CapChain: A Privacy Preserving Access Control Framework Based on Blockchain for Pervasive Environments

Tam Le, Matt W. Mutka

Department of Computer Science and Engineering
Michigan State University
{letam, mutka}@cse.msu.edu

***Abstract*—Devices to support pervasive computing and the Internet of Things (IoT) are becoming present in almost every aspect of our lives. Due to their limited power and computation, these devices often need to rely on some more powerful outsourced cloud services, which raises a security and privacy concern as IoT data is often sensitive. On the other hand, blockchain technology has recently gained much attention due to its decentralized, trustless and immutable design. We propose CapChain - an access control framework based on blockchain that allows users to share and delegate their access rights easily to IoT devices in public but still maintain privacy. To protect privacy, we adapt multiple techniques from anonymous crypto-currency blockchain systems to hide sensitive information, including users' identities and related information about the capabilities. We also build a testbed as a proof of concept.**

I. Introduction

Internet of Things (IoT) devices have become pervasive in almost every aspect of our lives, from home automation, health care to industries and transportation. Due to their limited power and computation, these devices often need to rely on some more powerful outsourced cloud services [1]. Users have no choice but to entrust these third-party servers to process and store their sensitive IoT data, which raises a security and privacy concern as a latent single point of failure can be present if the servers are compromised [2]. Therefore, companies and researchers have recently sought for an alternative solution from blockchain, the core technology behind the famous cryptocurrency system Bitcoin. A blockchain is a distributed public ledger that records transactions of digital assets. It is tamper-free thanks to the work of a consensus protocol and mining process called proof-of-work that guarantees every participant in the network behaves properly. Therefore, the blockchain technology allows IoT applications to be built on a decentralized, trustless network without the need of a central authority [3].

In this paper, we focus on the access control problem in IoT applications. With the growth of IoT devices, a user can hold keys/credentials to several devices in different domains. In most cases, the distribution of those keys is carried out through a trusted server, which is either owned by the device manufacturers or a third-party service, which can be a potential central point of failure. On the other hand, it is also inconvenient for users to have a separate account to store their keys for each application they are using. Therefore, it is desirable to have a global framework that supports reliable and flexible key sharing.

With blockchain, key exchange can be carried out via transactions, thus is untampered and can avoid attacks such as replay or man-in-the-middle [4]. Since data cannot be modified or deleted once it is published, the blockchain can be treated as a reliable access control list. However, public blockchains also suffer from a privacy problem since all can access data on the blockchains, which is not a favorable situation for access control as the usage of devices should only be visible to users within their private networks.

We propose CapChain - an access control framework based on blockchain that allows users to share and delegate their access rights easily to devices in public but still maintain privacy. Our idea is to treat the access rights, which are called capabilities in CapChain, as types of assets that can be transfered between users via transactions. We assume that every IoT device in CapChain has one or several ultimate owners who have full control over the device and are able to generate capabilities based on its own access control policies. The capabilities then can be delegated to other users via transactions on a public blockchain that serves as a public immutable ledger that records the capability delegation. In order to grant access to a certain user, the device needs to verify the existence of the relevant transaction in the blockchain. To protect privacy, we apply multiple techniques to hide sensitive information that can only be visible to relevant users. Our delegation system has the following characteristics:

- Each user's capability has an expiry date for auto revocation. As a rule of delegation, a receiver's capability cannot expire later than the expiry date of the sender's capability.
- Besides auto-expired capabilities, users can still track or revoke the whole chain of delegation originated from themselves if necessary.
- Identities of sender and receiver as well as transaction information are protected.

978-1-5386-4705-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SMARTCOMP.2018.00074

- Users need only one single master account to receive capabilities from different domains.

With anonymous transactions, CapChain allows users to share their home devices with other users as well as receive keys of other places such as their offices through a global network without worrying about their private information being revealed to the public, thus enables scalability to multiple domains and organizations.

II. Related work

We review related work on access control that are based on blockchain in section A. Section B describes some projects that attack the privacy problem in blockchain systems.

A. Access control in Blockchain

In terms of access control, Enigma [5] is a data management platform based on both blockchain and off-chain storage. It tackles common privacy issues with data ownership, data transparency and auditability and fine grained access control. To achieve privacy, only pointers to user data are stored on the blockchain, while the data themselves are encrypted before being randomly spread among a network of nodes and managed by a distributed hash table.

IBM introduced a blockchain-based architecture called ADEPT (Autonomous Decentralized Peer-to-Peer Telemetry) [6], in which a dynamic democracy of objects connected to a universal digital ledger, which provides users with secure identification and authentication.

FairAccess [7] is a blockchain-based access control framework that employs a similar idea to transfer access tokens via transactions as our CapChain. However, in order for the network to validate a token, FairAccess requires the access control policy to be included in the token transaction. Since the blockchain is visible to everyone, the policy can reveal much information about the device and its users, i.e. the type of the device and the type of access. In addition, since the device itself is not capable to store the entire blockchain, it must acquire the data from a trusted node, which means there still needs some kind of centralized authority in the network. Even in ADEPT, they also only expect the power and storage capabilities of smart products to increase in the future to meet the minimum requirement for blockchain functions [8]. It means that at the moment, there has not been a practical way for most of current resource-constrained devices to interact directly with blockchain yet.

B. Privacy issues

A typical Bitcoin transaction has 2 components:

- Input: a reference to an output from a previous transaction.
- Output: an output specifies the receiver and the amount of money to be sent.

In order for a transaction to be accepted by the network, it must satisfy the following: 1) the sender is the valid owner of the inputs; 2) all inputs are referred to valid, unspent outputs; 3) all inputs must be consumed by the outputs, which means

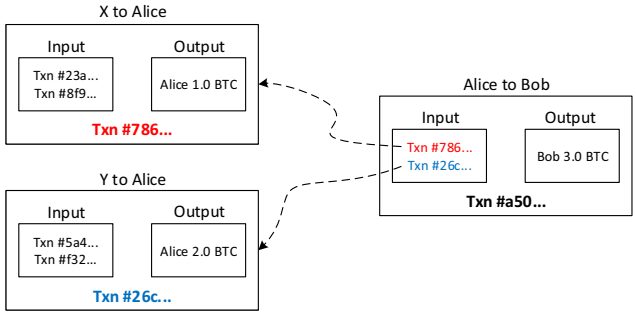


Fig. 1: Alice sends to Bob 3 Bitcoins, whose inputs refers to two previous outputs.

the total currency value from the inputs and outputs must be equal. Figure 1 depicts a sample transaction from Alice to Bob with an amount of 3.0 Bitcoins, in which the input consists of 2 previous outputs destined to Alice, a 1.0 Bitcoin and a 2.0 Bitcoin. Later if Bob wants to spend these 3 Bitcoins, he can make a transaction that refers to his output in Alice's transaction. In fact, Alice and Bob do not use their names but a pseudonym or so-called *address* that is derived from their public keys to receive bitcoins. However, if someone happens to know the user's identity associated with an address, they can link all transactions that are made from/to this user.

Recent cryptocurrency systems have been trying to protect sensitive information that is published to public blockchains. Chain [9] addresses 3 types of privacy concerns:

- Privacy of identity: the identities of sender and receiver should not be revealed.
- Privacy of amount: the amount to be transferred should not be revealed.
- Privacy of history: the inputs used in a transaction should not be traced to the previous transactions that created them as well as linked to future transactions.

For the privacy of amount, Maxwell et al. [10] developed Confidential Transactions, a protocol based on Elliptic Curve Cryptography for encrypting the input and output amounts of a transaction in a way that still allows the network to validate that the transaction balances. The protocol is adapted to other cryptocurrency systems such as Chain and Monero. Confidential Transactions is later extended to Confidential Assets [11], which does not only blind the amount but also the asset type, thus enables the transfer of multiple assets besides a single crypto-currency.

There are several projects that take various approaches to the anonymous currency, including CryptoNote [12] (and later enhanced by Monero [13]) and Zerocash [14]. The idea behind CryptoNote is a traceable ring signature [15], in which a sender mixes his input with different inputs from others and creates a ring signature that can prove that he knows the private key to one of the public keys in the ring. Monero is developed based on CryptoNote, but allows the amount of money to be

hidden as well by combining the Confidential Transactions with a ring signature called Multilayered Linkable Spontaneous Anonymous Group Signature (MLSAG) [13]. On the other hand, Zerocash is based on zk-SNARK [16], an efficient variant of a *zero-knowledge proof of knowledge*. Compared to CryptoNote, Zerocash is completely anonymous while in CryptoNote, the level of anonymity depends on the number of parties in a ring. However, the zero-knowledge proof in Zerocash is too computational expensive and is not suitable for IoT environments.

III. System overview

A. Capability

We define a capability (denoted by CAP) as a token that represents some access right to an IoT device, and is encrypted by a secret key shared between the device and its owner. For example, a user may have an "operate" capability to open/close a smart lock. Capabilities can only be generated and published to the blockchain by the device's owner, then are transferred between users via transactions.

In CapChain, the rule is transferred capabilities cannot have longer life time than the original ones. Thus, the expiration time can be treated similarly to the amount of money in cryptocurrency systems. In other words, if we ignore the context of capability, the fact that Alice, who holds a capability that will expire in 5 days, can only delegate the same capability that expires in no more than 5 days, is the same as Alice has 5 USD, and thus can only transfer 5 USD at maximum.

B. Blockchain and capability transactions

The blockchain stores all of capability transactions. It serves as an access control list that records the proof that a user is holding a certain capability and when it will be expired. There are 3 types of transactions: $tx_{publish}$, $tx_{delegate}$, and $tx_{confirm}$.

New capabilities are published to CapChain via $tx_{publish}$ with an initial expiry date set by device owners. To delegate capabilities, users need to post a $tx_{delegate}$. Just like a Bitcoin transaction, $tx_{delegate}$ also consists of input (aka the source capability) and output (aka the destination). $tx_{publish}$ can be considered as a special $tx_{delegate}$ but without the input. In a $tx_{delegate}$ transaction, senders has to specify a new expiry date of the delegated capability. As a rule of delegation, the new date cannot be later than the old one from the input capability. To prevent arbitrary delegations without actual usage of capabilities, a $tx_{delegate}$ needs to be confirmed by the device or an authority (e.g. the owner) before being used in a further delegation. The confirmation is posted via $tx_{confirm}$. Figure 4 depicts a chain of delegation, starting from Alice who is the owner of CAP1, then to Bob and Carol.

C. Authorization workflow

We address 3 entities in the network:

- IoT devices that have low computation and low storage. Depending on the communication type, the device may or may not have direct access to the Internet. In the latter case, it has to rely on a local proxy to look for transactions on the blockchain.
- Proxy: a more powerful device that acts as an actual node on CapChain.
- Mobile devices such as smartphones or laptops that act as wallets to receive and transfer capabilities.

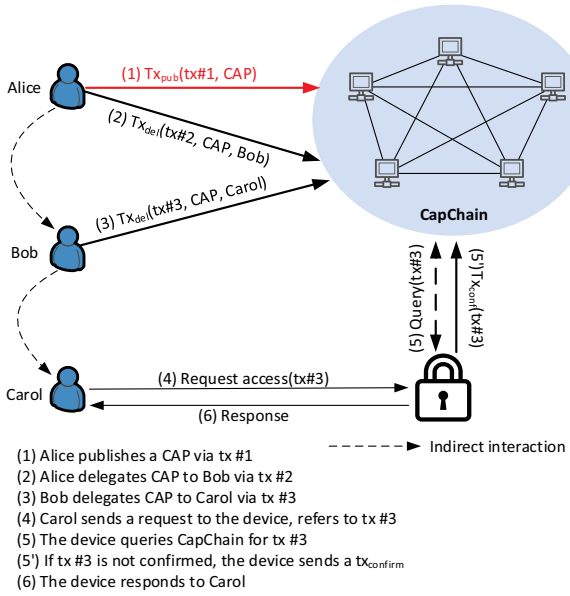


Fig. 2: System overview

To request for access, a user can prove his/her capability possession by signing the corresponding transaction. The device then will inquire to the blockchain (either directly or via local proxy) for the presence of the transaction and the capability and grant access accordingly.

D. Transaction linkability

Unlike the current anonymous crypto-currency systems where there is no way for the sender to trace where their sent money will be spent further, our users should be able to control all of the delegations made by their successors and revoke them if necessary. Therefore we utilize *transaction depth* - the number of hops from the current transaction to the root one made by the owner and design a simple hash chain so that given the current transaction depth, users can easily find all of the subsequent transactions. Generally, if a transaction is made anonymously, it is impossible to infer the depth of it. However, since we now attach the depth as a field in the transaction, it should also be protected from the public network.

In summary, the following information should be obfuscated:

- The identities of the sender and receiver
- The identity of the capability
- The expiry date of the capability
- The transaction depth

E. Monero's Ring Confidential Transactions

1) ECC terminology:

- E: an elliptic curve in the form of $y^2 = x^3 + ax + b$
- G: a generator (or base point) on E
- l: a prime order of the base point G
- $\mathcal{H}$: a cryptographic hash function
- A pair (P, p) is a ECC public-private key pair if $p \in [1, l-1]$ and P is a point such that $P = p \times G$.

2) Ring Confidential Transactions: As mentioned in section II, Monero project tries to tackle all 3 problems of privacy at the same time by introducing the Ring Confidential Transactions (Ring CT) protocol, a combination of Confidential Transactions protocol [10] and ring signature [15].

First, the privacy of amount requires that the amounts in the inputs and outputs should be presented in some encrypted forms while still allow the network to check the balance (i.e. $\sum inputs = \sum outputs$). In other words, a homomorphic encryption C is required to transform the amount values, such that $C(a) + C(b) = C(a+b)$. In Ring CT, in order to hide an amount a, a transaction includes a Pedersen commitment [17] to a, which is computed as follows :

$$C(a, x) = xG + aH$$

where x is a secret blinding factor chosen by the sender and H is a point on E such that it is difficult to find the discrete logarithm of H with respect to G (i.e. a value n such that $H = nG$). For simplicity, suppose a transaction has 1 input amount a and 2 output amounts a_1, a_2, the commitments to those input and outputs are:

$$C_{in} = xG + aH$$

$$C_{out,1} = x_1 G + a_1 H$$

$$C_{out,2} = x_2 G + a_2 H$$

where $x - x_1 - x_2 = z$. If the balance condition is satisfied, i.e. $a = a_1 + a_2$ we will have

$$C_{in} - \sum C_{out,i} = (x - x_1 - x_2)G = zG \qquad (1)$$

which is a commitment to 0 with secret key z and public key zG. Thus, the sender can replace the raw values a, a_1, a_2 with 3 commitments $C_{in}, C_{out,1}, C_{out,2}$ and create a signature with public key zG to prove the equality $a = a_1 + a_2$.

To hide the input C_{in}, the sender can form a ring of $n+1$ members by selecting n other inputs from the public blockchain $\{C_{in,1}, \ldots, C_{in,n}\}$. Since only he can know z, he can generate a signature on that ring to prove his ownership without revealing the actual input. The amount and blinding factor are also encrypted and transfered to the recipient via an ECDH key exchange. To prevent double spending, the signature also includes a key image $I = p\mathcal{H}_p(P)$ where $\mathcal{H}_p(P)$ is a hash to point function, so that any attempt to spend the input twice can be detected by the re-appearance of I in the signature.

However, checking (1) is not safe enough to guarantee that the transaction is valid since the equation still holds for negative values due to overflow. For example, from an input of 5 coins, two illegal outputs can be -3 and 8. In this case, the validation is still successful as $5 = -3 + 8$, but 8 coins have been created out of nowhere. Confidential Transactions prevent this by *range proof* - a proof that a committed value must be within a valid range without disclosing the actual value. The idea of the proof is also based on ring signature. Readers can refer to the original papers [10], [12], [13] for more details.

IV. System details

A. Capability publication

At this phase the owner initializes all capabilities associated with his devices and sends them to the blockchain via $\mathsf{tx}_{\mathsf{publish}}$ transaction. For scalability, an off-chain capability storage similar to [5] can be used to store capabilities so that the blockchain only needs to maintain a reference to each capability. Besides the reference, a capability is also mapped to a unique point $\mathcal{M}$ on an elliptic curve. This point will serve as a capability ID that is used in all delegation transaction.

It is required that the discrete log with respect to two capability points $\mathcal{M}_1, \mathcal{M}_2$ must be difficult to find. Otherwise it would be possible to convert a capability to another. We compute the capability point $\mathcal{M}$ as follows:

$$\mathcal{M} = \mathcal{H}(\mathsf{CAP}, \sigma)G \qquad (2)$$

where σ is a shared secret between the owner and the device.

It is possible that a malicious user can flood the network with fake capabilities or send those to other users. In such cases, the publication can be restricted to only valid devices by having manufacturers co-sign the transaction. ChainAnchor [18] provides a mechanism for device commissioning that can be applied to this scenario.

B. Transaction destinations

Our delegation involves 3 types of public keys:

- User's primary address: each user joining the blockchain has a public key as his/her primary address.
- One-time sub address: an address that is derived from primary address in order to receive anonymous transactions. A transaction is destined to 2 addresses: the user and the domain the contains the device.
- Domain's address: a public key V that represents a domain/organization. The corresponding private key v is known to both the local proxy and all devices within the domain.

1) Deterministic user's sub address: Since it is desirable not to expose the recipient's ID, the primary address should not be used as the destination of the transaction. Unlike CryptoNote or Monero that use a one-time unlinkable address that makes sure only the recipient can spend the money, we want to allow the delegators to link all of the delegation originated from them. Therefore we derive the recipient's sub address in a deterministic way as follows.

We start with the root delegation from the owner Alice to user Bob whose public key is $B = bG$:

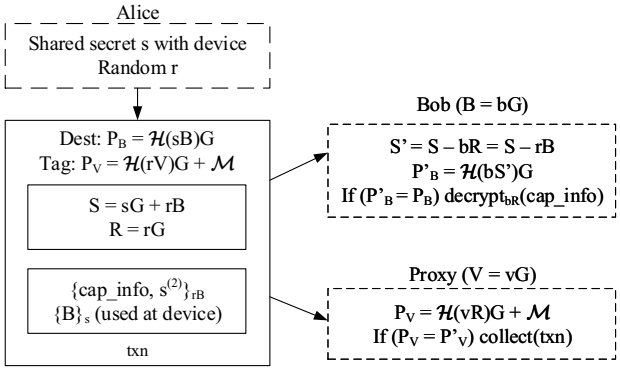


Fig. 3: Alice computes addresses and attaches necessary information to transaction txn. $\{X\}y$ denotes X is encrypted by key y, cap_info is the capability information, including CAP ID, expiry date, depth and blinding factors used in commitments.

- Alice chooses a secret s, which is also shared with her device, and computes the sub address P_B for Bob using Bob's primary address B: $P_B = \mathcal{H}(sB)G$. She also chooses another random secret r and computes $S = sG + rB$, $R = rG$.
- Alice encrypts all of the hidden values and blinding factors used in commitments using one-time shared secret with Bob $rB = brG = bR$ and attaches the encrypted information to the transaction, which is destined to P_B.
- Bob scans every incoming transaction and computes $S' = S - bR = S - rB$, then $P'_B = \mathcal{H}(bS')G$. The transaction is destined to Bob if P'_B equals to P_B.
- Since P_B is a one-time address, Alice encrypts Bob's primary address B using the secret s so that her device can use B for further verification.

Figure 3 demonstrates Alice's computation of addresses and how Bob recognizes his transactions.

If Alice wants to allow Bob to delegate the capability further, she will also encrypt a value $s^{(2)} = H(s)$ and attach it to the transaction. Now Bob can derive a sub address for Carol as $P_C = \mathcal{H}(s^{(2)}C)G$.

In general, a user with public primary address X who receives a capability at depth d will have the following sub address:

$$P_X = \mathcal{H}(s^{(d)}X)G \tag{3}$$

By computing recipient's addresses using hash chain as above, Alice is not only able to revoke Bob's capability but also Carol and any further user. It is straightforward to revoke Bob's by announcing his sub address P_B. In order to find Bob's transaction destined to Carol, recall that Bob has to provide his corresponding key image $I_B = p_B\mathcal{H}_p(P_B)$ in his ring signature. As the private key p_B is also known to Alice, she can easily compute I_B and look for the transaction that contains this key image and find Carol's sub address P_C. Knowing Carol is at depth 2, Alice can decrypt her primary address C and compute the private sub key p_C, thus the revocation can be continued. Figure 4 demonstrates a chain of delegation transactions starting from the owner Alice. It is crucial that Alice's revocation only works if Bob creates Carol's sub address properly using the correct depth and primary address. Although the confirmation process prevents Bob from lying about Carol's primary address, he can use a false depth. For example, instead of using $s^{(2)}$, Bob can raise it by 1 to $s^{(3)}$ and use it for Carol. Carol then tells the device that she is at depth 3 and since the address is still generated from the same seed, the authentication is successful though Alice can no longer trace back Carol unless she checks for every $s^{(i)}$. Therefore to force Bob to use the appropriate depth, the depth value should be attached to the transaction as a commitment so that Bob cannot change it to other values. We present how to create such commitment in section C.

2) Domain's sub address: As shown in figure 3, Alice also tags the domain to her transaction by computing the domain's sub address

$$P_V = \mathcal{H}(rV)G + \mathcal{M}^1 \tag{4}$$

The purpose of the domain's address is two-fold. First, similar to Bob, the local proxy can recompute the sub address $P'_V = \mathcal{H}(vR)G + \mathcal{M} = P_V$ so that it may store only transactions destined to its domain. Second, the tag also serves for revocation/confirmation purposes. As stated in section III-B, a $\mathsf{tx}_{\mathsf{delegate}}$ cannot be used for further delegation without a $\mathsf{tx}_{\mathsf{confirm}}$. Since the private key of $\mathcal{M}$ is known to the device, it also knows the private key p_V, hence can sign the transaction as an anonymous confirmation. The confirmation can be relayed by the local proxy, but the proxy is unable to sign the transaction by itself as it does not know the private key of $\mathcal{M}$, therefore compromise can be mitigated.

C. Delegation with commitments

1) Capability commitment:

a) Capability ID obfuscation and depth commitment: Suppose Alice is at depth $d-1$ of a delegation chain and is holding a capability CAP with ID point $\mathcal{M}$ and expiry time t. To make a delegation to Bob, first she needs to obfuscate her capability ID. We leverage Confidential Assets [11] by publishing an obfuscated ID M_{cap} as below

$$M_{cap} = \mathcal{M} + mG \tag{5}$$

where m is a blinding factor. After M is obfuscated, an input commitment to the depth $d-1$ will be:

$$M_{in} = (d-1)M_{cap} + m_1G \tag{6}$$

As the next depth must be d, the output commitment is

$$M_{out} = dM_{cap} + m_2G \tag{7}$$

[1]In practice $\mathcal{M}$ should be hidden so the proxy may not know which capability point to use. Therefore the tag instead uses a public obfuscated version of $\mathcal{M}$, which is presented in section C. In either case, only the device or the owner can compute the private key.

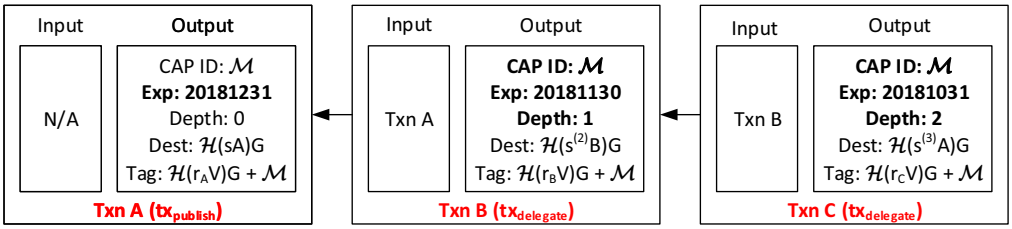


Fig. 4: An example of a delegation chain from Alice to Bob, then to Carol. **Bolds** are hidden information.

Since both $\mathcal{M}$ and d are hidden, it is necessary to build a ring signature that can prove that 1) M_{cap} is committed to $\mathcal{M}$ and 2) M_{out} and M_{in} are committed to d and $d-1$, respectively. We create a single ring signature that can prove both statements at the same time as follows. From equation (3) - (5) we have

$$M_{out} - M_{in} - M_{cap} = (m_2 - m_1)G \tag{8}$$
$$M_{out} - M_{in} - \mathcal{M} = (m_2 - m_1 - m)G \tag{9}$$

Hence

$$2(M_{out} - M_{in}) - M_{cap} - \mathcal{M} = [2(m_2 - m_1) - m]G = \mu G \tag{10}$$

As the left-hand side is a public key that only Alice can sign for with secret key $\mu = 2(m_2 - m_1) - m$, to prove that the commitments are committed to $\mathcal{M}$ and d, Alice can pick a number of inputs $M_{in,1}, ..., M_{in,n}$ and capability ID candidates $\mathcal{M}_1, ..., \mathcal{M}_n$ and create a ring signature on :

$$\{2(M_{out} - M_{in,1}) - \mathcal{M}_1 - M_{cap},$$
$$...,$$
$$2(M_{out} - M_{in,n}) - \mathcal{M}_n - M_{cap},$$
$$2(M_{out} - M_{in}) - \mathcal{M} - M_{cap}\}$$

b) Proof: we show that (10) is sufficient to guarantee that a malicious user cannot create illegal M_{cap} and depth. Suppose that Mike chooses $x, y \neq 1$ and computes the commitments as follows

$$\mathcal{M}' = x\mathcal{M} + mG$$
$$M_{in} = d\mathcal{M}' + m_1 G$$
$$M_{out} = (d+y)\mathcal{M}' + m_2 G$$

Hence

$$2(M_{out} - M_{in}) - \mathcal{M}' - \mathcal{M} = (2xy - x - 1)\mathcal{M} + \mu G \tag{11}$$

In order for Mike to sign with μ, $\mathcal{M}$ must be canceled out, implying $2xy - x - 1 = 0$. It is clear that $x = y = 1$ is the only solution, which means that Mike cannot cheat and create illegal commitments.

c) Capability commitment: After having the obfuscated M_{cap} for CAP, an input commitment to the capability with expiry time t is

$$C_{in} = xG + tM_{cap}$$

where x is another random scalar. Now to delegate CAP with a new expiry time t_1, two output commitments can be created as follows:

$$C_{out} = x_1 G + t_1 M_{cap} \tag{12}$$
$$\overline{C_{out}} = x_2 G + (t - t_1) M_{cap} \tag{13}$$

By the ring signature on $C_{in} - (C_{out} + \overline{C_{out}})$ and range proofs on t_1 and $t - t_1$, the network can verify that the new expiry time t_1 must not be later than the original one t.

The reason M_{cap} is used instead of $\mathcal{M}$ is it serves both as a blinded version of $\mathcal{M}$ and a public point similar to the point H in equation (1) so that the network can use to verify the ring signatures and range proofs. It should be noted that here C_{out} is the only valid delegated capability, the complement $\overline{C_{out}}$ is just a dump output that serves for the verification purpose.

Unlike money that cannot be double spent, capabilities can be transferred multiple times to different users. However as ring signature is linkable by the key image, it is possible to detect a repeated transfer of a capability. To avoid that, besides a normal capability output, a delegation can also include an echo output that sends back the capability to the sender but under a new address and invalidates the old one. The new echo address is generated in exactly the same way with the normal recipient's address, for example, Alice's echo address will be

$$P_A = \mathcal{H}(s^{(d)}A)G$$

D. Access request at device

In the first use of the capability received from Alice, Bob presents the transaction to the device and signs it with his primary keypair (B, b). The device then verifies the signature, decrypts the primary address B and checks if the sub address P_B is generated from the correct seed s. If everything is correct, it sends a $\mathsf{tx}_{\mathsf{confirm}}$ to CapChain by signing with the domain's sub address P_V. After the transaction is confirmed, Bob can use his sub address instead of his primary one for the signature and the device no longer needs to check the generation of the sub address. To prove the ownership of the capability received from Alice, Bob needs to:

1) Prove that his capability ID and expiration time are valid as these values are hidden from the transaction.

2) Prove that he is the recipient of the transaction.

Algorithm 1 Request verification at device

```
procedure VERIFY(txid)
    Verify commitment of capability
    if !is_confirmed(txid) then
        Verify commitment of depth d
        Verify user's sub address P_X = H(s^(d) X)G
        Post tx_confirm
    end if
    Decrypt user's primary address using s^(d)
    Verify user's signature by primary address
end procedure
```

To prove the capability information, from Eq. (5) and (12) we have

$$C = C_{out} - t_1\mathcal{M} = (x_1 + t_1 m)G \tag{14}$$

which is a public key with $c = x_1 + t_1 m$ as private key. Thus Bob can provide a signature with (C, c) as the public-private key pair. The device then can perform the same calculation as (14) to obtain the public key C and verify the signature.

To prove the identity, Bob needs to show that his sub address is generated by the correct depth d. Similar to the capability information, he also knows the private key of the public key $M_{out} - d\mathcal{M}$, hence can provide another signature. After verifying the depth, the device can decrypt Bob's primary address and check if Bob's sub address $P_B = \mathcal{H}(s^{(d)}B)G$. It is noticeable that any insider having the same capability but at a smaller depth than Bob can deduce his sub address as well as the private key. Therefore although this does not affect the delegation process as nobody except Bob can produce the ring signatures without the knowledge of the commitment blinding factors, Bob should use his primary address instead of the sub address as the proof of his identity.

Since the purpose of sub address is to facilitate the revocation process and the user's identity is proved using primary address instead, we only have the device check the generation of sub address at the first time a transaction is presented to reduce the computation overhead. The procedure is summarized in Algorithm 1.

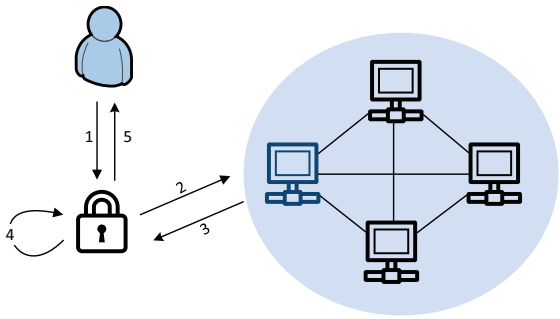
V. Experiments and Discussion

Our testbed includes an Arduino MKR1000 with a 32-bit ARM Cortex-M0+ MCU with 32 KB of SRAM and 256 KB of flash and a Raspberry Pi Zero W with a 1 GHz single-core CPU and 512 MB RAM. Both the Arduino and the Pi simulate smart devices, but the Pi is also a CapChain node that acts as a local proxy for the less powerful Arduino.

A. Blockchain performance

We build CapChain based on Monero's source codes. A test network is deployed on our MSU HPCC, which consists of 20 mining nodes and 20 wallet nodes that send transactions every 5 seconds. The Pi is a regular node that communicates with HPCC network through SSH tunneling. For convenience, we leverage the mining process to publish 4000 capabilities before starting the transfer. With the current Monero's implementation of proof of work, a new block is generated every 3 minutes on average, which means users should wait for at least 3 minutes before receiving a capability. For reliability, users may wait for more new blocks to make sure their transactions are on the main chain. However, such latency only occurs at the delegation phase as once the capability is received, it can be used at the IoT device without any more transaction involved, except $\mathsf{tx}_{\mathsf{confirm}}$ sent by the device to confirm that it has seen the capability.

Compared to the current size of a basic Monero transaction which is 13 KB, our transaction takes up to 30 KB due to additional signatures and range proofs. However, since an access right is often valid for a period of time, a delegation would happen less frequently than a money transaction, hence the growth of the blockchain should also be much slower. A Raspberry Pi with 1 GB spare storage (excluding the OS) can store more than 30000 delegations, which is scalable to larger organizations beyond that of home environments.



1. User sends a request (capability ID, transaction ID)
2. Device queries home proxy for the corresponding transaction
3. Proxy response (transaction is valid or not)
4. Device verifies signature
5. Device responses to user

Fig. 5: Testbed with a low-power Arduino as smart device and a trusted daemon running on blockchain

B. Performance at IoT devices

On the Arduino we used the ArduinoLibs [2], which supports Curve25519 [19] that is used in Monero/CryptoNote signature scheme. The performance is shown in Table I.

We found that the point multiplication operation contributes the most to the computation time. Since both the generation of sub address and signature verification requires 2 multiplications, they have similar speed. The verification of capability commitment takes the longest time due to 1 additional multiplication as in equation (14). Although the verification of depth should have a similar processing time, we found that with small values like depth which is often less than 10, doing a loop of point additions is much faster than multiplication, hence the actual processing time is reduced.

Although the signature verification is quite slow, we believe the performance can be improved with a slightly more

[2]https://rweather.github.io/arduinolibs/crypto.html

TABLE I: Processing time of request

Action	#multiplications	Processing time on Arduino MKR	Estimated time on Arduino Due
(1) Verifying sub address	2	1436 ms	315 ms
(2) Verifying depth commitment	3	1656 ms	363 ms
(3) Verifying user's signature	2	1394 ms	306 ms
(4) Verifying capability commitment	3	2125 ms	466 ms
(5) Other computations (parsing messages, decoding, etc.) and communication		111 ms	24 ms
Total round trip time at first use with confirmation		**6777 ms**	**1616 ms**
Total round trip time without confirmation (3)+(4)+(5)		**3685 ms**	**796 ms**

powerful hardware. According to ArduinoLibs benchmark, an Ed25519 signature verification [20] takes 306 ms on Arduino Due with ARM Cortex-M3 MCU, which is about 4.5 times faster than the Arduino MKR. Like CryptoNote, the verification of Ed25519 signature also takes 2 point multiplications, thus it should take a similar amount of time for the CryptoNote scheme to perform on the same hardware. Based on the reported benchmark from ArduinoLibs, we calculate a rough estimation of processing time on Arduino Due, assuming it is proportional to the processing time on the Arduino MKR. It is shown in table I that the total processing time without confirmation stage could be less than a second.

C. Consensus and incentives

Though our current implementation adapts proof of work from Monero, it can totally be replaced with an alternative consensus protocol such as [21], [22] that is less computational expensive and have lower latency. Consensus protocols that are suitable for IoT environments is one of the topics that we would like to explore in future works.

Most of current consensus protocols rely on a certain economic benefit that incentivizes users to participate and behave honestly. Since our objective is not to build a cryptocurrency, a bounty mechanism like [23] in which IoT data is provided as a reward can be an incentive. For example, device owners can publish a special capability for data acquisition and attach it to their transactions as a reward to miners. However, we do not eliminate the possibility of economic incentive as payments are usually required in many scenarios, for example, house and car rentals, parking services, etc.

VI. Conclusion

We present CapChain - an access control framework for sharing and delegation based on blockchain technology. CapChain is not only reliable thanks to the blockchain architecture but also preserves user privacy by hiding sensitive information about the access delegation from the public. Our experiments show the applicability and scalability of CapChain in IoT environments.

VII. Acknowledgement

This was supported in part by the National Science Foundation Grant No. CNS-1320561.

References

[1] J. Zhou, Z. Cao, X. Dong, and A. V. Vasilakos, "Security and privacy for cloud-based iot: Challenges," *IEEE Communications Magazine*, 2017.
[2] L. Wei, H. Zhu, Z. Cao, X. Dong, W. Jia, Y. Chen, and A. V. Vasilakos, "Security and privacy for storage and computation in cloud computing," *Information Sciences*, vol. 258, pp. 371–386, 2014.
[3] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[4] N. Kshetri, "Can blockchain strengthen the internet of things?" *IT Professional*, vol. 19, no. 4, pp. 68–72, 2017.
[5] G. Zyskind, O. Nathan, and A. . Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE Security and Privacy Workshops*, May 2015, pp. 180–184.
[6] P. Brody and V. Pureswaran, "Device democracy: Saving the future of the internet of things," *IBM, September*, 2014.
[7] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, "Fairaccess: a new blockchain-based access control framework for the internet of things," *Security and Communication Networks*, vol. 9, no. 18, 2016.
[8] P. Veena, S. Panikkar, S. Nair, and P. Brody, "Empowering the edge-practical insights on a decentralized internet of things," *IBM*, 2015.
[9] https://chain.com/docs/1.2/protocol/papers/whitepaper, [accessed 30-Jan-2018].
[10] G. Maxwell, "Confidential transactions," https://people.xiph.org/~greg/confidential_values.txt, 2015, [accessed 30-Jan-2018].
[11] A. Poelstra, A. Back, M. Friedenbach, G. Maxwell, and P. Wuille, "Confidential assets."
[12] N. van Saberhagen, "Cryptonote v 2.0," https://cryptonote.org/whitepaper.pdf, 2013, [accessed 30-Jan-2018].
[13] S. Noether, A. Mackenzie, and the Monero Research Lab, "Ring confidential transactions," *Ledger*, vol. 1, no. 0, pp. 1–18, 2016.
[14] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Security and Privacy (SP), 2014 IEEE Symposium on*, 2014.
[15] E. Fujisaki and K. Suzuki, "Traceable ring signature," in *Public Key Cryptography*, vol. 4450. Springer, 2007, pp. 181–200.
[16] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture." in *USENIX Security Symposium*, 2014, pp. 781–796.
[17] T. P. Pedersen *et al.*, "Non-interactive and information-theoretic secure verifiable secret sharing." in *Crypto*, vol. 91, no. 7. Springer, 1991.
[18] T. Hardjono and N. Smith, "Cloud-based commissioning of constrained devices using permissioned blockchains," in *Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security*, 2016.
[19] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *International Workshop on Public Key Cryptography*. Springer, 2006.
[20] S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (eddsa)," *RFC 8032*, 2017.
[21] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *In Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*, 2017.
[22] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.
[23] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *Open and Big Data (OBD), International Conference on*. IEEE, 2016.