TBAC: A Tokoin-based Accountable Access Control Scheme for the Internet of Things

Chunchi Liu, Minghui Xu, *Member, IEEE,* Hechuan Guo, Xiuzhen Cheng, *Fellow, IEEE,* Yinhao Xiao, Dongxiao Yu, Bei Gong, Arkady Yerukhimovich, Shengling Wang, and Weifeng Lyu

Abstract—**Overprivilege attack**, a widely reported phenomenon in IoT that accesses unauthorized or excessive resources, is notoriously hard to prevent, trace and mitigate. In this paper, we propose TBAC, a Tokoin-Based Access Control model enabled by blockchain and Trusted Execution Environment (TEE) technologies, to offer fine-grained access control and strong auditability for IoT. TBAC materializes the virtual access power into a definite-amount, secure and accountable cryptographic coin, termed "tokoin" (token+coin), and manages it using atomic and accountable state-transition functions in a blockchain. A tokoin carries a fine-grained policy defined by the resource owner to specify the requirements to be satisfied before an access is granted, and the behavioral constraints that describe the correct procedure to follow during access. The strong-auditability is achieved with blockchain and a TEE-enabled trusted access control object (TACO) to ensure that all access activities are securely monitored and auditable. We prototype TBAC by implementing all its functions with well-studied cryptographic primitives over different blockchain platforms, building a TACO on top of the ARM Cortex-M33 TEE microcontroller, and constructing a user-friendly APP for regular users. A case study is finally presented to demonstrate how TBAC is employed to enable autonomous and secure in-home cargo delivery.

Index Terms—Fine-grained Access Control; Access Procedure Control; Auditability; Overprivilege Attack; Blockchain; Trusted Execution Environment; IoT.

1 Introduction

With the rapid development of the Internet of Things (IoT), IoT devices have become much smaller, smarter, and more prevalent than ever before. Unfortunately, it has been widely reported that a variety of mainstream IoT devices and platforms such as Google Home [1], Amazon Alexa [2], [3] and Samsung SmartThings [4] have been secretly accessed by attackers without authorization. With the household penetration of smart home devices reaching 36.6% by the end of 2020 and being expected to hit 57.2% by 2025 [5], increasingly grave security threats have been posed to ordinary users and their everyday life.

Unauthorized access reveals the open challenge of **overprivilege attacks** in IoT security. Such attacks include 1) access unauthorized resources, 2) access excessive resources than granted, and 3) access while not complying with the access constraints – all are wished to be done secretly. Such attacks are common in practice, as Felt *et al.* reported that overprivilege vulnerabilities exist in over one-third of the current Android-driven IoT devices [6].

Overprivilege attacks are mainly caused by three significant design deficiencies of current IoT access control schemes: 1) **coarse granularity** of the access policy, 2) **lack of access procedure control**, and 3) **weak auditability** towards all access activities. Coarse granularity refers to a low expressiveness of the access policy – an inability to precisely define the required conditions to grant the access, the exact amount of resource to release, and the strict process regarding how the resource should be used. Lack of access procedure control refers to the missing of continuously monitoring how the resource is actually used, i.e., whether the access procedure strictly follows what are defined by the access policy, after access is granted. Weak auditability is an inability to securely log all access activities in detail. Coarse granularity is the direct cause of overprivilege attacks, while lack of access procedure control leads to the oversights of such attacks on the spot, and weak auditability implies the impotence of holding the attacker accountable. Many IoT access control mechanisms have been proposed in recent years, but unfortunately almost all of them suffer from one or more of the deficiencies mentioned above [7].

In this paper, we present TBAC, a Tokoin-Based Access Control model. The design objective of TBAC is to provide fine-grained access control for IoT applications that can not only verify the conditions for granting the access rights but also regulate the access procedure ensuring that the access policy is strictly followed in whole, with all activities logged

- *C. Liu was with the Department of Computer Science, The George Washington University and now with Ernst & Young. Email: liuchunchi@gwu.edu*
- *M. Xu, H. Guo, X. Cheng, and D. Yu are with the School of Computer and Science and Technology, Shandong University. Email: {mhxu, xzcheng, dxyu}@sdu.edu.cn, ghc@mail.sdu.edu.cn*
- *Y. Xiao is with the School of Information Science, Guangdong University of Finance and Economics. Email: 20191081@gdufe.edu.cn*
- *B. Gong is with Beijing University of Technology. Email: tekkman_blade@126.com*
- *A. Yerukhimovich is with the Department of Computer Science, The George Washington University. Email: arkady@gwu.edu*
- *S. Wang is with Beijing Normal University. Email: wangshengling@bnu.edu.cn*
- *W. Lyu is with Beihang University. Email: lwf@nlsde.buaa.edu.cn*

Corresponding author: Minghui Xu.

for auditing purpose. The access policy is carried by an accountable digital asset, namely a **tokoin** (token+coin), which is managed securely and auditably in the form of a cryptographic coin with the assistance of blockchain and Trusted Execution Environment (TEE) technologies. A tokoin can only be created, modified and revoked by the resource owner, and redeemed by a legitimate subject (user). In other words, to access an IoT resource, one needs a tokoin issued by the resource owner and redeems the resource by strictly following the policy carried by the tokoin. In this paper, we detail TBAC by making the following contributions.

First, by combining token and coin together, we materialize the "virtual" access right into a definite-amount, cryptographically-secure and accountable digital asset that is a tokoin. A tokoin represents both the access right and the credential to redeem it. With the tokoin, succeeding in an unauthorized access is as hard as forging a cryptographic coin; thus a resource owner can have high confidence of all upcoming access activities since only he can create, modify, and revoke the tokoin. By this way the resource owner can take full control of the access to his resource down to each access activity and behavior, without the need of delegating or relying on any third party such as a server and thus avoiding the risk of being secretly accessed without authorization. Benefiting from this design, we achieve flexibility in secure peer-to-peer access privilege delegation and redistribution, enabling new applications such as IoT resource rental and trading of IoT resource right-of-use.

Second, a tokoin carries the so-called ***4W1H*** access policy specified by the resource owner, which defines the access conditions to be met and the access procedure to be followed. The *4W1H* stands for ***who*** *is allowed to do* ***what*** *at* ***when*** *in* ***where*** *by* ***how***, where the *4W* present the static access conditions that must be satisfied in order for an access request to be granted, and the *1H* describes the dynamic access behavioral constraints specifying how to properly use the resource after access is granted. Such a fine-grained policy is carried by a tokoin, stored in a blockchain, and enforced by TEE. Specifically, TBAC adopts a blockchain for secure tokoin storage and atomic tokoin transfer, and employs a TEE-enabled trusted access control object (*TACO*) to verify whether or not the access conditions and the access procedure follow the predefined *4W1H* policy. TACO does the following: 1) collects status to make access decisions, 2) monitors the actual resource-using process and identifies misbehaviors, and 3) records proofs of misbehaviors (if occur) and makes them publicly verifiable on the blockchain. One can see that TBAC achieves fine-graininess, access procedure control, and strong accountability by adopting tokoins that carry owner-defined *4W1H* access policies and employing the blockchain and TEE technologies to securely log all on-chain/off-chain activities.

Third, a complete and effective access control scheme should implement the three processes of Authentication, Authorization and Auditing to ensure that an authentic subject is authorized to access the right amount of resource, no more no less, under verifiable conditions following a fully accountable procedure. We define all the tokoin functions and detail their implementations to demonstrate how these three processes are realized. We also provide Go-Tokoin and Ethereum-Tokoin, the two prototypes of the TBAC scheme, with the former following the native design (Tendermint-BFT) for best performance and the latter showing the adaptivity of TBAC to mainstream blockchain platforms (Ethereum). The TEE-enabled access control object is implemented in the ARM Cortex-M33 based microcontroller protected by the ARMv8-M TrustZone, to securely sample the physical environment for access condition verification and access procedure monitoring. To the best of our knowledge, we are the first to develop applications on the Cortex-M series TEE microcontrollers, which are specifically designed for low-cost trustworthy embedded systems by supporting hardware-level program security isolation. As the Cortex-M series TEE microcontrollers offer very little usable libraries, we build our access control object TACO roughly from the bare metal level, thus facing a great engineering challenge. A friendly TBAC Android App is developed for convenience and better user experience. All codes are open-sourced and available at Github.

Fourth, we present a TBAC-assisted in-home cargo delivery case study to demonstrate how TBAC is utilized for real world applications. This case study permits autonomous in-home delivery, in which a deliveryman can open the smart door and put the cargo down on his own in the designated area (e.g., mud area). It involves an access procedure control mechanism to guarantee that the deliveryman's actual behavior does not violate the access policy (e.g., not entering the main room) and ultimately the home owner's physical security. More importantly, this case study provides a seamless integration of blockchain and IoT to extend the on-chain trustworthiness to off-chain physical world. This piece of work itself has its own significance in many IoT applications that require trusted management. Our case study also shows that secure and accountable accesses to physical resources can be achieved by techniques that are used to be only available in the digital world.

This paper is organized as follows. We first present the most related work in Section 2. Then we propose our tokoin based access control model TBAC in Section 3 and detail its implementation in Section 4. The TBAC-assisted in-home cargo delivery case study is reported in Section 5. We discuss additional issues of TBAC and future research in Section 6.

2 Related Work

With IoT devices integrating deeper and deeper into our daily life, the challenge of overprivileged accesses becomes more and more grave since it extends digital threats to real-life safety. To address this problem, the IoT security community has conducted research along two major directions: security analysis and access control mechanism design, with the former aiming to discover existing device/platform's overprivilege vulnerabilities and fix them, while the latter intending to prevent overprivileged accesses by design. In the following we discuss their recent advances.

2.1 Mitigating Overprivilege Attacks through Security Analysis

Security analysis based countermeasures consider overprivilege vulnerabilities as programming deficiencies. They attempt to mitigate the problem by identifying deficiencies

based on learned characteristics. Fernandes *et al.* [4] uncovered a severe overprivilege vulnerability in the Samsung SmartThings platform, which allows attackers to falsely turn on a fire alarm. This work does not consider defensing such an attack. Zhang *et al.* [8] discovered that attackers can create malicious functions on the Amazon skill platform to launch overprivilege attacks and perform eavesdropping activities through Alexa. To mitigate this threat, the authors developed a skill-name scanner and a context-sensitive detector to filter out problematic functions. Jia *et al.* [9] presented a graph-based algorithm to automatically excavate the overprivilege weaknesses in smart home authentication protocols. Celik *et al.* [10] hand-crafted 20 common flawed apps, with which a model-checking based solution was developed to automatically identify the flaws.

In summary, the mechanisms mentioned above mitigate the overprivilege challenge by first discovering the vulnerabilities, then learning their features, and finally detecting their presence in a general environment. Their false-negative rates are usually high, and they may fail to detect emerging vulnerabilities (e.g., zero-day vulnerabilities).

2.2 Mitigating Overprivilege Attacks through the Design of New IoT Access Control Schemes

As discussed earlier, the overprivilege challenge is mainly caused by the following three design deficiencies of common access control mechanisms: 1) coarse granularity of access policy, 2) lack of access procedure control, and 3) weak auditability towards access activities. While an ideal IoT access control scheme should address all these three concerns, yet many, if not all, recently proposed methods fail to do so.

Works That Improve Auditability: When designing new access control schemes with better auditability, many researchers resort to blockchain utilizing its highly auditable nature in recent years. However, most blockchain-based access control schemes just simply combine blockchain with existing solutions, leaving coarse-granularity and the lack of access procedure control unaddressed. For examples, Zhang *et al.* [11] developed an ACL-based access control scheme on top of multiple Ethereum smart contracts, following the classic coarse-grained Unix style ACL $\{Read, Write, Execute\} \rightarrow \{Allow, Deny\}$; Xu *et al.* [12] proposed BlendCAC, which employs Ethereum smart contracts in replacement of the traditional capability access server to issue and manage coarse-grained Linux-style access capabilities; Maesa *et al.* [13] adopted smart contracts to realize attribute-based access policies, successfully transforming policy evaluations to smart contract executions. Sun *et al.* [14] presented a secure, lightly-weighted, and cross-domain IoT access control system, using blockchain to record the IoT entities' attributes, policy files' digests, and access decisions. All works mentioned above overlook the access procedure control; therefore it is hard to ensure that a resource is not abused after release, which obviously could make the resource vulnerable to overprivilege attacks.

Works That Improve Granularity and Access Procedure Control: Since coarse access control granularity is the main cause of overprivilege challenge, context-based and situational-aware access control emerge as new research directions for the purpose of obtaining more expressiveness and accuracy of access policies. Context-based access control takes consideration of the current status of the IoT devices/apps such as inter-procedure controls, data flow levels and call-stacks as parts of the control strategy. Such schemes increase the access granularity by more explicitly defining whitelisted data flow or control flow patterns and prohibiting all others. Typical works include [15], [16], [17], [18], [19]. Sergio *et al.* [15] proposed a capability-based IoT access control system that supports access right delegation and sophisticated access control customization. Tian *et al.* [16] designed SmartAuth that collects security-relevant information from an IoT app's description, codes and annotations to learn the app's actual functionality towards an IoT device, then guides the users to create more precise access policies for the app. This effectively increases the app's resource access granularity and bridges the gap between the functionalities explained to the user and the operations the app actually performs. Jia *et al.* [17] presented ContexIoT, a patch to Samsung SmartThings Platform Apps that can gather essential information from the environment of all the variables along the execution path of a security-sensitive action. The context is then sent to the backend permission-checking server for decision making and auditing. Yahyazadeh *et al.* [18] proposed EXPAT that records user expectations of how to use the resource and save them as access policies. EXPAT continuously monitors the app runtime to ensure that the user expectations are never violated. This mechanism increases access granularity and provides a certain degree of access procedure control. Bhatt *et al.* [19] developed a formal attribute-based access control model to enable fine-grained access control in AWS IoT. Situational-aware access control was studied by Schuster *et al.* [20], who presented an Environmental Situation Oracle (ESO) that senses the physical statuses and exposes a simple interface to tell the caller whether a "situation" (such as a human behavior) is active or not. This allows a finer granularity for making access decisions and can continuously monitor the access procedure after the resource is released. Maanak *et al.* [21] presented the concept of activity-centric access control for IoT devices, which identifies different entities involved along with the important factors to make an access control decision.

Multi-user Access Control: There exist recent access control works focusing on supporting multi-user control towards IoT devices. Zeng *et al.* [23] proposed a multi-user access control scheme for generally non-adversarial smart home environments. They discussed privacy challenges, tensions and disagreements, and access power imbalances among multiple users that share the same IoT device through user studies and survey feedbacks. He *et al.* [22] conducted a similar research in which access control and authentication for home IoT were re-envisioned through a 425-participant user study to investigate user tolerance to system malfunctions, default policy preferences, and the impacts of contexts on access decision making processes. Shantanu *et al.* [26] developed an access control architecture for constrained healthcare resources in IoT, which employs attribute-based, role-based, and capability-based mechanisms to reduce the number of policies required during multi-user access control. Sikder *et al.* [24] presented

	Major Techniques	Fine-grainess	Auditability	Access Procedure Control
IoTJ'18 Zhang et al. [11]	Smart Contract & Unix-style ACL	✖	✔	✖
iThings'18 BlendCAC [12]	Smart Contract & Linux-style Access Policy	✖	✔	✖
iThings'18 Maesa et al. [13]	Smart Contract & Attribute-based Access Policy	✔	✔	✖
Access'21 Sun et al. [14]	Policy-based Access Control	✔	✔	✖
MATH COMPUTE MODEL'13 CapBAC [15]	Capability-based Authorization	✔	✖	✔
USENIX'17 SmartAuth [16]	Compare what is claimed to be offered and what is actually offered	✔	✖	✔
NDSS'17 ContexIoT [17]	Gather environmental info for decision making	✔	✔	✖
SACMAT'19 EXPAT [18]	Take user expectations	✔	✖	✔
Access'21 AWS-IoTAC [19]	Attribute-based Access Policy	✔	✖	✖
CCS'18 Schuster et al. [20]	Environmental Situation Oracle	✔	✖	✔
SACMAT'21 Maanak et al. [21]	Activity-centric access control	✔	✖	✔
USENIX'18 He et al. [22]	User study	✔	✖	✖
USENIX'19 Zeng et al. [23]	User study	✖	✔	✖
WiSec'20 Kratos [24]	Policy manager to resolve conflicts	✔	✔	✖
JISIS'13 Hernandez et al. [25]	Distributed Capability-based Access Control	✔	✖	✖
J NETW COMPUTE APPL'19 Shantanu et al. [26]	Policy-based Access Control	✔	✖	✖
Tokoin-Based Access Control	Blockchain & TEE	✔	✔	✔

TABLE 1: Comparison between TBAC and the latest access control schemes. Legend ✔ means the work has the corresponding property and ✖ indicates that the work does not possess the property.

Kratos to handle the challenges brought by multi users sharing multiple devices with conflicting and dynamically changing demands in a smart home environment. Kratos employs an interaction module to collect different access control settings from the users, translates them into policies at a backend server, and finally analyzes the policies, resolves the potential conflicts, and generates the final policy via a policy manager. These interesting works target multi-user access control, tackling the difficulties orthogonal to the overprivilege challenge under our study.

2.3 Summary

The works mentioned above increase one or more perspectives of access control. Yet the following two vulnerabilities are still not sufficiently addressed. 1) The log to access activities may be subject to deletion, leaving IoT devices un-auditable and vulnerable to unlimited uses (an unlimited legal use is also an abuse). This can be better handled with blockchain-based schemes but often neglected by other mechanisms. In TBAC, access privileges (tokoins) are definite-amount, meaning that a tokoin can designate the exact amount of accesses and no unauthorized access can take place. 2) How a user is actually using the resource may not be fully monitored after access is granted. Access procedure control is only achieved by very few works such as app runtime analysis or physical monitoring as we know by now. In TBAC, we introduce access procedure control that continuously monitors the dynamic situations during the access activities. If a misbehaving situation is detected then we freeze the resource, record the proof of abused-access and make it public on blockchain. A comparative study on TBAC and the latest access control schemes mentioned above is presented in Table 1.

3 TBAC: Tokoin-Based Access Control

In this section, we first present our threat model and security assumptions, then provide a high-level overview on the TBAC protocol followed by its model definition, and finally discuss the security strengths of TBAC.

3.1 Threat Model and Security Assumptions

Assume that an attacker $\mathcal{A}$ has access to all public TBAC functions. From a cryptographic perspective, we consider the following adversary model,

1) **Probabilistic Polynomial Time (PPT) Attacker.** The attacker $\mathcal{A}$ can arbitrarily deviate from the protocol and perform any computationally efficient attack that runs in polynomial time and may use randomness to produce non-deterministic results.

From the perspective of IoT, we consider authorization-level security attacks by $\mathcal{A}$:

2) **Shallow-level Compromise.** The attacker $\mathcal{A}$ can penetrate local area network (LAN), lure the user to install a malicious app on the non-secure zone of the IoT control hub (*TACO* in our system) and gain user mode control.

The attack goal is to perform an unauthorized access or overprivileged access without leaving evidence:

- **Unauthorized Access.** An external attacker $\mathcal{A}_{ext}$ aims to perform an unauthorized access while no access privilege has been granted. This is usually done by exploiting security vulnerabilities in platforms [3] [27], devices or protocols [4] [28] [29].
- **Overprivileged Access.** An internal attacker $\mathcal{A}_{int}$ with only limited access privilege aims to perform an over-privileged access. This is usually caused by coarse granularity of the access policy or resource abuse after access is granted [30].
- **Covering Up Illegal Access:** The attacker intends to erase or hide the illegal access activities.

Our main security goal is to prevent privilege elevation at the authorization stage and achieve trustworthy access history audit.

Additionally, we assume the following building blocks are secure, thus no PPT adversary can obtain more than negligible advantage in compromising them.

- **Blockchain:** In this paper, we assume blockchain is a tamper-resistant distributed ledger.
  - Availability: Any data written to blockchain can be correctly retrieved by a blockchain node within a bounded time. Network-level attacks such as DDoS and eclipse are out-of-scope of this paper.
  - Integrity: Any data in blockchain is tamper-resistant.
- **Trusted Execution Environment (TEE):** We assume programs deployed in a TEE secure zone is tamper-resistant. Physical damage to TEE is beyond the scope of this paper.

3.2 High-level Overview on TBAC

We consider a large-scale IoT system that takes a three-layer (device-edge-cloud) architecture in which devices are in charge of environment sensing and edge/cloud layers host a secure distributed ledger and the device servers. Our TBAC constitutes three components: 1) a blockchain system to manage tokoin and its carried access policy, 2) a TEE-based trusted access control object (*TACO*) to sense the physical world, make actual access decisions, and monitor the access procedure, and 3) an App-based user interface. The blockchain protects the security of on-chain tokoin management while *TACO* guarantees that the off-chain resource access strictly follows the policy specified in a tokoin.

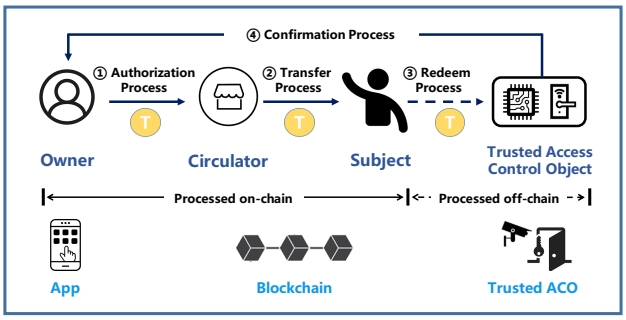


Fig. 1: Overview on TBAC

TBAC contains four steps starting from the creation of a tokoin and ending when the mission of the tokoin is completed, as shown in Fig. 1:

① **Authorization Process:** a resource owner creates a tokoin carrying an access policy. Note that the resource owner can only control accesses to the resources he/she owns.

② **Transfer Process:** a circulator transfers the tokoin arbitrarily to any other circulator or a subject through standard coin trading functions. Freedom of transfer changes only the holder of the tokoin – it does not change the invariant that only a legitimate subject can redeem the tokoin.

③ **Redeem Process:** a legitimate subject receives the tokoin from a circulator, and redeems it to the trusted access control object (*TACO*). *TACO* receives the tokoin, retrieves the corresponding access policy, checks if the current physical status meets the access conditions and make actual access decisions.

④ **Confirmation Process:** if access is granted, *TACO* monitors the use of the resource afterwards until the access activity finishes. If the subject abuses the resource, which violates the access policy, *TACO* takes appropriate actions with the captured proof uploaded to blockchain.

Each of these processes is initiated through a blockchain function call. There also exist other functions such as tokoin revocation and modification, which are used by the resource owner to revoke or revise a tokoin before it is redeemed. We denote the entity roles in the above TBAC process as $\mathcal{R} = \{r_O, r_S, r_C, TACO\}$, in which r_O refers to the resource owner, r_S a subject, r_C a circulator, and *TACO* the trusted access control object. Let r_H denote the current holder of a tokoin, which could be any of $\{r_O, r_S, r_C\}$. Note that r_S can be a group of legal identities instead of one specific subject, which provides flexibility in practice as the exact subject to redeem the resource may not be known when the tokoin is created.

```
Policy 1:
    Subject:
        0x40dCaF065caF80004342c1A9f3bcdC83A01e40bc
    Resource:
        com.example.mysolution:00000001828
    Time:
        22-02-16T12:05:00Z,
        22-02-16T12:15:00Z
    Location:
        (31.24011, 121.49790),
        (31.23982,121.49861),
        (31.23923,121.49757),
        (31.23905,121.49834)
    Procedure:
        PPC 1: Trespassing
            200
        PPC 2: Duration
            10m
```

Fig. 2: An example tokoin access policy.

3.3 TBAC Model Definition

The core to TBAC is a fine-grained access policy constructed to precisely specify what exact resources should be released under what circumstances and how the resources can be used. Accordingly, we propose the 4W1H access policy which defines *Who* can access *What* resource in *Where* at *When* by *How*. The *4W* refers to the static access conditions to be met based on which the access is granted, and the 1H condition defines a dynamic process of *how* to use the physical resources after the access is granted. This access policy, together with the processes of *access condition verification* and *access procedure monitoring*, can enforce the correct use of the physical resource.

In the actual syntax of the 4W1H policy, the 4W condition can be intuitively set. As seen from our given example in Fig. 2, the subject field is the address of a single subject or an identifier of a group of subjects[1], the resource field is the ID or address of the resource, the time field is the time slot to access the resource, and the location field is a geo-fence in which the resource can be accessed. The 1H condition is specified as a list of *perspective procedural constraints* (PPC). Each PPC restrains the subject's behavior on a degree-of-freedom of control he receives towards the resource. For example, if we give a permission to someone to adjust a thermostat, we restrain what temperature range the subject can set; if we allow the subject to unlock a door, we restrain how far the subject can enter the room and how long he can stay. In Fig. 2, two PPCs are defined, with PPC 1 specifying the farthest line the subject can reach before a trespass is detected and PPC 2 the duration of this access process.

Next we define TBAC, the Tokoin-based Access Control model, and its nine high-level functions in Fig. 3: Gen, Verify, Create, Transfer, Modify, Revoke, Redeem,

1. One can use a cryptographic accumulator to register an identity into a group and verify whether or not an identity is included in the group.

Tokoin-based Access Control (TBAC) Model Definition

TBAC consists of the following nine polynomial time functions:

- Gen(s): Takes a master security parameter s and generates a public/private key pair (pk, sk) for each participant.[1]
- Verify(t, σ_{sk}, f_{id}): Given a tokoin t, a signature σ_{sk}, and a function-call handler f_{id}, where $f_{id} \in \{$*Create*, *Transfer*, *Modify*, *Revoke*, *Redeem*$\}$, output '1' if the signature σ_{sk} is signed by the function caller pk, tokoin t is valid, and pk has the permission to perform function f_{id} on t. All the following functions must first call Verify to authenticate tokoin validity and caller identity.
- Create(*policy*): Given an access policy *policy*, produces a new tokoin t and sets its owner to caller pk.
  - pk then acquires role r_O.
- Transfer(t, pk'): Transfers the ownership of tokoin t to pk'.
  - Requires caller is r_H
- Modify$(t,$ *policy**): Updates tokoin t with a new access policy *policy**.
  - Requires caller is r_O
- Revoke(t): Nullifies a tokoin t.
  - Requires caller is r_O
- Redeem(t): Given a tokoin t, on PolicyCheck$(t) = 1$, grants access to the IoT resource D.
  - Requires caller is r_S and PolicyCheck$(t) = 1$, otherwise transfers t back to r_S.
- PolicyCheck: *TACO* samples the current environment and verifies whether the pre-access conditions and access procedure are both compliant to *policy*. Output '1' if yes and '0' otherwise.
- Auditing: Logs all function calls and writes them to tokoin metadata.

1. For the ease of presentation we use the public key pk to represent the participant's identity.

Fig. 3: TBAC Model Definition and Its Functions

PolicyCheck, and Auditing. These functions describe ideal functionalities to achieve, which are necessary for any TBAC implementation regardless of which platform to build on because they constitute the Authentication, Authorization, and Auditing (AAA) processes that are critical to any access control scheme [31]. Specifically, the Authentication process is an act of establishing or confirming the identity or capability as authentic, therefore it includes functions Verify and PolicyCheck; the Authorization process determines whether a person or a process is authorized to perform a given access activity, thus it includes Create, Transfer, Modify, Revoke, and Redeem, with Create creating a tokoin defining the access privilege while Transfer, Modify, and Revoke making proper modifications to the tokoin capability, and Redeem finally taking back the tokoin capability and redeeming the agreed resource; and the Auditing process is obviously implemented by function Auditing, which makes accountable audits over all activities and modifications to the access tokoin.

We detail our TBAC implementation in the next section, where we show the structure of tokoin t in a tuple format, the corresponding block structure and the exact steps in implementing these nine functions. For the ease of reading, we collect the major notations and their abbreviations used in this paper in Table 2, and summarize the properties of TBAC as follows.

Security: No tokoin can be falsely created, tampered with, redeemed, or revoked by an adversary. It prohibits unauthorized uses of IoT resources.

Fine-Grained Access Control: A tokoin carries both static restrictions for pre-access condition verification in order for the requested access to be granted, and procedural restrictions for during-access behavior obedience monitoring to avoid possible abuse of the resource. It prohibits overprivileged uses of IoT resources.

Auditability: Usage and changes of tokoins are securely audited. It prohibits the concealment of unauthorized or overprivileged access activities over the IoT resources.

Direct Access Privilege Management: The IoT resource owner can directly manage the access privileges towards his resource with trust and transparency, without the need of any third party access server.

Description	Notation/Abbreviation
Resource Owner	r_O
Tokoin Holder	$r_H = \{r_O, r_C, r_S\}$
Circulator	r_C
Subject	r_S
Resource	D
Trusted Access Control Object	*TACO*
Perspective Procedural Constraints	PPC
Function-call Handler	f_{id}
Access Policy	*policy*
Public Key and Private Key of participant i	pk_i, sk_i
Signature	σ
tokoin	t

TABLE 2: Notations and Abbreviations

3.4 Security Discussions

As discussed in the threat model, TBAC faces two kinds of attacks: unauthorized access performed by external adversary $\mathcal{A}_{ext}$, and overprivileged access performed by internal adversary $\mathcal{A}_{int}$, with both wishing to be done secretly. The difference between $\mathcal{A}_{ext}$ and $\mathcal{A}_{int}$ is that the former does not possess a valid tokoin while the latter does.

There are three ways for $\mathcal{A}_{ext}$ to perform an unauthorized access: (A) stealing or forging a valid tokoin; (B) replaying a *Redeem* message, pretending this message is sent by a legitimate subject; or (C) penetrating the TEE secure zone to manipulate access decisions. $\mathcal{A}_{ext}$ might steal or forge a tokoin by i) tampering with the blockchain and changing an existing tokoin's owner address or holder address to its own address, or ii) conducting a replay attack to resend a previously intercepted function call message to blockchain. According to our security assumptions presented in Section 3.1, blockchain is considered as a secure distributed ledger that supports trusted storage and atomic state transitions, and is free from manipulation; therefore i) is nullified. Additionally, one can include a timestep in each signature to prevent successful replay attacks to nullify ii). For the same reason one can thwart attack (B) replaying a *Redeem* message, based on the fact that the possibility of forging a digital signature in polynomial time is negligible.

The attack (C) can be defended based on the security assumption that all programs written to the TEE secure zone is free from tampering (Section 3.1); as a result, *TACO* can securely retrieve access policies from a tokoin in blockchain, make correct access decisions according to the access conditions, and monitor the access procedures as we expected. The Cortex-M series microcontroller we employ in this study is also one of the very few TEE choices not yet been found vulnerable in real life. Any shallow-level attack in the non-secure zone will not work since the encrypting and signing of data are done in the secure zone and an end-to-end security channel is established between TEE secure zone and blockchain.

The overprivileged access can be defended as follows. An internal adversary $\mathcal{A}_{int}$ may want to penetrate TEE secure zone to fake the data, change the access decisions, or delete the proofs (particularly those proofs showing that $\mathcal{A}_{int}$ violates the access procedural restraints). With the 4W1H fine-grained access policy clearly defining the conditions under which the target resource can be accessed and the correct procedure to use it, and that *TACO* can faithfully execute the pre-access decision-making and during-access monitoring processes, we are confident to claim that TBAC can prohibit overprivileged accesses by granting only the exact amount of resources and privileges to the subject and keeping all access behaviors fully accountable on blockchain.

4 TBAC System Implementation

In this section, we detail our implementation of the TBAC system. First we present the primitive elements such as tokoin structure, function-calling message structure and a few cryptographic building blocks. Then we describe the implementations of all TBAC functions. Finally we introduce our TBAC prototype implementation, including the blockchain ledger for tokoin manipulation, *TACO*, and an Android TBAC App (TAP) for the ease of use.

4.1 Primitive Elements and Building Blocks

The basic structure of a tokoin $t = (t_{ID}, pk_O, pk_H, policy, isValid)$, where pk_O and pk_H are respectively the public keys of the owner and current holder of t, t_{ID} is a number uniquely identifying t among all the tokoins generated by the owner pk_O, *policy* defines *who is allowed to do what by how in where at when*, and *isValid* is a binary indicator with $isValid = 1$ if and only if t is still valid (not redeemed and not revoked). Note that a tokoin t is uniquely identified in TBAC by the two-tuple pk_O and t_{ID}, denoted by $pk_O||t_{ID}$, as two tokoins generated by different participants may have the same t_{ID}. For conciseness, *isValid* is omitted and t is used instead of $pk_O||t_{ID}$ to identify a tokoin, if clear from context.

A tokoin is stored on-chain and managed by functions Create, Transfer, Modify, Revoke, and Redeem. Each function is called by a message signed with the function caller's secret key sk. All blockchain nodes receiving the message must first verify its authenticity and the caller role based on the carried signature before triggering any activity defined by the function. In TBAC, a message from a caller with public key pk has the following format:

$$msg : (t, f_{id}, [policy], [isValid], [pk'])_{\sigma_{sk}} \quad (1)$$

where f_{id} is the handler of the function to be called, i.e., $f_{id} \in \{Create, Transfer, Modify, Revoke, Redeem\}$, σ_{sk} is the message signature signed by the secret key of the function caller pk, and the square brackets contain optional parameters that depend on f_{id}: if $f_{id} = Create$ then *policy* and *isValid* are reqiured; if $f_{id} = Modify$, then a new *policy* is required; and if $f_{id} = Transfer$ then a new receiver's address $\{pk'\}$ is required.

In our TBAC implementation, we utilize multiple key cryptographic primitives that have been proved secure and computationally efficient. They are carefully chosen to fulfill TBAC's security requirements while ensuring not over-qualified for the tasks and not increasing unnecessary overheads.

- ***Tendermint-BFT:*** Tendermint-BFT is an improved BFT consensus algorithm over Practical BFT. It assigns different weights to different validator nodes, thus can effectively defend against Sybil attacks [32].
- ***Digital Signature:*** We adopt ECDSA because it has a shorter 256-bit key and slightly lighter computation burden compared to RSA, thus especially suitable for embedded systems.
- ***Cryptographic Accumulator:*** A cryptographic accumulator describes a set of public keys with a short, verifiable signature. Given a public key, it can efficiently verify whether or not this key is a member of the group. We use it to define a group of legitimate subjects for flexibility.

A full graphical illustration of our TBAC protocol is presented in Fig. 4, which shows a chronological sequence of all the functions affecting a tokoin throughout its lifecycle. More specifically, a blue vertical rectangle represents a tokoin while the blue vertical line it resides in indicates

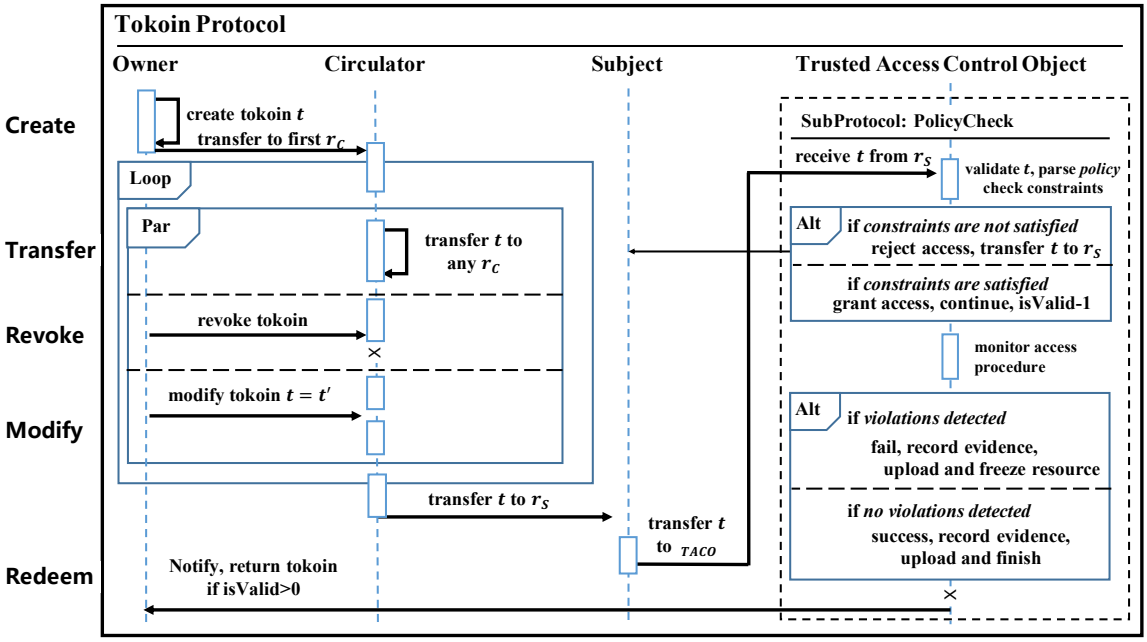


Fig. 4: The Full TBAC Protocol Sequence Diagram

who possesses the tokoin; and a small cross says that the tokoin becomes invalid ($t.isValid = 0$). We also have a few keywords in Fig. 4, with Loop meaning that all functions within the box can be repeatedly called, Par signaling that any of the operations in the small box can be executed, and Alt indicating an if-else branch. We assume the key generation Gen is done by all participants before the whole process starts and a PKI is ready to use.

4.2 Implementation of the Authentication, Authorization, and Auditing Processes

4.2.1 Authentication

The Authentication process verifies 1) the authenticity of a tokoin, 2) the caller's role of a tokoin, and 3) whether the access policy is strictly followed. In service to the Authorization process, the Authentication process in TBAC makes use of functions Verify() and PolicyCheck().

Implementation of Verify()**.** The function Verify() first authenticates message integrity by verifying signature σ with pk; if passed, it then verifies the function caller pk's role with respect to t. There are two different cases:

- $role = r_S$, where Verify() checks pk against the cryptographic accumulator acc stored in the access policy and it returns 1 if and only if pk is a legitimate subject for t.
- $role = r_O/r_H$, where the caller's public key pk is equal to $t.pk_O$ or $t.pk_H$, meaning that the caller is the resource owner or the current holder of t.

Implementation of PolicyCheck() **and *TACO*.** The implementation of PolicyCheck() requires a secure and trusted access control object *TACO* that can securely and correctly 1) communicate with the blockchain network to fetch access policy, 2) sample the current physical environment and make access decisions, 3) verify whether the access procedure is strictly followed after access is granted, and 4) upload the access results back to blockchain. In TBAC, we utilize TEE to implement *TACO* as it provides a trusted execution environment that guarantees security-sensitive programs to be tamper-proof and correctly executed.

To achieve goal 1 and 4, we implement a blockchain client inside the TEE secure zone to ensure that it communicates directly to our blockchain network via an authenticated SSL/TLS channel. Data encryption and signature are performed in TEE secure zone to realize an end-to-end security. Such design allows *TACO* to work independently as a light-weight blockchain node that does not maintain a full ledger but can listen to the network layer messages without relying on any unreliable proxy.

To achieve goal 2, we wire-connect the monitoring sensors (such as GPS receiver or security camera) to the TEE secure zone, and develop the sensor drivers in TEE. This can make sure that the sensors collect correct and authentic data without risking data interception or alteration. By comparing the sensed data against the access policy fetched from the redeemed tokoin, *TACO* can make correct access decisions for the redeemer.

To achieve goal 3, we need to implement application-specific algorithms to check whether any of the perspective procedural constraints (See Fig. 2 for an example) defined in the access policy is violated, and if Yes, appropriate actions are taken and a proof of violation is sent to Blockchain for auditing purpose. See Section 5 for a detailed implementation of how access procedure control is realized in our case study.

4.2.2 Authorization

An Authorization process determines whether a person or a process is authorized to perform an access activity. In TBAC, this process can grant a capability (tokoin) to access the resource, modify the capability, or redeem the capability under correct access policy. It includes the following five functions, whose implementations are detailed in sequel:

1) Create(): create a new tokoin;
2) Modify(): modify the access policy of an existing tokoin;
3) Transfer(): transfer a tokoin to another participant;
4) Revoke(): revoke a tokoin;
5) Redeem(): take in a tokoin and redeem the resource.

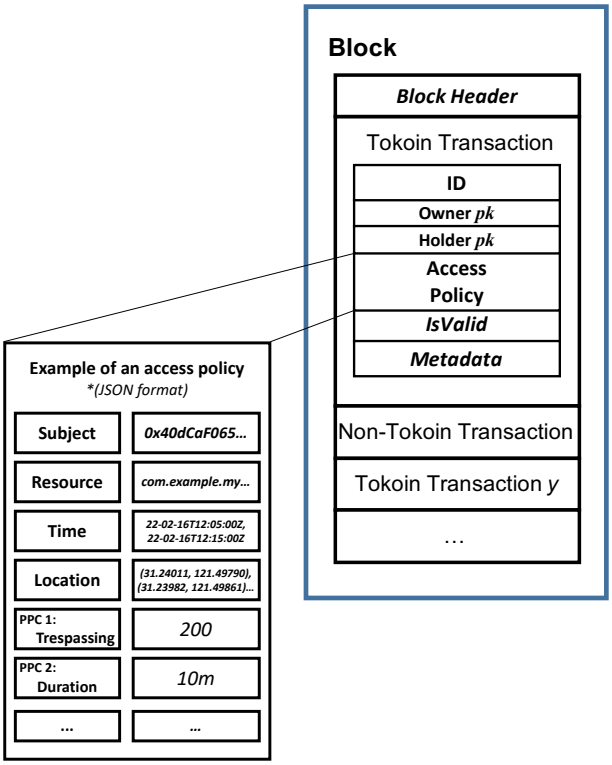


Fig. 5: Block Structure and URPO

The implementations of these five functions require explanation of the storage of tokoins. A tokoin should be stored securely and can be transferred atomically. To realize this objective, we develop the UnRedeemed Policy Output (URPO) model, which is similar to Bitcoin's UTXO model, to manage the tokoins in the distributed ledger. Any operation over a tokoin, including the creation of the tokoin, starts from a message sent to all blockchain miner nodes for verification and consensus approval. If successful, the process takes in an existing tokoin if available, performs the operations as requested, and stores the processed tokoin in the next block within the ledger. This whole process is called a *tokoin transaction*. As long as the latest tokoin transaction output has a valid $t.isValid$, the tokoin remains valid.

The main purpose of this design is to maintain high atomicity and accountability of a tokoin, which implies that only atomic and one-to-one transitions can be allowed to operate a tokoin. Each transaction must be verified by all validating nodes in the blockchain with public knowledge, and no tokoin can be forged or forfeited out of thin air. The block structure is illustrated in Fig. 5. Each block contains a number of tokoin transactions and each tokoin transaction contains the native information of the tokoin as well as a *metadata* field logging the corresponding information of history activities over this tokoin. An access policy within a tokoin is represented in the form of JSON key-value pairs for its simplicity [33], [34], as illustrated in Fig. 5. One can see that such a structure allows users to flexibly define their own access policies at different granularity levels.

Any registered participant can create a tokoin, as long as he issues access tokoins only to his own resource. To make a tokoin Create() call, the participant sends out a message $msg : [Create, \_policy, \_isValid]_{\sigma_{sk}}$ signed with his secret key sk. Upon correct verification of the signature, Create() creates a new tokoin $t = (t_{ID}, pk_O = pk, pk_H = pk, policy{=}\_policy, isValid{=}\_isValid)$ and returns t_{ID}.

The implementations of Modify() and Transfer() are similar to that of Create(), in that they all require an authenticated request message carrying the corresponding function handler f_{id} sent to the blockchain system. But there are subtleties that differ them significantly: Modify() can be called only by a tokoin owner, and it changes the access policy of the tokoin; Transfer(), on the other hand, can also be called by the current holder of a tokoin, and it carries the public key pk'_H of the next holder thus changing the current holder of the tokoin upon completion. Accordingly, the message for Modify() has a format of $[t, Modify, policy*]_{\sigma_{sk}}$ and that for Transfer() has a format of $[t, Transfer, pk']_{\sigma_{sk}}$.

When receiving a Modify() message, the verifiers in the blockchain system first check whether $\mathsf{Verify}(t, \sigma_{sk}, Modify) = 1$, i.e., check if t is valid, has a valid signature σ_{sk} signed by pk, and $pk = t.pk_O$; if yes, the access policy of the tokoin is changed and the corresponding transaction is recorded in the next block. Note that a revision on an access policy can modify any key-value pair of the policy, and can add new or delete existing key-value pairs, to redefine the access policy. Also note that the values within a policy are all plaintext-modifiable except the access subject group, which consists of one or more individuals and is described by a cryptographic accumulator; therefore, to add or delete subject pk', $Add/Del_{ACC}(pk')$ and $Update_{ACC}()$ should be called to add or delete the subject and update the value of *Accumulator* in the access policy. Similarly, when receiving a Transfer() message, the verifiers first check whether $\mathsf{Verify}(t, \sigma_{sk}, Transfer) = 1$, i.e., check if t is valid, has a valid signature σ_{sk} signed by pk, and $pk = t.pk_O || t.pk_H$; if yes, the current holder of the tokoin $t.pk_H$ is set to pk' and the corresponding transaction is recorded in the next block.

The implementation of Revoke() is rather simple. To revoke a tokoin t, the owner of t sends out a message $msg : [t, Revoke]_{\sigma_{sk}}$ to the blockchain system, in which the verifiers first check whether $\mathsf{Verify}(t, \sigma_{sk}, Revoke) = 1$, i.e., check if t is valid, has a valid signature σ_{sk} signed by pk, and $pk = t.pk_O$; if yes, $t.isValid$ is set to 0 nullifying the tokoin t in the system.

The implementation of Redeem() is a bit complicated. Upon receiving a Redeem() message $msg : [t, Redeem]_{\sigma_{sk}}$ from a holder, the verifiers need to check whether $\mathsf{Verify}(t, \sigma_{sk}, Redeem) = 1$, i.e., check if t is valid, has a valid signature σ_{sk} signed by pk, andpk is a member of the cryptographic accumulator $ACC()$; if yes, t is transferred to *TACO*, who then calls PolicyCheck() to redeem the requested resource. If PolicyCheck() successfully returns, which means that the access process is successful, *TACO* sends a confirmation message to the tokoin owner r_O. All activities in the redeem process, including those from PolicyCheck(), are recorded in the *metadata* field of the tokoin transaction for Redeem().

4.2.3 Auditing

Auditability and traceability are native in TBAC, as all operations over a tokoin and all resource access activities are logged within the *metadata* field of a tokoin transaction stored in the blockchain. Such auditing evidence is publicized and verified by the whole blockchain system, and as a result, it is globally legitimate. Under the security assumption that the blockchain system is free from manipulation and the consensus process is not compromised, the auditability of tokoin management and access control activities can be securely guaranteed.

4.3 TBAC Prototype Implementations

Our TBAC system consists of three components: 1) a blockchain-based distributed ledger that securely manages the tokoin access capabilities, supports secure atomic tokoin operations in the form of transactions, and logs all activities with security significance for auditing purpose, 2) a trusted access control object TACO within a TEE chipset that hosts the programs of embedded blockchain clients and sensor drivers in its secure zone for collecting trusted environment evidence upon redemption and making correct, attack-free access control decisions, and 3) an App-based user interface for a good user experience.

We have two implementations of the first component: a native implementation *Go-Tokoin* in Go language (Golang) that follows our original design for the best performance, and an Ethereum based implementation *Ethereum-Tokoin* in Solidity that shows adaptivity of TBAC to mainstream blockchain platforms. We run Ethereum-Tokoin in both Ethereum Mainnet and Quorum, a consortium fork version of Ethereum that uses Raft or Istanbul-BFT consensus[2]. The tokoin functions are tested on the Native Go-Tokoin and the adaptive Ethereum-Tokoin. The experiments are run with seven virtual nodes on top of a PC with the following setup: 8-Core Intel i7-6700HQ @ 2.6GHz, 16G memory, Ubuntu 18.04.1 GNU/Linux. We record and analyze the performance of the implementations for the same case study reported in Section5. One can preview the results in Fig. 10, which indicate that our native Go-Tokoin takes typically 40-60 millisecond to confirm a transaction, while the Ethereum-Tokoin in Quorum consortium chain takes about 1 second (more than a magnitude) and that in Mainnet takes about 30 to 50 seconds (one more magnitude than that).

4.3.1 Go-Tokoin: Native Golang Implementation

We implement the main blockchain system of TBAC with over 3618 lines of code in Golang, including the Tendermint-BFT consensus and the intercommunication protocols between a participant and the blockchain ledger. Golang is selected because it is a memory-safe, high-concurrent, high-usable language that is quite popular in the security community. As we build our native system pretty much from scratch, we concentrate on flexibility and customized optimization while strictly following the detailed construction presented in Section 4.1. A complete working system is available in Github at https://github.com/zhuaiballl/Go-Tokoin.

2. https://github.com/jpmorganchase/quorum

4.3.2 Ethereum-Tokoin: Adaptive to Mainstream Platforms

Although our Go-Tokoin native implementation has better performance and more design flexibility as demonstrated by our case study in Section 5, we still want TBAC to be readily available in other mainstream blockchain platforms. Thus we implement all required TBAC functions in Ethereum Solidity, in the form of a smart contract. As we manage tokoins as digital assets, we develop the interface on top of ERC-721 that is often used to represent unique digital assets or collectibles, and can be tracked individually. Then, we develop our own TBAC Smart Contract on top of the ERC-721 interface, implementing the aforementioned TBAC-specific functions, Ethereum events, and data members. Each user can mint a tokoin by deploying a tokoin smart contract. Users can directly implement their tokoin contracts with Remix-Ethereum IDE, or simply use the TBAC mobile App (presented in the following subsection) to auto-generate one. We keep this open-source on the Github at https://github.com/DES-PER-ADO/Ethereum-Tokoin.

4.3.3 The Trusted Access Control Object TACO

The functionalities and implementations of *TACO* are explained earlier so here we discuss more on our hardware choices. The most popular TEE choices include Intel Software Guard Extensions (SGX), ARM TrustZone, and AMD Secure Execution Environment. In our implementation, we adopt the LPC55S69-EVK microcontroller that uses ARM Cortex-M33 chip on ARMv8-M TrustZone architecture for the following reasons: 1) more rigorous security as its secure zone and non-secure zone are precisely partitioned, 2) lower power consumption, and 3) lower monetary cost ($20 for Cortex-M33) compared to Cortex-A (~$200) and Intel SGX (~$400) chips. The cost we have to pay is the availability of very little library or SDK, thus a majority of our codes are written by us in C and Assembly. Compared to the rich libraries of Inter SGX and Cortex-A with developer-friendly IDEs, we build the TEE system almost directly on a baremetal MCU. We write about 3691 lines of code and our executable binaries burnt into the MCU is about 2790 KB.

Note that the code for our trusted access control object TACO is application-specific as the dynamic procedural constraints are different. The one for the in-home cargo delivery case study is available at https://github.com/DES-PER-ADO/TACO.

4.3.4 TAP: the TBAC App

To avoid users from being messed up with different programming languages, and more importantly, to provide a better user experience with no code exposure, we develop an easy-to-use TBAC mobile APP in Android, namely TAP, that integrates a TBAC interface with a script/-contract wrapper. The purpose of TAP is to keep users from being exposed to Solidity, Javascript, Golang, or C code. Similarly, the code of TAP is application-specific and the one for our in-home cargo delivery case study is available at https://github.com/DES-PER-ADO/TAP-Cargo-Delivery and this part takes 4711 lines of code.

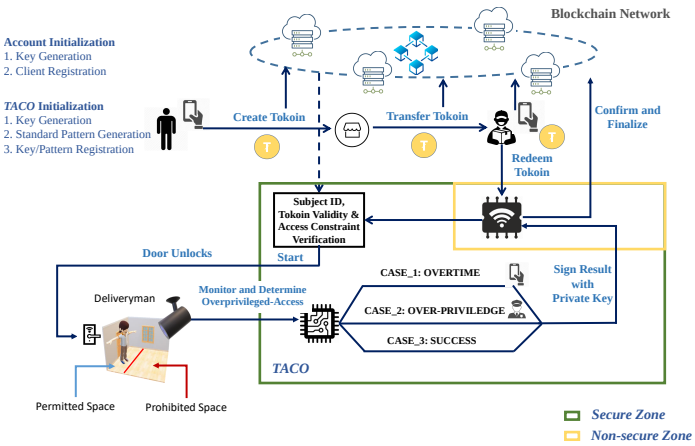


Fig. 6: TBAC Assisted In-home Cargo Delivery

5 Case Study: TBAC Assisted In-home Cargo Delivery

In this section, we report a case study that employs TBAC to assist in-home cargo delivery. This case study demonstrates how TBAC and IoT can seamlessly work together to control a smart lock in a secure, fine-grained, and accountable manner for safe in-home delivery.

In the US, online-purchased merchandises are usually delivered to the outside doorstep of a house, thereby risk of being stolen. With the help of the smart door lock, a deliveryman can open the house door and leave the cargo inside. This may seem to be a good solution and in fact it has been adopted by Amazon [35]. Nevertheless, by signing up for this in-home delivery service, users would surrender to the unlimited, unconditional, and unauditable accesses to their homes as an unlimited access token is issued from the door lock manufacturer's access server to Amazon after authorizing Amazon the access privilege to the door. It may get worse if the Amazon's server is compromised or the token is stolen or abused. In this section, we show that with TBAC, one can have high confidence that only the customer-approved accesses can take place, with a complete auditing. We also emphasize that with a fine-grained access policy specified by the customer, a robust access control object can monitor the delivery procedure to ensure that the deliveryman does not intrude the house by doing more than dropping the package.

Fig. 6 demonstrates our TBAC assisted in-home cargo delivery case study. When an order is placed, a tokoin specifying the detailed access policy to the customer's house is also created. The order and the tokoin are sent together to the seller, who then transfers the tokoin to the first courier of the package when it leaves the warehouse. In transit the tokoin changes its holder when the package is handed to a different courier. The customer can monitor this process via TAP and can change the access policy based on the shipment status. When a courier arrives at the doorstep of the destination house, the tokoin is redeemed and the package is dropped inside home if the access policy verification succeeds. For security and safety, the access policy specifies that the deliveryman cannot walk out of the mud area to enter the main house. Thus a video camera is adopted to monitor the procedure and a violation is reported immediately when detected. In the following we present this case study in two processes: an initialization process and a delivery process.

5.1 Initialization

The initialization process consists of customer account initialization and TACO initialization. Note that we assume that the seller and its couriers are TBAC clients thus no extra work is needed here.

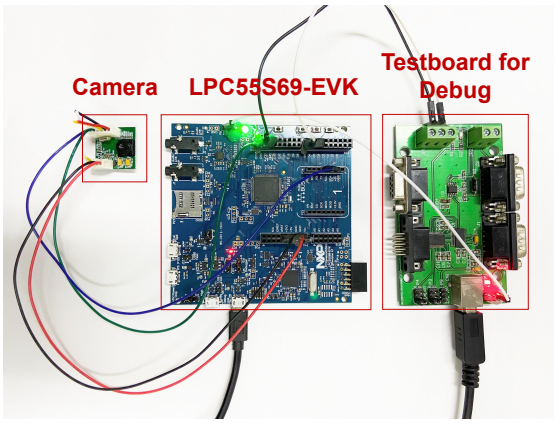


Fig. 7: Experiment Setup of TEE

Account Initialization. Upon initialization, a customer calls the Gen function to generate a pair of keys sk and pk, registers itself with and broadcasts its public key pk to the blockchain network, and keeps the private key sk to itself. The customer downloads the TAP software and becomes a TBAC participant who can talk with the blockchain securely.

TACO* System Configuration.** To establish the *TACO* for our in-home cargo delivery case study, we adopt an LPC55S69-EVK microcontroller secured by an ARMv8-M TrustZone, a smart lock, a GPS receiver, and a low voltage UART serial image sensor camera. In addition, a testboard is utilized for debugging purpose. Upon initialization, *TACO* generates a secret key sk and a paired public key pk using crypto-libraries in LPC55S69-EVK. Then it broadcasts its pk to the blockchain for self-registration. The pk is used as the address and the unique identifier of the *TACO* system in the blockchain. *TACO* also uses the camera to capture the ***normal background of the home mud area when nobody shows up, and stores it as a `STANDARD PATTERN` into its secure zone for future detection of over-privileged behaviors such as the deliveryman walking out of the mud area to enter the main room. The pk and the `STANDARD PATTERN` are then registered as a tokoin transaction in the blockchain.

Note that we have implemented the drivers of the GPS receiver and the UART serial camera in the secure zone, which directly connects to the corresponding devices for secure data collection. Also in our case study *TACO* is able to command the smart lock via secure communications with the lock server, who can securely talk with the smart lock, thus avoiding the hassle of writing a driver in the secure zone and connecting it to the lock. Fig. 7 illustrates the TEE setup with the sensor camera – the GPS receiver is omitted for better illustration.

5.2 Delivery Process

The delivery process is depicted in Fig. 6, whose steps are detailed as follows.

Create a tokoin. Before an order is placed, the customer mints a tokoin by sending $msg : [pk||t_{ID}, create, policy]_{\sigma_{rO}}$ to the blockchain, who then creates a tokoin t and logs it in a transaction. The *policy* specifies *who* (any legitimate deliveryman) is allowed to do *what* (cargo delivery) in *where* (address of the house) at *when* (e.g., 2-3PM tomorrow) by *how* (enter the house, drop the package in the mud area, then leave the house in 5 minutes). See Fig. 8(a) and Fig. 8(b) for an illustration. The tokoin t and the order together are then sent to the seller.

Transfer the tokoin. The seller then assigns a courier to this order, and transfers t to the courier via a Transfer message when the package is handed to the courier. As we have discussed before, t can be freely transferred among the couriers through a standard transfer operation, which eases the re-distribution of the delivery job for better logistic convenience. Besides, the customer has the right to track the tokoin through a tokoin map as shown in Fig. 8(c). It can also modify the tokoin (e.g., changing the delivery time window) during this process at its will before the tokoin is redeemed.

Redeem the tokoin. When a deliveryman arrives at the house address and wishes to redeem the tokoin t to complete the job, Redeem is called, which sends the tokoin t to *TACO*, who would perform the following three tasks for tokoin redemption.

1. Access Condition Verification. After receiving t, *TACO* needs to Verify: 1) whether t is a valid tokoin; 2) whether the deliveryman is a legitimate subject according to the cryptographic accumulator carried by *policy*; and 3) whether the spatio-temporal access conditions are met, i.e., the time of delivery and the delivery address are all consistent with what are specified by *policy*. If all verifications succeed, *TACO* instructs the smart door lock to open the door and let the deliveryman in to drop the package.

2. Access Procedure Monitoring. After entering the house, the deliveryman should drop the package in the mud area and leave in time specified by *policy*. To monitor this process, *TACO* constantly reads inputs from the UART serial camera and checks the position of the deliveryman by detecting moving objects in the video. Specifically, *TACO* computes the difference between the `STANDARD PATTERN` and every video frame, and adds them up as a differential monitoring pattern, which is obviously a bitmap with boolean 1 for presence and 0 for absence of the deliveryman. The determination of a violation, i.e., an overpriviledged access, is detected if there is a boolean 1 out of the mud area, see the illustration in Fig. 9, which uses an imaginary red line to represent the boundary of the mud area.

There are two possible types of violations:

- Case 1: the deliveryman stays longer than the time specified by *policy*. In this case, *TACO* would first ring an alarm bell, then send a signed `OVERTIME` message to the customer. If the deliveryman does not leave immediately, *TACO* may call the police.
- Case 2: the deliveryman walks out of the permitted mud area to enter the main room. If this case is detected, as shown Fig. 9(c)(d), the corresponding pattern of motion trajectory is recorded as a proof of an over-privileged behavior; then *TACO* sends a signed `OVER-PRIVILEGED PATTERN` to the blockchain and takes appropriate measures such as locking the smart door and calling the police.

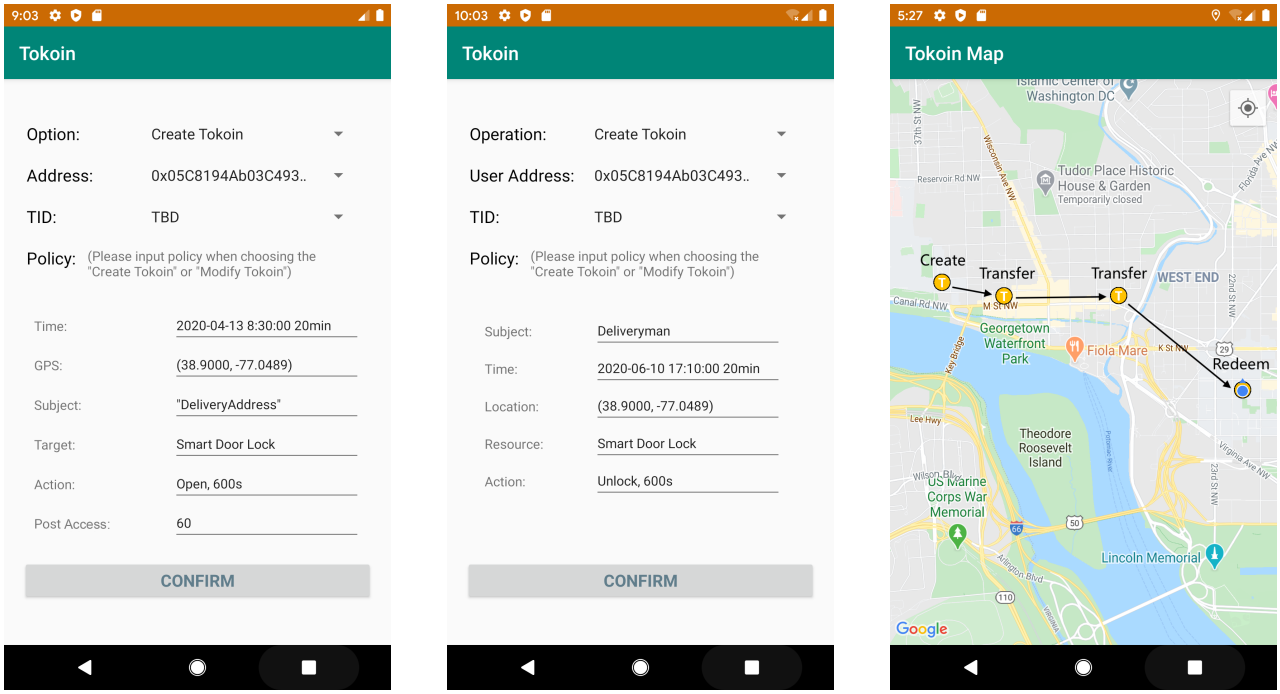
3. Post-Access Management. If no violation is detected, *TACO* sends a signed `SUCCESS` to the blockchain after the deliveryman successfully drops the package in the mud area and leaves the house. Note that all the data for access condition verification and access procedure monitoring must be signed with the private key of *TACO* and stored within the TEE secure zone. This can guarantee the integrity and the trustworthiness of the data. The data itself or a digest of the data (if the data is too big) is also sent to the blockchain to be included in a tokoin transaction as a script for auditing the Redeem operation.

5.3 Performance Evaluation

The execution time for each tokoin management function in our case study is reported in Fig. 10. This figure shows the performance of different TBAC implementations, in logarithmic scale. The reason why we choose log scale is because the confirmation times of Go-Tokoin and Ethereum-Tokoin in three different platforms vary up to two magnitudes. Our native Go-Tokoin takes typically 40-60 milliseconds to confirm each transaction, while the Ethereum-Tokoin in Quorum consortium chain takes about 1 second (more than a magnitude) and that in Mainnet takes about 30 to 50 seconds (one more magnitude than that). One should notice that Go-Tokoin takes much longer time in Redeem, which includes the time for PolicyCheck. This is because Redeem requires *TACO* to sample the sensor readings, analyze the data, take corresponding actions if needed, and finally communicate the data back to blockchain. Note that we actually transmit only the digest of the video data in our case study, thus the packet size is small, which is less than 0.2 MB. Also note that the total size of the executable binaries (including sensor drivers and processing algorithms) compiled and stored in the MCU is about 2790 KB, which does not cost too much memory. Besides, the transaction cost on the Ethereum Mainnet is about 2967k Gas on average ($14.7 USD, in July 16, 2019), while our consortium Go-Tokoin is free.

6 Conclusions and Discussions

In this paper we propose TBAC, an accountable IoT access control model that makes use of blockchain and TEE technologies to realize its goal of offering fine-grained access (procedure) control with strong auditability. We design and implement three components: 1) a blockchain system to manage tokoin and its access policy, 2) a trusted access control object (*TACO*) to sense the physical world, make actual access decisions, and monitor the access procedure, and 3) an App-based user interface. We also present a TBAC-assisted in-home cargo delivery case study to illustrate how TBAC is employed to secure the procedure for a deliveryman to open a door and drop the package in the mud area of a room without entering the main room, demonstrating the efficient performance and effective control over the deliveryman's access behavior by TBAC.



(a) Creating tokoin (b) Detailed information of a tokoin (c) Geographic trace of a tokoin

Fig. 8: Android Tokoin Activities for Cargo Delivery

(a) An actual frame of a *TACO* camera capture (b) A *TACO* captured pattern of benign behavior (c) A *TACO* captured pattern of minor violation behavior (d) A *TACO* captured pattern of a major violation behavior

Fig. 9: A Set of Actual Captures of *TACO* for Overprivileged-Access Monitoring

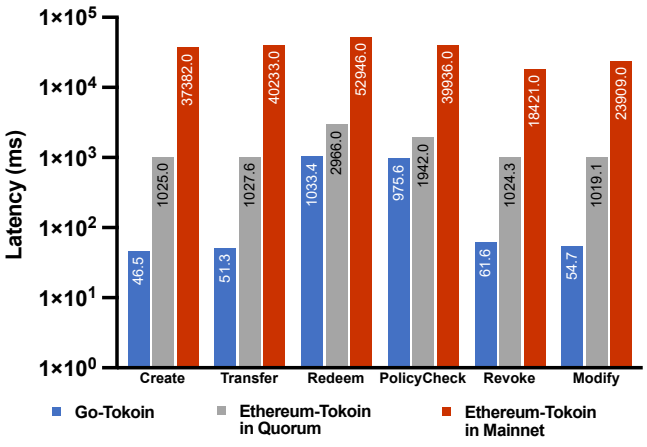


Fig. 10: Function Execution Time

Our current TBAC implementation as well as the case study focus on the application scenario where a resource owner would like to grant a one-time access to his resource to one or more designated subjects when certain conditions ('when' and 'where') should be met and certain procedures ('how')) need to be followed. In fact, TBAC can be extended along the following lines, which mainly bring engineering issues that do not affect the security of the TBAC model.

- It is trivial to allow a tokoin to be reused/redeemed, by adding a perspective procedural constraint (PPC) "times" in the access policy specifying the number of times the tokoin is allowed to be used. To support this extension, the tokoin needs to be modified (i.e., decreasing "times" by 1) and sent back to the blockchain after a successful redemption, which could be done securely by either TACO in TEE or the resource owner, with necessary changes in the Modify and Redeem functions. This extension allows the owner to give a subject the right to access the resource multiple times; when combining with other constraints, granting the access on a daily basis is also possible.
- Our current implementation of TBAC gives multiple

subjects the opportunity to access the resource with the same tokoin via a cryptographic accumulator but only one subject is allowed to finally redeem the tokoin. By combining with the first extension, the same tokoin can be used by multiple subjects at different times; when multiple subjects access the same resource simultaneously with different tokoins, access conflict may occur and the function of TACO needs to be extended to handle the races and resolve conflicts. When the resource has multiple owners and one would like to allow all owners to have the same modify/revoke rights over a tokoin created by one owner, a cryptographic accumulator can be adopted to designate the group of legitimate owners of the tokoin (replacing the pk_O field of a tokoin with a cryptographic accumulator).

- The current description of the *4W1H* policy stresses sampling physical world to collect the information ('when' and 'where') for the pre-access condition verification to grant access. In fact, it is not hard to support decision-making based on information from the digital world, e.g., a username and password pair or a one-time password, by adding the corresponding information in the access policy, e.g., adding a password line as a static condition in the access policy, as long as the functionality of TACO is correspondingly expanded. This extension is particularly useful for applications such as errand delegation, where an account owner needs to delegate someone to process certain urgent business on his behalf without disclosing his own sensitive account information.

These extensions make applications of TBAC very versatile. For examples, by developing appropriate TACO functionality, TBAC can support situational-aware access control in which the status of the physical environment (or an abstract *situation*) needs to be monitored (see [20]), and can resolve races among different IoT devices by adopting the approach presented in [24] (TACO takes the responsibility of the backend server and policy manager in [24]). Nevertheless, TBAC is not a "master key" to solve all access control problems; it has its own limitations and restrictions. First, TBAC needs the support of blockchain and TEE technologies, which bring extra cost and may not be available; second, TBAC requires owners to define access policy, which places extra burden on them, and offers fine-grained access control with strong auditability, which may imply excessive control for ordinary users in daily life; both concerns may hurt users' experiences making them completely turn off the service. Therefore, one should consider the tradeoff among cost, application requirements, and user experiences when making the decision on whether or not to adopt TBAC. Note that there exist many well-studied access control mechanisms that perfectly fit their particular application needs. In fact, TBAC is orthogonal to them, and can be adopted as a supplement to enhance their security when needed.

Our future research will be carried out along the following three directions. First, we will develop a more complicated case study to further investigate the strength of TBAC in fine-grained access control with strong auditibility. We target a multi-user smart home IoT system where an IoT device has multiple owners/users that may control it with conflict commands and that may access it on a daily basis. We will implement the extensions mentioned above to support such a scenario. Second, we intend to explore the applicability of TBAC beyond access control. We will consider the heavy machinery rental business in which a number of IoT devices are deployed within each rental machine to monitor how the machine is used from all angles (when and where is used by who and how). Such an application may also involve loans and payments, bringing interesting research challenges when considering the seamless integration of device use control and the related loans/payments via blockchain. Third, we will develop TBAC-enabled apps that can allow a staff to reply a particular email on behalf of its manager or enable a friend to withdraw certain amount of money via an ATM machine from a bank account without disclosing any sensitive account information to the staff or the friend. Such apps are good research projects that involve challenges requiring different tradeoffs and optimizations.

ACKNOWLEDGMENTS

This research was partially supported by the National Key R&D Program of China under grant 2019YFB2102600, the National Natural Science Foundation of China (No.62232010, 62302266, 62002067), the Blockchain Core Technology Strategic Research Program of Ministry of Education of China, Shandong Science Fund for Key Fundamental Research Project (ZR2022ZD02), Shandong Science Fund for Excellent Young Scholars (No.2023HWYQ-008), and the Fundamental Research Funds for the Central Universities.

7 AVAILABILITY

Open source codes of Go-Tokoin, Ethereum-Tokoin, TEE-enabled Access Control Object (TACO) and Tokoin App (TAP) are available at:

1) https://github.com/zhuaiballl/Go-Tokoin
2) https://github.com/DES-PER-ADO/Ethereum-Tokoin
3) https://github.com/DES-PER-ADO/TACO
4) https://github.com/DES-PER-ADO/TAP-Cargo-Delivery

REFERENCES

[1] CNN Tech. Google admits its new smart speaker was eavesdropping on users. Available at https://money.cnn.com/2017/10/11/technology/google-home-mini-security-flaw/index.html, October 12 2017.

[2] The Washington Post. Alexa has been eavesdropping on you this whole time. Available at https://www.washingtonpost.com/technology/2019/05/06/alexa-has-been-eavesdropping-you-this-whole-time/, May 6 2019.

[3] Deepak Kumar, Riccardo Paccagnella, Paul Murley, Eric Hennenfent, Joshua Mason, Adam Bates, and Michael Bailey. Skill squatting attacks on amazon alexa. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 33–47, 2018.

[4] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging smart home applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 636–654. IEEE, 2016.

[5] Statista. Smart home report on united states. Available at https://www.statista.com/outlook/279/109/smart-home/united-states, October 2020.

[6] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638, 2011.

[7] Yinhao Xiao, Yizhen Jia, Chunchi Liu, Xiuzhen Cheng, Jiguo Yu, and Weifeng Lv. Edge computing security: State of the art and challenges. *Proceedings of the IEEE*, 107(8):1608–1631, June 2019.

[8] N. Zhang, X. Mi, X. Feng, X. Wang, Y. Tian, and F. Qian. Dangerous skills: Understanding and mitigating security risks of voice-controlled third-party functions on virtual personal assistant systems. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1381–1396, May 2019.

[9] Yizhen Jia, Yinhao Xiao, Jiguo Yu, Xiuzhen Cheng, Zhenkai Liang, and Zhiguo Wan. A novel graph-based mechanism for identifying traffic vulnerabilities in smart home iot. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1493–1501. IEEE, 2018.

[10] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. Iotguard: Dynamic enforcement of security and safety policy in commodity iot. In *NDSS*, 2019.

[11] Yuanyu Zhang, Shoji Kasahara, Yulong Shen, Xiaohong Jiang, and Jianxiong Wan. Smart contract-based access control for the internet of things. *IEEE Internet of Things Journal*, 6(2):1594–1605, 2018.

[12] Ronghua Xu, Yu Chen, Erik Blasch, and Genshe Chen. Blendcac: A blockchain-enabled decentralized capability-based access control for iots. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1027–1034. IEEE, 2018.

[13] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. Blockchain based access control services. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1379–1386. IEEE, 2018.

[14] Shuang Sun, Rong Du, Shudong Chen, and Weiwei Li. Blockchain-based iot access control system: towards security, lightweight, and cross-domain. *IEEE Access*, 9:36868–36878, 2021.

[15] Sergio Gusmeroli, Salvatore Piccione, and Domenico Rotondi. A capability-based security approach to manage access control in the internet of things. *Mathematical and Computer Modelling*, 58(5-6):1189–1205, 2013.

[16] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague. Smartauth: User-centered authorization for the internet of things. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 361–378, 2017.

[17] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, Atul Prakash, and SJ Unviersity. Contexlot: Towards providing contextual integrity to appified iot platforms. In *NDSS*, volume 2, pages 2–2, 2017.

[18] Moosa Yahyazadeh, Proyash Podder, Endadul Hoque, and Omar Chowdhury. Expat: Expectation-based policy analysis and enforcement for appified smart-home platforms. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, pages 61–72, 2019.

[19] Smriti Bhatt, Thanh Kim Pham, Maanak Gupta, James Benson, Jaehong Park, and Ravi Sandhu. Attribute-based access control for aws internet of things and secure industries of the future. *IEEE Access*, 9:107200–107223, 2021.

[20] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Situational access control in the internet of things. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1056–1073, 2018.

[21] Maanak Gupta and Ravi Sandhu. Towards activity-centric access control for smart collaborative ecosystems. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, pages 155–164, 2021.

[22] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. Rethinking access control and authentication for the home internet of things (iot). In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 255–272, 2018.

[23] Eric Zeng and Franziska Roesner. Understanding and improving security and privacy in multi-user smart homes: a design exploration and in-home user study. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 159–176, 2019.

[24] Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A Selcuk Uluagac. Kratos: Multi-user multi-device-aware access control system for the smart home. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 1–12, 2020.

[25] José L Hernández-Ramos, Antonio J Jara, Leandro Marin, and Antonio F Skarmeta. Distributed capability-based access control for the internet of things. *Journal of Internet Services and Information Security (JISIS)*, 3(3/4):1–16, 2013.

[26] Shantanu Pal, Michael Hitchens, Vijay Varadharajan, and Tahiry Rabehaja. Policy-based access control for constrained healthcare resources in the context of the internet of things. *Journal of Network and Computer Applications*, 139:57–74, 2019.

[27] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. Understanding linux malware. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 161–175. IEEE, 2018.

[28] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1093–1110, 2017.

[29] F5Labs. The hunt for iot: Multi-purpose attack thingbots threaten internet stability and human life. Available at https://www.f5.com/labs/articles/threat-intelligence/the-hunt-for-iot--multi-purpose-attack-thingbots-threaten-intern, Oct 24 2018.

[30] Earlence Fernandes, Amir Rahmati, Jaeyeon Jung, and Atul Prakash. Decentralized action integrity for trigger-action iot platforms. In *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.

[31] Ravi Sandhu and Pierangela Samarati. Authentication, access control, and audit. *ACM Computing Surveys (CSUR)*, 28(1):241–243, 1996.

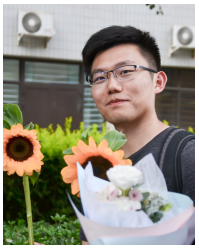
[32] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.

[33] Axiomatics. Top ten reasons why developers don't adopt abac. Available at https://www.slideshare.net/Axiomatics/axio-irm-summit-2014/17.

[34] Amazon. Grammar of the iam json policy language. Available at https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_grammar.html.

[35] Amazon. Key by amazon smart lock kit. Available at https://www.amazon.com/gp/help/customer/display.html?ie=UTF8&\nodeId=202104340.

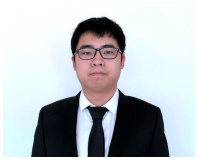
Chunchi Liu obtained his PhD degree from The George Washington University, Washington DC, USA, in 2020, and his BS degree with distinction from Beijing Normal University, Beijing, China, in 2017, both in Computer Science. His current research focuses on Blockchain, Internet of Things, Security, and Applied Cryptography.



Minghui Xu is an Assistant Professor at Shandong University who received his PhD in Computer Science from The George Washington University in 2021 and his Bachelor's degree in Physics from Beijing Normal University in 2018. His research interests include blockchain, distributed computing, and cryptography.

Hechuan Guo is a PhD student in Computer Science at Shandong University, Qingdao, China. He received his BS Degree in Computer Science in 2017 and MS degree in Engineering in 2020, both from Beijing Normal University, Beijing, China. His current research focuses on Blockchain, Consensus Protocols, Security, and Applied Cryptography.

Xiuzhen Cheng received her MS and PhD degrees in computer science from University of Minnesota – Twin Cities, in 2000 and 2002, respectively. She was a faculty member at the Department of Computer Science, The George Washington University, from 2002-2020. Currently she is a professor of computer science at Shandong University, Qingdao, China. Her research focuses on blockchain computing, IOT Security, and privacy-aware computing. She is a Fellow of IEEE.



Yinhao Xiao received his Ph.D. degree in computer science from The George Washington University, Washington, DC, USA, in 2019. He is a faculty member with the School of Information Science, Guangdong University of Finance and Economics, Guangzhou, China. His current research interests include IoT security, smartphone security, and binary security.

Dongxiao Yu received his BS degree in Mathematics in 2006 from Shandong University, and PhD degree in Computer Science in 2014 from The University of Hong Kong. He became an associate professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, in 2016. Currently he is a professor in the School of Computer Science and Technology, Shandong University. His research interests include wireless networking, distributed computing, and graph algorithms.

Bei Gong received his BS degree from Shandong University in 2005, and Ph.D. degree from Beijing University of Technology in 2012. Currently he is an associate professor in Computer Science at Beijing University of Technology. His research interests include trusted computing, Internet of things security, mobile Internet of things, mobile edge computing.

Arkady Yerukhimovich is an assistant professor at The George Washington University since Fall 2018. Prior to that, he was a research scientist in the Secure Resilient Systems and Technology Group at MIT Lincoln Laboratory where he worked on applying tools from theoretical cryptography for practical applications. He received his PhD in August 2011 under Jonathan Katz in the Computer Science department at the University of Maryland. His research aims to enable collaboration between distrusting parties.

Shengling Wang is a professor in College of Information Science and Technology, Beijing Normal University. She received her Ph.D. in 2008 from Xi'an Jiaotong University. After that, she did her postdoctoral research in the Department of Computer Science and Technology at Tsinghua University. Then she worked as a faculty member from 2010 to 2013 in the Institute of Computing Technology of the Chinese Academy of Sciences. Her research focuses on mobile/wireless networks, game theory, and crowdsourcing.

Weifeng Lv received his Ph.D. degree in computer science from Beihang University in 1998. His current research interests include massive information system, urban cognitive computing, swarm intelligence, and smart cities. He is a professor of computer science at Beihang University. He received multiple internationally renowned awards, including the second prize of the 2016 China National Science and Technology Invention Award and the first prize of the 2010 Beijing Science and Technology Award.