PPCA: Privacy-Preserving Conditional Actions for IoT Environments Using Smart Contracts

Tam Le
Michigan State University
East Lansing, Michigan
letam@msu.edu

Matt W. Mutka
Michigan State University
East Lansing, Michigan
mutka@msu.edu

ABSTRACT

Automated tasks play an important role in both consumer and industrial IoT environments. In many scenarios, the IoT tasks are performed given certain conditions. To facilitate the tasks, it is necessary to delegate a third party to listen to events that trigger the conditions and issue commands to the IoT resources accordingly. However, without restriction, the third party may be over-privileged and able to control the resources unconditionally. We define the third party's permission to act under some conditions as a conditional action. We propose PPCA, a privacy-preserving service that allows users to create conditional actions in a decentralized platform using smart contracts. PPCA can guarantee strict privilege at the third party that holds conditional actions. By generalizing a variety of conditions into simple forms of conditional logic, the conditions can be verified in a privacy-preserving manner. We build a prototype of PPCA on Ethereum. The performance shows the feasibility of PPCA in IoT environments.

CCS CONCEPTS

• Security and privacy → Access control;

KEYWORDS

privacy, blockchain, smart contract, IoT

ACM Reference Format:

Tam Le and Matt W. Mutka. 2019. PPCA: Privacy-Preserving Conditional Actions for IoT Environments Using Smart Contracts. In *16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, November 12–14, 2019, Houston, TX, USA. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3360774.3360794>

1 INTRODUCTION

With the development of smart devices, automation plays an important role in both consumer and industrial IoT environments. In most scenarios, automated IoT devices are controlled according to conditions within the environment. For example, in home automation, a common scenario is to have a light turn on if occupancy is detected in a room by a thermostat. In terms of access control, the scenario involves 2 types of permissions: 1) to access data of the thermostat;

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MobiQuitous, November 12–14, 2019, Houston, TX, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-7283-1/19/11...\$15.00

<https://doi.org/10.1145/3360774.3360794>

Figure 1: A thermostat frequently reports occupancy status to the blockchain to control the light bulb.

2) to control the light. A trivial way to implement this automated task is to give a third party both permissions so that it can listen to events from the thermostat and command the light accordingly. However, giving control permission without restriction could allow the third party to be over-privileged, as it now can actually turn on/off the light any time regardless of the occupancy status. IFTTT (If-This-Then-That [1]) is a popular platform that serves as such a third party. It acts a bridge between different services and allows users to specify a set of conditional statements (called applets) to trigger their services. As an trusted party that stores a large number of API keys, IFTTT could be an attractive target to attackers who want to exploit the over-privileged permissions.

To prevent such attacks, it is necessary to restrict the third party so that it is only allowed to act given certain conditions. We call such type of actions *conditional actions*. As a distributed ledger, we find blockchain an ideal decentralized platform for conditional actions, where the blockchain can serve as a public witness to verify the given conditions, which can be defined in forms of conditional statements with the help of smart contracts. For example, the above home scenario can be implemented on a blockchain as described in Figure 1, which shows a thermostat that regularly uploads the occupancy status to a smart contract on the blockchain. If the status says “occupied”, the smart contract sends a message to the light, telling it to switch on.

Although the above implementation seems straightforward, it has several problems with privacy:

- If a condition is translated directly to a smart contract, it can reveal users’ sensitive information. For example, the occupancy status could tell if the home is vacant or not. Even if the data may be “sealed”, events triggered by the conditions can still be observed, which can help adversaries to study users’ activities.
- In most cases, the conditions rely on certain data providers. The data can be in the form of texts such as occupancy messages, but also numerical values such as weather data or locations. Due to the heterogeneity of the IoT, it is difficult to define a smart contract that involve various type of data for the conditions without losing privacy.
- Posting data to a blockchain regularly is not cost-effective, especially for payment-required platforms such as Ethereum.

In addition, having events triggered by the blockchain can cause delay as new blocks need time to be confirmed. This is not suitable for real-time services.

In this paper we design PPCA - a privacy-preserving service for conditional actions using smart contracts. Our model comes from the insight that most conditions can be described in forms of either texts or numerical values. Therefore, using homomorphic encryption, we can transform the data into commitments that are verifiable by the blockchain network but do not expose the real data.

2 SYSTEM DESIGN

2.1 Definitions

2.1.1 Condition components. A conditional statement consists of the following components:

- Data provider: provides data for the conditions. The thermostat in Figure 1 is an example of a data provider.
- Data reported by the data provider.
- Trigger: is the data that satisfies certain conditions.
- Action: specifies what happens when the conditions are satisfied.

Figure 2 describes two conditional statements and their corresponding components. For example, for a conditional statement “if my thermostat reports occupancy status as occupied, turn on the light”, the data provider is the thermostat, the reported data is the occupancy status, “occupied” is the trigger and the action is to turn on the light.

If my thermostat reports occupancy status as occupied, turn on the light.
If the sensor reports temperature higher than 75F, turn on the AC.

Figure 2: Condition components

In order for an action to be executed, besides the data provider, we also specify two other parties:

- Target service: is the endpoint of the action. The target service should be able to verify the conditional permissions and grant access accordingly. The light bulb in Figure 1 is an example of a target service.
- Watcher: regularly receives data from the data provider and watches for when the data satisfies the condition, then makes a request to trigger the target service accordingly. The watcher can be a home PC or a server in a larger-scale system.

2.2 Design goals

We address the following design goals:

- Strict privacy on blockchains: no private data should be exposed on the blockchain.
- Weak privacy with data provider and target services: data providers only provide data and do not need to be aware of the conditions that trigger the target services. Likewise, the target services also may process the access request without knowing about the triggers.

- Strict privilege: watchers can request the target services only if the conditions are satisfied.
- Permissions should be fast to confirm if the target is a resource-constraint device.

Figure 3: Non-privacy model with blockchain as a watcher.

2.3 A non-privacy model

We start with a straight-forward but non-privacy model as described in Figure 3. In this model, users first register their conditional statements on to the blockchain. The blockchain also serves as a watcher that regularly evaluates data from data providers to trigger the target service.

This model has several benefits. First, with the blockchain serving as a central endpoint, the data provider and target resource can be free from most heavy operations. Second, the target resource does not need to be aware of any condition. If the smart contract is well-designed, the blockchain will be a secure watcher that cannot have false execution and over-privileged access. The blockchain is also resilient to denial-of-service attack. If the condition is satisfied, the action must be performed without any interruption. However, since the data is sent to the blockchain, as previously discussed in Section 1, there may be significant delay when using the blockchain to trigger events. Additionally, it is a challenge to design a smart contract that not only can evaluate a variety of conditions but also does not expose user privacy.

2.4 The proposed model

Since having data posted directly to the blockchain is not ideal for privacy, we propose to have the watcher as a separate entity. In this model, only condition definitions are registered on the blockchain. The communication between the data provider and watcher can be carried out off-chain. When the received data satisfies the conditions, the watcher presents a proof to the target resource for a request. The target resource then retrieves the condition definitions from the blockchain and verifies the request.

Figure 4: Proposed model with watcher separated from the blockchain.

Figure 5: Workflow to verify a conditional action.

With most of the important data being transferred off-chain, the only sensitive data left on the blockchain is the condition definitions. A trivial approach to protect data privacy is to have the data encrypted. Following this approach, the user can first encrypt the entire condition before posting it to the blockchain, then the target service will need to decrypt it. There are several drawbacks with this approach:

- The condition cannot be modified. If the user wants to adjust a parameter, he/she would need to revoke the old condition and create a new one.
- It is not scalable. Since the condition is encrypted, only the target resource can decrypt it. If the user wants to use the condition to another target, he/she would need to either encrypt it with the new target’s key or share the key to the new target. In the latter case, a revocation scheme is required to manage all of the targets.
- A translator is needed to translate various conditions to a format that the target can use to verify requests.

Although there are various types of data, we notice that most conditions can be described by $data == x$, $data \leq x$ or $data \geq x$, where $data$ can be represented in either texts or numerical values. Therefore, the triggers can be categorized into two types:

- Exact values: usually in text forms. For example, the light status is “on” or “off”, the occupancy status is “vacant” or “occupied”, etc.
- Range values: usually in numerical forms. For example, the temperature is higher than 75F, the current location is within 10 feet from home, etc.

By generalizing the triggers to these 2 types, using Elliptic Curve Cryptography (ECC) and Pedersen commitments, we can published the triggers (specifically, $data$ and x) in the form of commitments to the blockchain instead of plain data. An advantage of using the Pedersen commitment scheme is that it has a homomorphic property that allows the construction of a proof that the committed value is in a certain range. Therefore, the target service can still verify the trigger using the proof without knowing what the real trigger value is.

The workflow is as follows:

- (1) The user uploads to the blockchain a commitment to the trigger value and the data provider that will provide the value.
- (2) The user grants the watcher a permission to a target device, specifying the trigger required to use the permission. The permission is also stored on the blockchain.

- (3) The watcher keeps receiving data from the data provider. If the data activates the trigger (i.e. the trigger condition satisfies), it asked the data provider for the commitment to the trigger data.
- (4) The watcher generates a proof that the trigger condition satisfies and makes a request to the target device.
- (5) Upon receiving the watcher’s request, the target device queries the blockchain for the permission and trigger commitment, then verifies the proof.

2.5 Threat model and assumptions

We assume that the data published to the blockchain cannot be tampered. Therefore, nobody can modify triggers or permissions once they are registered on the blockchain, except the user who defines the conditional action and may be able to revoke it through a revoke transaction.

The user is assumed to have both access to the data provider and target service. Hence, the watcher can be granted permission to access the data provider by the user. The watcher can be considered a representative of the user, as it can access both the data provider and target service, except that the action on the target service is conditional. The watcher may be compromised and attempt to make an action even when the condition is not satisfied. The watcher may also refuse to perform when the condition is satisfied. However, in case of compromise, we consider over-privilege more important than denial-of-service.

On the other hand, since conditional permissions depends on data providers, we assume that the data provider is honest, i.e. the provided data is always true. It is possible that the data provider may be compromised to provide false data, but this type of attack is not in the scope of this paper.

We consider the watcher and data provider as two separate physical entities. One can argue that if the data provider is honest, the watcher can be embedded into the data provider, which then can be allowed to command the target service directly. For example, the thermostat can be allowed to trigger the light. However, this is not possible if the action involves multiple conditions with multiple data providers. Additionally, in many cases, the data provider and target service are not managed by the same user (for example, using an external weather service to turn on lights at sunset). Therefore, allowing it to make a request would result in over-privilege access.

3 CONDITION TRANSFORMATION

3.1 Preliminaries

Our computations are based on Elliptic Curve Cryptography (ECC). An elliptic curve E is a set of points that satisfies the equation $y^2 = x^3 + ax + b$. A public key P with private key p is a point on E such that $P = p \times G$, where G is a base point on E . We assume that both user, data provider and watcher have a public key that is registered on the blockchain.

3.2 Transformation of range values

3.2.1 Range proofs using Pedersen commitments. For range values, our goal is to build a proof that $a < b$ without revealing both a and b . Particularly, the upper bound b is specified by the user and represents the condition for a request while value a is provided by

the data source. The watcher can make a request to a target device if it can present a proof that $a < b$.

To implement this kind of range conditions, we adopt the range proofs using Pedersen commitments in Confidential Transactions [18]. Instead of publishing the raw values of b , the user computes a commitment to b by

$$C(b) = bH + x_b G \quad (1)$$

where x_b is a secret key that can be shared with the watcher. Similarly, the data source also creates a commitment to a by:

$$C(a) = bH + x_a G \quad (2)$$

To prove that $a < b$, a prover can compute a commitment to $b - a$ by

$$C(b - a) = (b - a)H + yG \quad (3)$$

Since $C = C(b - a) + C(a) - C(b) = (y + x_a - x_b)G$ which is a public key, a signature by C can be used to prove the inequality if the signer knows y, x_a, x_b . Although no party knows all of the 3 secrets (the watcher only knows y and x_b while the data provider only knows x_a), it is possible for them to provide a co-signature. We apply MuSig [16], a multi-signature scheme that is designed for Bitcoin, to compute the co-signature.

It is noteworthy that the commitment $C(a)$ can be computed by any party not just the data source. Therefore, besides the inequality proof, the data source also needs to provide another signature to prove that it is the source of $C(a)$. Having too many signatures causes more computation overhead at not only the data provider but also the verifier. Hence, we incorporate the identity proof in the range proof so that the proving process will only result in one final signature. We provide more details in the next sections.

3.2.2 MuSig recaps. Let $\mathbb{X} = \{X_1, X_2, \dots, X_n\}$ be the public keys of the co-signers. Every signer computes the same X, L as follows:

$$\begin{aligned} L &= H(X_1, X_2, \dots, X_n) \\ X &= \sum H(L, X_i) X_i \end{aligned} \quad (4)$$

Each signer chooses a random value r_i , and shares $R_i = r_i G$ with the other signers, then they each compute their part of the co-signature for message m

$$s_i = r_i + H(X, R, m) H(L, X_i) x_i$$

where $R = \sum R_i$. The final signature is (R, s) where $s = \sum s_i$. The verifier verifies the signature by checking if $sG = R + H(X, R, m)X$.

3.2.3 Condition registration for numerical values. In this step the owner registers the data source's public-private key pair (D, d) and two commitments for the lower bound and upper bound (v_L, v_U) :

$$\begin{aligned} C_L &= v_L D + x_L G \\ C_U &= v_U D + x_U G \end{aligned} \quad (5)$$

where x_L, x_U are blinding factors that are shared to the watcher. The watcher also knows about v_L and v_U .

3.2.4 Data provider and watcher co-operate. When the watcher notices a trigger value v , it asks the data provider to co-sign a proof that $v_L < v < v_U$. First, to construct the proof, the watcher and data provider exchange a one-time random key, $X_W = x_W G$ and $X_D = x_D G$, respectively. With each party now holding 2 keys (one main key and one sub-key), the shared factors L, X from (4) can be computed by

$$\begin{aligned} L &= H(W, X_W, D, X_D) \\ X &= H(L, W)W + H(L, X_W)X_W + H(L, D)D + H(L, X_D)X_D \end{aligned} \quad (6)$$

Hence, the data provider can compute a commitment to v by

$$C_D = vD + H(L, D)D + H(L, X_D)X_D \quad (7)$$

and its MuSig part $\{R_D, s_D\}$ on a message m will be

$$\begin{aligned} R_D &= r_D G \\ s_D &= r_D + H(X, R, m) H(L, X_D) x_D + H(X, R, m) H(L, D) d \end{aligned} \quad (8)$$

On the watcher's side, to prove $v_L < v < v_U$, it prepares two commitments to $v - v_L$ and $v_U - v$ as follows:

$$\overline{C}_L = (v - v_L)D - H(L, X_W)X_W - H(L, W)W - x_L G \quad (9)$$

Similarly, the commitment for the upper bound is:

$$\overline{C}_U = (v_U - v)D + H(L, X_W)X_W + H(L, W)W - x_H G \quad (10)$$

The watcher can also generate the second part of the MuSig:

$$\begin{aligned} R_W &= r_W G \\ s_W &= r_W + H(X, R, m) H(L, X_W) x_W + H(X, R, m) H(L, W) w \end{aligned} \quad (11)$$

After exchanging all needed information, the watcher gathers 2 MuSig parts and computes the final signature $\{R, s\}$ where $R = R_W + R_D, s = s_W + s_D$. The final proof to be presented to the target device is $\pi = \{C_D, \overline{C}_L, \overline{C}_U, X_D, X_W, [R, s]\}$

3.2.5 Target device verifies the request. From equation (5), (7), (9), (10), it can be seen that

$$\begin{aligned} C_D - (C_L + \overline{C}_L) &= C_U - (\overline{C}_U + C_D) \\ &= H(L, D)D + H(L, X_D)X_D + H(L, W)W + H(L, X_W)X_W \\ &= X \end{aligned} \quad (12)$$

Therefore, given the aggregated MuSig on X , the target device can verify both the range proof and the identities of the watcher and datasource. Particularly, given the proof π , the target device performs the following operations to verify the request:

- Verify the range proof by checking if

$$\begin{aligned} C_D - (C_L + \overline{C}_L) &= C_U - (\overline{C}_U + C_D) \\ &= H(L, D)D + H(L, X_D)X_D + H(L, W)W + H(L, X_W)X_W \end{aligned} \quad (13)$$

- Verify the MuSig:

$$sG = R + H(X, R, m)X \quad (14)$$

- Checking (13)-(14) alone is not enough to guarantee that the condition is satisfied. Since the elliptic curve is a cyclic group, $v - v_L$ and $v_U - v$ may overflow, which can still make (13) correct even though v is invalid. Since the difference $v - v_L$ and $v_U - v$ do not expose v, v_L or v_U , they can be revealed to the device for overflow checking. To retrieve more privacy, a range proof using ring signature can be computed instead

Figure 6: Steps to construct a proof for a conditional action (CA) with range value and verification at the target service. The CA is referred by the ID id_{CA} on the blockchain.

of presenting the plain values. However, range proofs are more complicated and would take more gas/time to build.

Figure 6 describes the details of the protocol to construct the range proof and verify the conditional action at the target service.

3.2.6 Security guarantees.

- (1) The watcher cannot produce fake data and generate a proof by itself since it requires the data provider as the co-signer.
- (2) The watcher cannot generate a valid proof with data that does not satisfy the conditions.

Let v' be a value the does not satisfy the lower bound v_L , i.e. $v' < v_L$. The commitment to v' will be

$$C'_D = v'D + H(L, D)D + H(L, X_D)X_D \quad (15)$$

To satisfy (), the commitment $\overline{C'_L}$ generated by the watcher must be committed to $v' - v_L$. However, since $v' < v_L, v' - v_L$ is negative and will not pass the overflow checking. Similarly, if v' exceeds the upper bound v_U , the commitment will also be invalid.

3.3 Transformation of exact values

The transformation of exact values is simpler than range values. The user register an exact value v to the blockchain by mapping v to a point M as follows:

$$P = h(v, s)G \quad (16)$$

where s is a shared secret between the user and data provider. Since the data provider knows the private key of P , it can generate a signature with M as the public key. Therefore, a proof for trigger v can be a co-signature between the watcher with public key W and data provider with public key P .

Figure 7: Steps to construct a proof for a conditional action (CA) with exact value.

4 IMPLEMENTATION

4.1 Curve choice

We choose alt_bn128 as the elliptic curve for our ECC operations. Since alt_bn128 is supported by Ethereum in precompiled contracts, using it can reduce the cost of our contract. Although other curves may be chosen since technically the smart contract is only to store commitments and the ECC operations are performed locally at the front-end, using alt_bn128 enables the future use of blockchain as a watcher.

4.2 The front-end

We implement the front-end interface for users in Node.js and in Python for the watcher and data provider. With alt_bn128 supported, all the ECC operations can be also written in Solidity (the

smart contract language of Ethereum) and run in Ethereum Virtual Machines (EVM). However, we opt into Python since it runs faster than using EVM. The Python library for alt_bn128 is from [2].

4.3 The smart contract

The smart contract is implemented on Ethereum for registration and management of conditional actions.

4.3.1 Identities. Technically, each user needs an Ethereum address as an identity to make transactions. Although the Ethereum address is already a public key¹, it uses the secp256k1 curve instead of alt_bn128. Therefore, we manage a mapping from an Ethereum address to an alt_bn128 public key. Before using the system, each entity (user, data provider or watcher) uses their master address to register the public key on the smart contract.

4.3.2 Data structures.

Public keys. A list of registered alt_bn128 public keys as discussed above.

User permissions. As discussed earlier, we assume the user is already permitted to use the target resource and how the permission is granted is not in the scope of the paper. Therefore, we mimic the resource ownership by a table that records permissions held by a user. The permission is only an abstract field that represented by a 32-byte token. Depending on the target endpoint, the actual permission can be an API key, or a capability token, etc.

Commitment. Represents a trigger value that can be either to a text message or a numerical values. A commitment is identified by a tuple of:

- `commitment_value`: a Pedersen commitment to the value.
- `provider_address`: the Ethereum address of the data provider.
- `commitment_hash`: the 256-bit hash value of the commitment, used as a key to look up commitments.

Trigger. Represents the condition to activate a permission. It is identified by a tuple of

- `commitment_hash`
- `trigger_info`: a JSON string that contains details of the trigger, including type of trigger (i.e. lower bound, upper bound or exact value), scaling factor (since commitments are only for integers, floating-point values needs to be scaled before generating commitments). In addition, as discussed in section 3.2, the blinding factors are also needs to be shared to watchers. Therefore, `trigger_info` also includes these values in encrypted form that only the watchers can decrypt. To save transaction cost, the blinding factors can be shared via off-chain communication. However we choose on-chain storage for convenience so that watchers only need to maintain one communication endpoint to the blockchain.

Conditional action (CA). A CA is identified by

- `holder_address`: the Ethereum address of the permission's holder. The holder can be a watcher, or a normal user.
- `resource_address`: the Ethereum address of the target service that will verify the permission.

- `action_token`: represents the action that can be performed at the target resource. This is the token from the user permissions table. Only the owner of the token can issue the conditional action.
- `trigger[]`: an array of triggers needed to activate the token. Access is granted only if all of the triggers are satisfied. The trigger array represents the AND logic. To have the OR condition, a separate permission with the same {holder, resource, token} but different set of triggers can be created, so that the watcher only needs to present 1 permission that satisfies.

4.3.3 Functions. The functions that actually spend gas in out smart contract are:

- `registerPubkey()`
- `registerTrigger()`
- `registerConditionalAction()`

The cost of these functions is shown in Table 1.

Table 1: Ethereum gas cost of register functions.

Function	Gas	Estimated USD ²
<code>registerPubkey()</code>	69410	\$0.0732
<code>registerTrigger()</code>	121797	\$0.1285
<code>registerConditionalAction()</code>	313098	\$0.3303

In addition, there are also static helper functions that can be executed locally and does not cost gas for proof/signature generation and verification.

5 EVALUATION

As a proof-of-concept, we set up a private Ethereum network with 2 Dell laptops with Intel Core i5 processor and a Raspberry Pi Zero W with a single-core CPU. With these 3 machines, we perform 3 different tests where the data provider and target resource run on different platforms. While the watcher always run on the laptop, the data provider and target resource can both run on the Raspberry Pi. Since the watcher acts as a broker that can serve multiple conditions, it is not practical to have it run on a weak device. On the other hand, in many scenarios, the data provider and target resource are IoT resource-constraint devices.

Table 2: Node configuration.

Role	Test 1	Test 2	Test 3
Watcher	Dell	Dell	Dell
Data provider	Dell	Dell	R. Pi
Target resource	Dell	R. Pi	R. Pi

For each test, we evaluate the three following conditional actions (CA):

¹To be precise, it is the hash of the public key.

²The estimation is based on <https://ethgasstation.info>, as of September 2019.

CA1 (*Conditional action with exact trigger value*). “If the room is occupied, turn on the light”. In this CA, the data provider regularly sends “vacant” or “occupied”.

CA2 (*Conditional action with range trigger value*). “If the room temperature is higher than 75F, turn on the AC”. In this CA, the data provider regularly sends a random number in the range of (60, 80).

CA3 (*Conditional action with multiple trigger value*). “If my current location is within 10 ft from home, turn on the heat”. Since the location coordinate consists of 2 values, there are totally 4 triggers in this CA (each longitude/latitude has 2 triggers, lower bound and upper bound).

Table 3 shows the total time to perform each action, counting from when a trigger is detected until the request is verified. The processing time is broken down to two phases: 1) time to construct a proof between watcher and data provider and 2) time to verify a request at the target resource.

Table 3: Processing time.

		Test 1	Test 2	Test 3
CA1	Construct proof	0.049	0.053	0.60
	Verify request	0.161	1.556	1.693
	Total	0.210	1.609	2.293
CA2	Construct proof	0.191	0.269	2.862
	Verify request	0.322	3.878	3.731
	Total	0.513	4.147	6.593
CA3	Construct proof	0.180	0.378	4.419
	Verify request	0.917	5.165	5.377
	Total	1.097	5.543	9.796

As shown in the table, if the CAs are performed using regular PCs, the processing time is only within a second. CA1 with exact value has the best time, since it does not require any commitment. The bottleneck is at the Raspberry Pi, which takes 0.5 second to perform a multiplication. Hence, the more complicated the CA is, the longer it takes to verify a request. Although a range proof is more costly and may take more than 4 seconds, actions that involve range values usually are not time-critical and users often are unable to detect exactly when a trigger happens. For example, for CA2, since users usually are not aware of a change in temperature quickly, a few seconds of delay should not affect users’ experiences.

It is noteworthy that our experiments are not optimized. Although ECC operations on Python is faster than Javascript, C/C++ may be a better choice for constrained platforms. Unfortunately, it is not supported by Ethereum API. In addition, our model of Raspberry Pi only has a single-core armv6 CPU and 512 MB of RAM, which is the lowest configuration among other current models. A benchmark in [6] shows that a Raspberry Pi with a quad-core CPU can process thousands of signatures per second on the Ed25519 curve. On the other hand, our implementation can only process around 5 alt_bn 128 signatures per second even on a regular PCs. Therefore, we believe with an optimized implementation, the processing time should be significantly smaller.

6 SECURITY AND PRIVACY ANALYSIS

Watcher’s strict privilege PPCA can guarantee strict privilege on watchers if the data provider is honest. As shown in section 3, the watcher cannot trigger the target service when the conditions are not satisfied. The co-signature guarantees that the watcher cannot make a request without the data provider.

Blockchain privacy In Ethereum, the identities of users, watchers, data providers and target services are only protected by pseudonyms (i.e. Ethereum addresses). This means that if someone happens to know which service an address belongs to, they may be able to guess what type of conditional actions the user are using. However, since only commitments of values are stored on the blockchain, nobody can know about the real triggers. There have been proposals to use ring signatures or zero-knowledge proofs to bring anonymity to Ethereum [8, 17], but they are focused on payment applications instead of IoT.

Data provider and target service privacy Unless the addresses are known, the data provider would be unable to know what service its data can trigger. If the data is always sent to the watcher in the form of commitments even when it is a non-trigger, the data provider would not know when the trigger happens. However, if data is sent in normal form and only is committed when the watcher requests, the trigger may be revealed to the provider. How the data is transmitted should be decided by the user, depending on how strict of privacy they want.

Likewise, the target service would also not know about the data provider if it does not know the address. In the verification step, the service needs $v - v_l$ and $v_u - v$ to check for overflow. This may reveal how “far” the trigger value v is from the bounds, but the real value is never disclosed. Although a range proof with ring signatures for overflow checking can be computed [15, 18] to obtain more privacy, we chose not to implement it due to the high cost, especially on constrained devices. More recently, Bünz et al. propose Bulletproof [9], a much shorter and more efficient range proof. Although its performance on constrained devices has not been confirmed, Bulletproof could be a potential improvement for privacy.

7 RELATED WORK

Conditional tasks. IFTTT [1] allow users to define workflows to trigger a service in the form of conditional statements. Although starting with web services, IFTTT has become popular for home automation tasks. Besides IFTTT, other known platforms are Zapier [3] and Microsoft Flow [4]. However, they are more focused on productivity tasks than IoT. Recently, Zhou et al. introduce APEX [22], a system that determines and satisfies all of the preconditions before a command can be executed. APEX follows “backward” direction, unlike IFTTT which works “forward”, i.e. executing command when the condition satisfies. APEX mainly focuses on the execution order of preconditions and resolving concurrency between commands that have the same target. On the other hand, our focus is the role of the command issuer and how to prevent over-privilege commands.

Trust of data providers to smart contracts. A problem with smart contracts is that they cannot access real-world data (e.g. weather data, stock prices, etc.). An *oracle* is a service that provides

data from outside world to smart contracts. A challenge to oracle services is how to be sure that the data provided by oracles is correct. Town Crier [21] is an oracle service that provides authenticated data feed using a combination of Intel Software Guard Extension (SGX) and a smart-contract front end. It retrieves data from HTTPS-enabled websites and serves the data to contracts on the blockchain. In our framework, if the blockchain serves as watcher, the data provider can be considered an oracle. However, even if the data can be trusted, it is still a challenge to handle private data on smart contracts.

Private smart contracts. There are two main approaches to solve the lack of confidentiality and privacy on smart contracts: i) using cryptographic techniques such as zero-knowledge proofs, secure multi-party computation [13, 23]; ii) using trusted execution environments (TEE) such as Intel SGX [7, 10]. Although the former can provide strong privacy, it is too computationally expensive and is not suitable for IoT environments. On the other hand, the latter approach using TEE requires special hardware and trust on an attestation service (which is Intel in the case of SGX). Following none of these two approaches, Arbitrum [12] is a platform for scalable and private smart contracts that only publishes hashes of contract states on the blockchain. Although Arbitrum is shown to significantly reduce the on-chain computation cost compared to Ethereum, its performance on a real blockchain network was not evaluated. In addition, how Arbitrum performs on low-power devices is also not clear.

Blockchains for access control in IoT. There have been extensive studies on using blockchains for access control management in IoT systems [11, 19, 20]. In terms of smart contracts, WAVE [6] is a decentralized authorization system that uses smart contracts to manage permissions to IoT resources. User information in the permissions is protected using Identity-based encryption. Slock.it [5] provides a blockchain platform that allows users to rent, sell and share their assets using smart contracts. Access to the assets can be granted if certain rules are satisfied (e.g. the renter makes a payment). To the best of our knowledge, privacy-preserving conditional actions on smart contracts has not been explored yet.

8 CONCLUSION

In this paper, we propose PPCA, a decentralized service to create conditional automated tasks in IoT environments. Using smart contracts and homomorphic cryptography, PPCA can guarantee user privacy and strict privilege that the third party who watches for trigger events can perform actions only when the events satisfy the conditions. The prototype performance shows the feasibility of PPCA in IoT environments.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation. Any opinions, findings, conclusions or recommendations reflects the views of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

- [1] [n. d.]. <https://iftt.com/discover>.
- [2] [n. d.]. <https://github.com/solidblu1992/RingCTToken>.
- [3] [n. d.]. <https://zapier.com>.
- [4] [n. d.]. <https://flow.microsoft.com>.
- [5] [n. d.]. <https://slock.it>.
- [6] Michael P. Andersen, John Kolb, Kaifei Chen, Gabriel Fierro, David E. Culler, and Raluca Ada Popa. 2017. Wave: A decentralized authorization system for IoT via blockchain smart contracts. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2017-234* (2017).
- [7] Mic Bowman, Andrea Miele, Michael Steiner, and Bruno Vavala. 2018. Private data objects: an overview. *arXiv preprint arXiv:1807.05686* (2018).
- [8] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. [n. d.]. Zether: Towards Privacy in a Smart Contract World.
- [9] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 315–334.
- [10] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2018. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution. *CoRR abs/1804.05141* (2018). [arXiv:1804.05141](http://arxiv.org/abs/1804.05141) <http://arxiv.org/abs/1804.05141>
- [11] Ali Dorri, Salil S. Kanhere, Raja Jurdak, and Praveen Gauravaram. 2017. Blockchain for IoT security and privacy: The case study of a smart home. In *2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)*. IEEE, 618–623.
- [12] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. 2018. Arbitrum: Scalable, private smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1353–1370.
- [13] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*. 839–858. <https://doi.org/10.1109/SP.2016.55>
- [14] Tam Le and Matt W. Mutka. 2018. CapChain: A Privacy Preserving Access Control Framework Based on Blockchain for Pervasive Environments. In *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 57–64.
- [15] Greg Maxwell. 2015. Confidential transactions (2015). URL: https://github.com/ElementsProject/elementsproject.github.io/blob/master/confidential_values.md (2015).
- [16] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. 2018. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography* (2018), 1–26.
- [17] Sarah Meiklejohn and Rebekah Mercer. 2018. Möbius: Trustless tumbling for transaction privacy. *Proceedings on Privacy Enhancing Technologies* 2018, 2 (2018), 105–121.
- [18] Shen Noether, Adam Mackenzie, et al. 2016. Ring confidential transactions. *Ledger* 1 (2016), 1–18.
- [19] Aafaf Ouaddah, Anas Abou Elkalam, and Abdellah Ait Ouahman. 2016. FairAccess: a new Blockchain-based access control framework for the Internet of Things. *Security and Communication Networks* 9, 18 (2016), 5943–5964.
- [20] Otto Julio Ahlert Pinno, André Ricardo Abed Grégio, and Luis CE De Bona. 2017. Controlchain: Blockchain as a central enabler for access control authorizations in the IoT. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 1–6.
- [21] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM, 270–282.
- [22] Qian Zhou and Fan Ye. 2019. APEX: automatic precondition execution with isolation and atomicity in internet-of-things. In *Proceedings of the International Conference on Internet of Things Design and Implementation*. ACM, 25–36.
- [23] Guy Zyskind, Oz Nathan, and Alex Pentland. 2015. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471* (2015).