C}$ aborts the game. Otherwise, $\mathcal{C}$ inputs (C_x, HD_x) in M_2 into the instance PUF_i queried in *Extract (DID_i)-Query* to generate R_x. R_x will be divided into $\mathit{Slice}_{x,1}$, $\mathit{Slice}_{x,2}$ to compute PS_x and V_3. Afterward, $\mathcal{C}$ sends $M_3 = \{N, V_3 = H_1(C_x || N || H_1(N || \mathit{PS}_x))\}$ to $\mathcal{A}$.
- When $\mathcal{A}$ sends ($\prod_{\mathit{VID}_i}^{l}$, M_3), $\mathcal{C}$ can directly obtain R_x, N, PS_x from previous queries. These elements could be used to verify the validity of M_3. At last, $\mathcal{C}$ will randomly choose a new challenge to generate $(C_{x+1}, R_{x+1}, \mathit{HD}_{x+1})$ to update L_{crp} for DID_i.

Finally, once $\mathcal{A}$ forges a valid login request $M_3^* = \{V_3 = H_1(C_x || N || H_1(N || \mathit{PS}_x))\}$, the solution to *DUP* problem can be found by $\mathcal{C}$. The legality of M_3^* means that $\mathcal{A}$ has queried the PUF secret in L_{h_1}. Thus, $\mathcal{C}$ first randomly chooses a tuple (m, r) from L_{h_1}. Then, $\mathcal{C}$ divides z already prepared in *DUP* instance into (z_1, z_2) to compute $(N^* z_1 + z_2)$. If $(N || N^* z_1 + z_2)$ equals to m, $\mathcal{C}$ can decide that z is the PUF response. The *DUP* problem will be solved.

Now, we compute the probability of solving *DUP* problem. Three events are defined as:

- E_1: $\mathcal{C}$ performs a right simulation and a useful attack ($\mathcal{C}$ aborts the *Send-Query*).
- E_2: $\mathcal{A}$ forges the valid M^*.
- E_3: $\mathcal{C}$ picks a right tuple from L_{h_i}.

We get $Pr[E_1] = (1 - 1/(q_s+1))^{q_s}$, $Pr[E_2|E_1] = \varepsilon_1$, and $Pr[E_3|E_1 \wedge E_2] = 1/q_h$, where q_s and q_h are the maximum times of *Send-Query* and *h_i-Query*. As a result, we get the probability as follows:

$$\begin{aligned}
&\Pr[E_1 \wedge E_2 \wedge E_3] \\
&\quad = \Pr[E_1] \cdot \Pr[E_1|E_2] \cdot \Pr[E_3|E_1 \wedge E_2] \\
&\quad = (1 - \tfrac{1}{q_s+1})^{q_s} \cdot \varepsilon_1 \cdot \tfrac{1}{q_h}
\end{aligned} \tag{2}$$

Since the probability of solving *DUP* problem is not negligible, it contracts the hardness of *DUP*. Therefore, $\mathcal{A}$ cannot impersonate a device to pass the verifier's authentication with non-negligible probability. *Lemma* 1 can be proved. □

Lemma 2: No P.P.T adversary $\mathcal{A}$ could impersonate the verifier VID_t to pass the authentication of the IoT device with non-negligible probability.

Proof: Assume that $\mathcal{A}$ enables to perform the impersonation of the verifiers with a non-negligible probability ε_2. We will show that $\mathcal{C}$ can solve *CDH* problem with non-negligible probability.

Given an instance $(P, n_2 \cdot P, skv \cdot P)$ of *CDH* problem. The $skv \cdot P$ denotes the public key of the verifier. $\mathcal{C}$ sets up the system, publishes parameters, and answers $\mathcal{A}$'s queries.

Note that h_i-*Query*, *Extract-Query*, *Send-Query*, *BC-Query*, *RevealPUFSecret-Query*, *ExtractPUFSlices-Query* are almost the same as the queries defined in *Lemma* 1. There are only three differences:

(1) *Extract (DID$_i$)-Query* will respond to all IoT devices' PUF instances.

(2) *Extract (VID$_i$)-Query* is added to simulate the registration of $\mathcal{A}$. To ensure the correctness, if $VID_i = VID_t$, return $(VID, \perp)$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ constructs (skv_i, pkv_i) and returns to $\mathcal{A}$, and then updates L_k.

(3) In *Send* ($\prod_{\Omega}^{l}$, m_j)*-Query*, if Ω is the verifier and m_j=M_1 contains N_2, $\mathcal{C}$ aborts the game. Because $\mathcal{C}$ does not know the verifier's private key. Otherwise, $\mathcal{C}$ continues the protocol and returns the message to $\mathcal{A}$.

Finally, $\mathcal{A}$ forges a valid login message $M_2^* = \{C_x, HD_x, N, H_1(VID), V_2\}$ to pass device's verification, where $V_2 = H_1(C_x||HD_x||N||H_1(VID)||X^*)$. It means that $\mathcal{A}$ has queried $X^* = n_2 \cdot pkv$ in the hash list. Thus, $\mathcal{C}$ can find X^* from L_{h_3} as a solution to solve *CDH* problem. As is explained in *Lemma* 1, the probability of solving *CDH* problem can be calculated as:

$$\Pr[Adv_{CDH}] = (1 - \frac{1}{q_s + 1})^{q_s} \cdot \varepsilon_2 \cdot \frac{1}{q_h}. \tag{3}$$

The probability of solving *CDH* problem is non-negligible, and it contradicts the hard assumption of *CDH* problem. Thus, *Lemma* 2 could be proved. □

Theorem 1: The mutual authentication-security of our authentication scheme will be achieved, if the hardness of *DUP* and *CDH* problems both hold.

Proof: The P.P.T adversary that breaks the MA-security of our scheme can forge login messages to pass the verification of devices or verifiers. However, *Lemma* 1 and *Lemma* 2 prove that no adversary could perform malicious impersonations. Therefore, Theorem 1 can be proved. □

D. Security Discussions

In this part, we discuss the security features and functions of our scheme.

1) Unidirectional authentication: To authenticate the IoT device, the verifier will send transactions to obtain the PUF secret $(N \cdot R_1 + R_2)$ from the blockchain and SPs. The authenticity of the IoT device will be verified by checking the validity of the hash value V_3. Only the valid IoT device has the unique PUF circuit to compute the correct PUF secret, which will then be inserted in V_3.

2) Mutual authentication: If mutual authentication is required, the IoT device will insert the random number $N_2 = n_2 \cdot P \in \mathbb{G}$ in M_1 and check whether the element $X = n_2 \cdot pkv$ is contained in the returned V_2. Only the valid verifier can use its private key to compute the correct $X = N_2 \cdot skv$, due to the hardness of computational Diffie-Hellman problem.

3) Recoverable PUFs-based encryption key: The recovery of PUFs-based encryption key is guaranteed by SSS system. As long as no less than t SPs are available, the TA will use the challenge C_r kept with the encryption data *ED* to require enough sub-slices from SPs and reconstruct the PUF response. As a result, the encryption keys generated from the hardware PUF circuit are always recoverable.

4) Secure PUF secrets: The PUF secret can be reconstructed by PUF secret slices, which are encrypted and shared to the slice user in the form of transactions. The attacker cannot impersonate a legal slice user to retrieve PSSs. Because SPs will query the user public key from the blockchain and verify the user's Schnorr signature to accept the sharing request. The security of Schnorr signature could be reduced to the discrete logarithm problem by the forking lemma [25]. Moreover, the on-chain information, such as the user public key, will be tamper-proofing and trustworthy, as long as the blockchain is not compromised.

In addition, each PUF secret slice is encrypted into the transaction by the element $X = (y \cdot \delta \cdot P)$. If *CDH* problem is hard, the adversary cannot compute the correct X from the given $y \cdot P$ and $\delta \cdot P$ to extract real PUF secret slices. Furthermore, only the transactions proposed by authorized participants (slice providers) in the consortium blockchain can be validated and recorded. In general, the security of PUF secrets could be guaranteed.

5) No online devices in data sharing: The shared data is uploaded to the cloud server and the data receiver needs to propose transactions to obtain the PUF secret from SPs. The PUF secret is utilized as the seed of key derivation function to recover the decryption key. The authorization of data sharing is supported by the concept of pay-as-you-go [27]. The data receiver will first pay the data sharing service through transactions and then obtain the decryption capability. This approach ensures that the data receiver does not require direct online device access to obtain PUF keys.

Then, we discuss how our scheme provides multidimensional CRPs security protection.

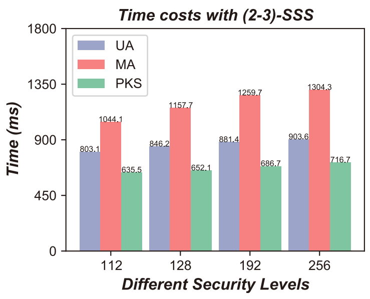
6) Resistance to impersonation attack: In our scheme, CRPs are distributed to SPs and stored in the form of SSS slices. The attacker may corrupt the storage of SPs or SPs would try to collude with each other.

Let $H(\cdot)$ be the Shannon's entropy function [16], [28]. The (t-n)-SSS with the secret S and secret set $(s_1, s_2, ..., s_n)$ is correct and perfect:

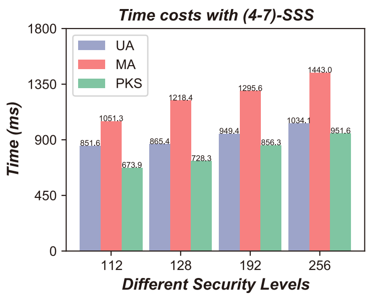
$$\begin{aligned} &\textit{Correctness}: H(S|s_1, s_2, \ldots, s_n) = 0, if(n \geq t). \\ &\textit{Perfectness}: H(S|s_1, s_2, \ldots, s_n) = H(S), if(n < t). \end{aligned} \tag{4}$$

However, the attacker can only get $m(< t)$ PUF slices from SPs, according to the threat model. Based on the *Perfectness* of SSS, the entropy function (4) shows $H(R_x|s_1, s_2, \ldots, s_{m_2}) = H(S)$. The attacker still learns nothing about the PUF response. Essentially, the threshold feature of SSS ensures the security of CRPs storage and prevent impersonation attack.

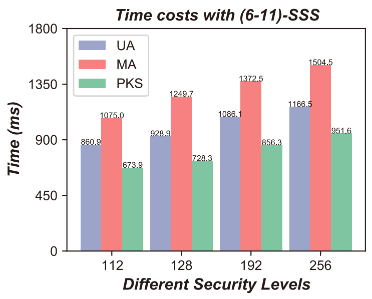
7) Resistance to available attack: SPs together leverage blockchain transactions to transmit PUF secret slices to distributed slice users and synchronize the CRPs usage order for each request. There is no trusted authority involved in the



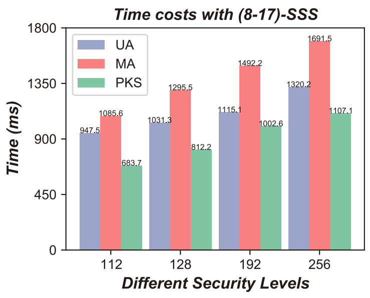
(a) (2-3)-SSS



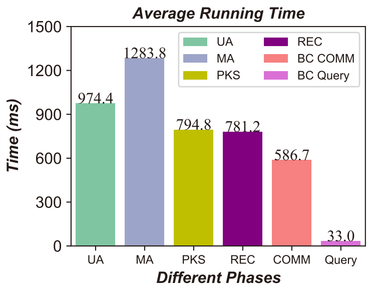
(b) (4-7)-SSS



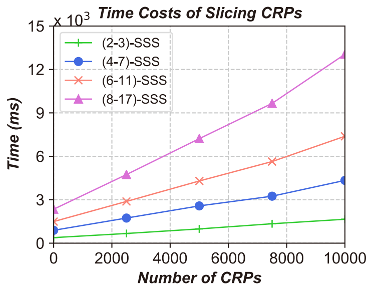
(c) (6-11)-SSS



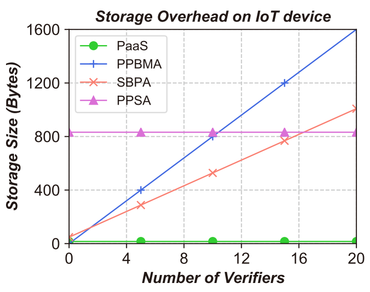
(d) (8-17)-SSS



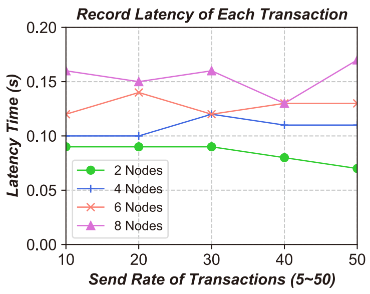
(e) Average Running Time



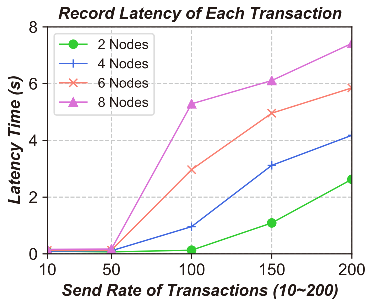
(f) Time of Slicing CRPs



(g) Storage Brought by Authentication



(h) Invoke Latency



(i) Invoke Latency

Fig. 8. Experiment results.

authentication and data sharing processes. As long as there are no less than t SPs alive, available attacks could be resisted.

8) Resistance to machine learning attack: While authenticating the IoT device, the verifier will obtain the one-time pad PUF secret $N{\cdot}Slice_1{+}Slice_2$, which is calculated by the newest CRP updated for each session. However, it is hard to determine both two slices from one equation to recover the original PUF response $R = Slice_1||Slice_2$. It means that no raw CRPs would be exposed to the curious verifier or attackers from the public channel. Therefore, the machine learning attack could be effectively prevented.

9) Resistance to single point failure: If the PUF circuit is broken and CRPs backup is lost, the TA will use the challenge kept with ciphertext to require enough sub-slices from SPs and reconstruct the PUF response. Based on the *Correctness* of SSS, once obtaining at least t PUF slices, CRPs used for data encryption and sharing still remain recoverable.

10) Resistance to replay attack: The random number mechanism is adopted by our scheme. Random numbers N, N_2, and $y_i \in \mathbb{Z}_q$ are inserted to ensure that communication messages are fresh and independent of other sessions. Therefore, our authentication scheme is free from the replay attack.

VII. Experimental Evaluation

In the experiment, we built a proof-of-concept prototype to analyze the actual running time, conduct the overhead comparisons, and evaluate the smart contract performance.

A. Experiment Settings

In our prototype, the PUF circuit, blockchain network, and involved entities were all implemented as below.

(1) PUF circuit: In this paper, the hardware circuit of XOR-APUFs was instanced on a Xilinx Virtex-5 FPGA board to provide hardware fingerprints. Our PUF circuit costs 421 LUTs, 214 registers. And, the generation of a 128-bit PUF response takes 2816 clock cycles with a clock frequency of 100 MHz (0.028 ms). The average reliability of the PUFs is 92.29 %, and the biggest noise is 10.8 % (14 bits). The BCH (127, 43, 29) was chosen to correct the biggest 14-bit error.

(2) Blockchain network: The consortium blockchain Hyperledger Fabric was run on a Linux system with 2 cores and 4-GB RAM to build the blockchain network. The configuration

TABLE I
COMPARISONS OF COMPUTATION[b] AND COMMUNICATION[a] OVERHEAD OF PAAS

Protocols	Computation Time on Device	Computation Time on verifier	Communication on Device
[10]	$6T_H+2T_{Ga}+T_{bp} \approx 2170.23$ ms	$5T_H+2T_{Ga}+T_{bp} \approx 134.97$ ms	2752 bits
[11]	$3T_{Gm}+13T_H \approx 144.47$ ms	$5T_{Gm}+12T_H+T_{MreDe}+T_{MreEn} \approx 111.28$ ms	2656 bits
[4]	$2T_H+T_{en} \approx 0.834$ ms	$4T_H+3T_{en} \approx 0.063$ ms	1280 bits
UA	$3T_H \approx 0.816$ ms	$n*(T_{Gm}+T_{Ga}+T_H)+T_H \approx 1.57*n+0.006$ ms	1152 bits
MA	$3T_H+2T_{Gm} \approx 94.504$ ms	$(n+1)*(T_{Gm}+T_H)+n*T_{Ga} \approx 1.57*n+3.148$ ms	1928 bits

[a] The communication overhead was calculated at 128-bit security level. The length of hash function $|H|$ is 256 bits, $|ID|, C, HD$ are all 128 bits, N is 64 bits. The ECC private/public keys are 256/520 bits. The number of SPs is n.

[b] To perform T_H hash function, T_{Gm} multiplication on $\mathbb{G}$, T_{Ga} addition on $\mathbb{G}$, T_{bp} bilinear mapping, T_{en} symmetric encryption, the IoT device respectively takes 0.272 ms, 46.98 ms, 0.136 ms 2168.47 ms, 0.29 ms. And, the verifier costs 0.006 ms, 1.568 ms, 0.004 ms, 132.94 ms, 0.013 ms. The multi-receiver encryption and decryption in [11] takes about 103.38 ms.

parameters *BatachTimeout* and *MaxMessageCount* are set as 50 ms and 10. The smart contract was realized by GO language. In addition, the blockchain client was constructed by Java-SDK and installed on the verifier/data receiver and SPs to invoke or query the smart contract. The performance of the smart contract was evaluated by Hyperledger Caliper.

(3) Entities: The relevant operations of entities were realized by Java 1.8 and BouncyCastle 1.60 library. The verifier and SPs were deployed on a laptop with 16 GB RAM and an I5 core. Moreover, the IoT device was realized on a Raspberry Pi 3b+, which communicates with our PUF circuit instanced on FPGA to exchange CRPs through the serial port.

B. Actual Time Costs

We implemented our system with different settings to analyze the actual running time. We configured four types of SSS with different combinations of t and n, including (2-3)-SSS, (4-7)-SSS, (6-11)-SSS, (8-17)-SSS. Besides, the security level was set from 112-bit to 256-bit.

First, the actual time costs of authentication (UA and MA), and PUFs key sharing (PKS) are shown in Fig. 8 (a) to Fig. 8 (d). The average running time is depicted in Fig. 8 (e). It is efficient for our scheme to spend only 974.45 ms and 794.80 ms on average on UA and PKS. Moreover, the reconstruction (REC) of PUF response takes about 781.23 ms. The MA process will need a bit more time, which costs 1283.79 ms. The blockchain communication (BC COMM) that transmits the PUF secret slices takes 586.7 ms for distinct requests. Overall, our PaaS enables to use various SSS settings to efficiently provide PUFs-based mechanisms at different security levels.

In addition, the results presented in these figures indicate that the running time increases step-wise upward with the increasing security level and the parameters of SSS. We further calculated the standard deviations of these time costs. The standard deviations of UA, MA, and PKS time are 136.5, 178.29, and 139.6. It means that the MA time is more sensitive to the variations in the security level and parameters of SSS. Because the IoT device should perform ECC cryptography during the MA process. Compared with verifiers and SPs, the IoT device needs much more efforts to finish high-security level cryptography algorithms (e.g., secp521r1, SHA-512).

Then, the time of slicing CRPs during the registration was evaluated. The relevant results are presented in Fig. 8(f). It shows that if the parameters of SSS are big (e.g., 8 and 17), it will take much time for the IoT device to slice CRPs. For example, when slicing 10000 CRPs, it takes the IoT device 13.051 s with (8-17)-SSS. However, the time that uses (2-3)-SSS is only 1.652 s. To make the registration phase more practical, the time of slicing CRPs should be limited to an acceptable value.

According to the above discussions, it is recommended that: (1) We should choose proper security levels in mutual authentication scenarios; (2) If the IoT device is very resource-constrained, it should properly adjust the number of the registered CRPs and parameters of SSS.

In general, our work can achieve different security levels and adapt to various settings of SSS. Meanwhile, the efficiency of PUFs-based security mechanisms deployed in various IoT scenarios could also be guaranteed.

C. Comparative Analysis

In this part, we compare the unidirectional and mutual authentication with relevant IoT authentication studies [4], [10], [11], which also involves CRPs management.

To evaluate the computation overhead, we count the most time-consuming basic cryptographic operations of IoT devices and verifiers in Table 1. For IoT devices, our UA service requires the minimum computation overhead. Apart from PUF operations, devices in UA only perform three hash functions, which approximately cost 0.816 ms. The computation overhead for the device in work [4] is close to UA service, which takes two hash functions and one symmetric encryption. The difference gap is only 0.018 ms. The work [10] puts the heaviest overhead on the IoT device, as the bilinear mapping that takes 2168.47 ms should be performed. The work [11] and our MA service both involve operations of scalar multiplication (46.98 ms) on $\mathbb{G}$. However, our MA service takes one T_{Gm} less than their work [11].

For the verifier, the studies [11], [10] take relatively heavy overhead, which needs 134.97 ms and 111.28 ms. Because multi-receiver encryption/decryption and bilinear mapping take up many computation resources. The work [4] takes the minimum computation overhead (0.063 ms). Moreover, the computation overheads in UA and MA are linear to the parameter n of SSS, as the verifier takes T_{Gm} to construct schnorr signature for each SP. However, the verifier could efficiently compute T_{Gm} (1.568 ms). Thus, our PaaS brings relatively low overhead to users (verifiers and data receivers).

To evaluate communication overhead, we only calculate the messages received and transmitted by IoT devices. The device in UA service takes the minimum communication overhead (1152 bits). The device in MA service communicates 648 bits more than work [4], since the additional element on $\mathbb{G}$ should be exchanged. The studies [11], [10] put relatively heavy communication overhead on IoT devices.

Finally, we compare the device's storage overhead brought by authentication with PPBMA [11], SBPA [29], and PPSA [4]. PPBMA and SBPA should store verifiers' certificates or public keys in the device. Hence, the overhead is linear to the number of verifiers, as is presented in Fig. 8(g). There is no need for devices in PaaS to store verifier's information. Each device will install a client to directly query verifier's public key from the blockchain. As a result, the storage size is only 16 bytes. Similarly, PPSA's storage for devices is also a constant value. However, it should keep a set of pseudo identities to achieve anonymity. Assume that there are 50 identities. The storage size of PPSA is 832 bytes.

To sum up, our PUF service has a comparatively satisfactory performance on computation, communication, and storage overheads, which are appropriate for IoT scenarios.

D. Performance of Smart Contract

We simulated four kinds of blockchain networks to evaluate the performance of the smart contract. These networks consist of different numbers of blockchain nodes.

The invoke latency evaluates the efficiency of the on-chain communications. This latency measures the time duration from the point that the transaction is proposed to the point that the proposed transaction is recorded in the blockchain. As is depicted in Fig. 8 (h) and Fig. 8(i), when the send rate is under 50 transactions per second (TPS), the latency is low and around 0.01∼0.15 s in all networks. However, the latency increases sharply with increasing concurrent transactions as shown in Fig. 8 (i). It is easy to find that the network with more nodes always has a larger latency time under the same send rate. It takes more time to reach the consensus and record the proposed transaction. Moreover, the query latency is always stable to be around 32.96 ms. Because data only needs to be retrieved from the local copy of the blockchain ledger.

Our PUF service needs to propose three transactions during each PUFs-based service request. Hence, the efficiency is determined by the invoke latency. However, when concurrent transactions per second are more than 50 in our simulation, the performance of the invoke latency is not satisfactory. Because the blockchain network implemented in our work only has a low throughput around 50 TPS.

It is recommended to choose blockchain networks with high throughputs to greatly reduce the invoke latency and maintain high efficiency. When deployed in industrial scenarios, it only needs to replace our simulated blockchain network with high-performance network, such as FastFabric [30]. The throughput of FastFabric could even reach 20000 TPS. Therefore, our scheme has the potential to efficiently process concurrent requests in industrial deployments.

VIII. In-depth and Extensive Discussions

In this section, we conduct additional discussions from a deep and broad perspective to show the promising prospect of our scheme.

A. Complexity and Usability

The integration of SSS system, blockchain, and PUFs increases complexity, but the impact brought to the usability is acceptable and worthwhile. We explain this point from the following three aspects:

(1) From the IoT device perspective: The increased complexity has little impact on IoT devices. Operations relevant to SSS and PUFs are all efficient and lightweight. The comparative analysis shows that computation overhead on IoT devices is satisfactory and suitable for most IoT devices in the distributed authentication environment, such as drones, vehicles, smart meters, etc. However, our scheme would not be regarded as super lightweight, as the hash function is involved. Extremely resource-constrained IoT environments, such as implanted medical devices and body area networks, cannot afford our scheme.

(2) From the verifier perspective: Our scheme put forward special requirements for verifiers, but the brought complexity is worthwhile. The verifier should install the blockchain client to interact with available blockchain nodes. Although extra overhead is brought to verifiers, it makes the deployment phase flexible, convenient, and reliable. Each verifier only needs to be registered in the blockchain, needless of adapting to the change of the SSS system.

(3) From the PaaS framework perspective: The integration of SSS and blockchain system has great potential to be acceptable by less technically advanced IoT environments. The blockchain is recently built as a key network infrastructure [18]. The cost of leveraging the blockchain system would be greatly reduced. Moreover, the PaaS framework in our three-layer architecture only interacts with the slice user layer and would not influence the IoT device layer too much. After properly improving the abilities of verifiers, most distributed IoT environments can be deployed in our system.

B. Scalability Analysis and Future Directions

We give a detailed analysis of how to scale our scheme in larger networks as follows.

To handle access requests and data flows from a vast number of devices, we should first deploy more verifiers to reduce response time and communication bandwidth. However, the key factor that restricts the scalability is the blockchain throughput and consensus latency. As is shown in Fig. 8(h∼i), if concurrent requests increase to a large number, the blockchain network cannot handle transactions in an efficient latency (0.01∼0.15 s). The latency even increases to several seconds and greatly influences the efficiency of authentication and key sharing. Fig. 8(i) also indicates that the number of blockchain nodes directly affects the blockchain performance. However, we cannot simply reduce blockchain nodes to improve performance. The number of blockchain nodes is closely related to the security of SSS slices, the storage capacity of slices, and adaptability to SSS parameters. Hence, we should increase the throughput and reduce the consensus latency as much as possible under a certain node scale. The mechanisms of blockchain sharding, node selection, and threshold signature can be properly combined to improve the blockchain performance. If the number of requests still exceeds the throughput of the blockchain too much, we will consider registering new devices into a parallelly deployed blockchain.

In future work, improving blockchain performance under a certain node scale would be formulated as a joint optimization problem using a Markov decision process [31]. We plan to use deep reinforcement learning to find the optimal strategy in an intelligence way.

C. Adaptability and Future-Proofing

In this part, we first discuss how our solution adapts to the impending quantum attack.

As one of the emerging security threats brought by the changing technologies, the quantum attack would be a real threat to current public-key cryptography systems. The public key system involved in our solution and blockchain system is elliptic curve cryptography, the security of which relies on the hardness of discrete logarithm problem. However, the combination of Grover algorithm (quantum search algorithm) [32] and side channel attack has the chance to break the elliptic curve cryptography [33].

To withstand quantum computing attacks, it is an inevitable choice to replace the public key algorithms in our scheme with post-quantum algorithms, such as lattice-based signature (CRYSTALS-Dilithium and Falcon), code-based public key encryption (McEliece). However, migrating our scheme as well as the blockchain system to post-quantum cryptography (PQC) presents a brand-new research area. Advancements in migration research will facilitate our scheme's adaptation to the era of quantum computing.

Then, we explain why our work would be future-proofed against upcoming technological advancements in IoT following four aspects:

(1) PUF circuit: The convenience of our scheme is that we separate the PUF circuit from the design of authentication and data sharing. The PUF circuit is only regarded as a security primitive to provide keys for devices. We can replace our XOR-APUFs with proper and advanced PUF circuits.

(2) Devices: Ongoing developments in integrated circuits and IoT hardware will enhance devices' storage, communication, and computing abilities. As a result, our scheme would have the potential to be applied to more IoT applications. We can even deploy mobile devices as verifiers to increase the flexibility and scalability of our scheme.

(3) Architecture: The promising AIoT [34] is the combination of Artificial Intelligence (AI) and IoT to support more intelligent and efficient applications. The edge servers are integrated in AIoT to afford AI computation tasks. The three-layer architecture of our system is consistent with the concept of edge computing and AIoT. Our solution has the potential to adapt to upcoming distributed IoT architectures and ensure the security of them.

(4) Blockchain: The sharding mechanism [35] is the frontier of blockchain and helps to greatly improve blockchain performance. The development of sharding mechanism would enable our scheme to better meet the scalability and efficiency requirements of future IoT environments.

D. Real-World Applicability

We first discuss the potential real-world applications. The applications of our scheme are divided into two types: latency tolerance and low latency. The first type is featured by providing large-scale data collection. Typical scenarios are smart grids, smart meters, Internet of Video Things, which focus on transmitting data securely. The latency brought by authentication and session key establishment can be tolerated. Differently, the core requirement of the second type is efficiency. In applications relevant to transportation, devices (drones or vehicles) need to keep real-time communications with road side units or base stations. The latency should be reduced as much as possible.

Then, we describe the challenges in deploying and operating our scheme in real-world applications, and provide corresponding recommendations and guidance.

(1) Deployment: We should pay more attention to the registration and update of CRPs. We recommend two ways of distributing CRPs. The first is recommended for initial registration. The user can directly get access to IoT devices and collect data using his own device. Then, the multi-factor (password, biometrics, verification code) authentication could be achieved between the user device and trusted authority to support securely uploading CRPs. The second way does not require human-machine interaction and is particularly suggested for system maintenance. In this method, the device uses its PUF circuit to authenticate with the trusted authority periodically and update CRPs automatically.

(2) Operation: We should seriously consider how to properly run the blockchain to adaptively achieve scalability and efficiency. In general, consortium blockchain networks, such as Hyperledger Fabric, FISCO, and Ethereum, could be chosen to implement our scheme. In real-time communication applications, consensus algorithms that reduce the latency should be selected to keep efficiency. The applications featured by large-scale data collection should use consensus algorithms with high throughput. If efficiency and scalability are all urgently required, it is recommended to deploy a new blockchain network in parallel when necessary.

IX. Conclusion

In this paper, we propose PUFs-based distributed authentication and recoverable data sharing with multidimensional CRPs security protection. We proved the security of our authentication and data sharing. The experimental evaluation shows that it only costs 974.45 ms, 1283.79 ms, 794.80 ms on average to complete UA, MA, and PKS in different settings. Moreover, the brought computation, communication, and storage overheads are comparatively satisfactory. When deployed within high-performance blockchain networks, our PaaS framework could efficiently process concurrent transactions in approximately 0.01$\sim$0.15 s. We also perform discussions from aspects of complexity and usability, scalability analysis and future directions, adaptability and future-proofing, real-world applicability, to show the promising prospect of our scheme. In conclusion, our work is efficient, practical, and suitable for IoT deployments.

REFERENCES

[1] Y. S. Gao, S. F. Al-Sarawi, and D. Abbott, "Physical unclonable functions," *Nature Electronics*, vol. 3, no. 2, pp. 81–91, 2020.

[2] Y. Zheng, W. Liu, C. Gu, and C. H. Chang, "Puf-based mutual authentication and key exchange protocol for peer-to-peer iot applications," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–18, 2022.

[3] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Examining smart-card security under the threat of power analysis attacks," *IEEE Transactions on Computers*, vol. 51, no. 5, pp. 541–552, 2002.

[4] M. N. Aman, U. Javaid, and B. Sikdar, "A privacy-preserving and scalable authentication protocol for the internet of vehicles," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 1123–1139, 2021.

[5] P. Gope and B. Sikdar, "An efficient privacy-preserving authenticated key agreement scheme for edge-assisted internet of drones," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, pp. 13 621–13 630, 2020.

[6] P. Gope and B. Sikdar, "Privacy-aware authenticated key agreement scheme for secure smart grid communication," *IEEE Transactions on Smart Grid*, vol. 10, no. 4, pp. 3953–3962, 2019.

[7] M. Barbareschi, V. Casola, A. D. Benedictis, E. L. Montagna, and N. Mazzocca, "On the adoption of physically unclonable functions to secure iiot devices," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7781–7790, 2021.

[8] S. Dai, H. Li, and F. Zhang, "Memory leakage-resilient searchable symmetric encryption," *Future Generation Computer Systems*, vol. 62, pp. 76–84, 2016.

[9] Y. Liu, B. Li, Y. Zhang, and X. Zhao, "A huffman-based joint compression and encryption scheme for secure data storage using physical unclonable functions," *Electronics*, vol. 10, no. 11, 2021.

[10] U. Chatterjee, V. Govindan, R. Sadhukhan, D. Mukhopadhyay, R. S. Chakraborty, D. Mahata, and M. M. Prabhu, "Building puf based authentication and key exchange protocol for iot without explicit crps in verifier database," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 3, pp. 424–437, 2019.

[11] 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.

[12] M. A. Qureshi and A. Munir, "Puf-rake: A puf-based robust and lightweight authentication and key establishment protocol," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2457–2475, 2022.

[13] G. Bansal, N. Naren, V. Chamola, B. Sikdar, N. Kumar, and M. Guizani, "Lightweight mutual authentication protocol for v2g using physical unclonable function," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7234–7246, 2020.

[14] B. Harishma, P. Mathew, S. Patranabis, U. Chatterjee, U. Agarwal, M. Maheshwari, S. Dey, and D. Mukhopadhyay, "Safe is the new smart: Puf-based authentication for load modification-resistant smart meters," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 663–680, 2022.

[15] U. Chaterjee, D. Mukhopadhyay, and R. S. Chakraborty, "3PPA: A private puf protocol for anonymous authentication," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 756–769, 2021.

[16] S. Chen, B. Li, Z. Chen, Y. Zhang, C. Wang, and C. Tao, "Novel strong-puf-based authentication protocols leveraging shamir's secret sharing," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[17] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, 1979.

[18] Y. Wang, Z. Su, J. Ni, N. Zhang, and X. Shen, "Blockchain-empowered space-air-ground integrated networks: Opportunities, challenges, and solutions," *IEEE Communications Surveys and Tutorials*, vol. 24, no. 1, pp. 160–209, 2022.

[19] H. Y. Kim, L. Xu, W. Shi, and T. Suh, "A secure and flexible fpga-based blockchain system for the iiot," *Computer*, vol. 54, no. 2, pp. 50–59, 2021.

[20] S. Goldwasser and S. Micali, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, p. 186–208, 1989.

[21] Y. Yu, Y. Zhao, Y. Li, X. Du, L. Wang, and M. Guizani, "Blockchain-based anonymous authentication with selective revocation for smart industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 5, pp. 3290–3300, 2020.

[22] L. Wu, J. Wang, K.-K. R. Choo, and D. He, "Secure key agreement and key protection for mobile device user authentication," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 319–330, 2019.

[23] X. Li, D. He, Y. Gao, X. Liu, S. Chan, M. Pan, and K.-K. R. Choo, "Light: Lightweight authentication for intra embedded integrated electronic systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1088–1103, 2023.

[24] U. Chatterjee, R. S. Chakraborty, and D. Mukhopadhyay, "A puf-based secure communication protocol for iot," *ACM Trans. Embed. Comput. Syst.*, vol. 16, apr 2017.

[25] J. Wang, L. Wu, K. Choo, and D. He, "Blockchain-based anonymous authentication with key management for smart grid edge computing infrastructure," *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1, 2019.

[26] H. Guo, Z. Zhang, J. Xu, N. An, and X. Lan, "Accountable proxy re-encryption for secure data sharing," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, pp. 145–159, 2021.

[27] T. Wang, H. Ma, Y. Zhou, R. Zhang, and Z. Song, "Fully accountable data sharing for pay-as-you-go cloud scenes," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 4, pp. 2005–2016, 2021.

[28] C. L. F. Corniaux and H. Ghodosi, "An entropy-based demonstration of the security of shamir's secret sharing scheme," in *2014 International Conference on Information Science, Electronics and Electrical Engineering*, vol. 1, pp. 46–48, 2014.

[29] J. Zhao, W. Bian, D. Xu, B. Jie, and H. Zhang, "A secure biometrics and pufs-based authentication scheme with key agreement for multi-server environments," *IEEE Access*, vol. PP, no. 99, pp. 1–1, 2020.

[30] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Conference Proceedings, pp. 455–463.

[31] Z. Yang, R. Yang, F. R. Yu, M. Li, Y. Zhang, and Y. Teng, "Sharded blockchain for collaborative computing in the internet of things: Combined of dynamic clustering and deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 9, no. 17, pp. 16494–16509, 2022.

[32] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, (New York, NY, USA), p. 212–219, Association for Computing Machinery, 1996.

[33] W. Chao, C. Lin, J. Hui-Hui, and H. U. Feng, "Ecc fault attack algorithm based on grover's quantum search algorithm with 0.1π phase rotation," *Journal on Communications*, 2017.

[34] T. Liu, J. Xia, Z. Ling, X. Fu, S. Yu, and M. Chen, "Efficient federated learning for aiot applications using knowledge distillation," *IEEE Internet of Things Journal*, vol. 10, no. 8, pp. 7229–7243, 2023.

[35] Y. Liu, X. Xing, H. Cheng, D. Li, Z. Guan, J. Liu, and Q. Wu, "A flexible sharding blockchain protocol based on cross-shard byzantine fault tolerance," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2276–2291, 2023.