Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 9968743, 14 pages
https://doi.org/10.1155/2021/9968743

Research Article

Fast Policy Interpretation and Dynamic Conflict Resolution for Blockchain-Based IoT System

Yaozheng Fang, Zhaolong Jian, Zongming Jin, Xueshuo Xie, Ye Lu, and Tao Li

College of Computer Science, Nankai University, China

Correspondence should be addressed to Xueshuo Xie; xueshuoxie@mail.nankai.edu.cn

Received 9 March 2021; Revised 29 April 2021; Accepted 28 June 2021; Published 9 July 2021

Academic Editor: Chi-Hua Chen

Although the blockchain-based Internet of Things (BC-IoT) has been applied in many fields, it still faces many security attacks due to lacking policy-based security management (PbSM). Previous PbSM is usually time-consuming, which is difficult to integrate into BC-IoT directly. The high-latency policy conflict resolving in traditional PbSM cannot meet the BC-IoT's low-latency requirement. Moreover, the conflict resolution rate is low as the PbSM usually neglects the runtime information. Therefore, it is challenging that achieving an efficient PbSM for BC-IoT and overcomes both time and resource consumption. To address the problem, we propose a novel PbSM for BC-IoT named FPICR to realize fast policy interpretation and dynamic conflict resolution efficiently. We first present policy templates based on system log to interpret policy in high speed in BC-IoT. Benefiting from matching the characteristics of the system processing, FPICR supports interpreting a policy into the smart contract directly without complex content parsing. We then propose a weighted directed policy graph (WDPG) to evaluate the importance of the deployed policies more accurately. To improve the policy conflict resolution rate, we implement the resolution algorithm through reconstructing the WDPG. Taking the traits of these properties, FPICR thus can also remove the redundant data to compress storage space by the WDPG. Experiment results highlight that FPICR outperforms the baseline in all measure metrics. Especially, compared with the state-of-the-art method, the speedup of interpretation in FPICR is about up to $2.1\times$. The conflict resolution rate in FPICR can be improved by 6.2% on average and achieve up to 96.1%.

1. Introduction

Internet of Things (IoT) has been widely used in many fields such as smart cities [1], industrial control [2, 3], online gaming [4], and distributed computing system [5, 6]. However, massive IoT devices and networks face various real threats [7], in consequence, the security of IoT is becoming more and more important. Blockchain has many advantages such as decentralization, trustworthiness, anonymity, and immutability. Owing to these merits, blockchain has become a major solution to a lot of domains, such as supply chain, healthcare, and transportation [8]. The success stories in these domains inspire researchers in IoT to apply blockchain by using smart contract to address the problems of single-point failure and data security in IoT system. Therefore, blockchain-based IoT system (BC-IoT) has been a research hotspot [9, 10], and a proposal called smart contract-based access control [11] falls into this category and already demonstrates the benefits of adopting blockchain for IoT system.

Although the new features of BC-IoT can help protect against the security risks to a certain extent, BC-IoT still faces some real attacks such as DDoS, on-off attack, and Parity Wallet attack [12–15], due to lacking policy-based security management (or short, PbSM). In fact, the attacks can be defended by deploying security policies in policy-based security management (PbSM). Unfortunately, traditional PbSM methods are so time-consuming and resource-intensive. These shortcomings make it difficult to integrate these PbSM methods into the BC-IoT system directly and freely. For example, traditional PbSM methods resolve policy conflict through comparing static priorities, the time-consuming process thus cannot meet the requirements of low latency in the BC-IoT system. And neglecting run-time information in the traditional PbSM methods leads to the rate of policy conflict resolution degradation. Besides, the policies in PbSM also bring high storage cost, which is not a reasonable choice for the resource-constrained devices in BC-IoT. Therefore, traditional PbSM methods are inefficient for BC-IoT.

It is challenging to achieve an effective and efficient PbSM in the BC-IoT system. First, it is difficult to make full use of policy language expression to interpret policy quickly and accurately, according to the characteristic of smart contract. It should be noted that reducing interpretation latency by removing complex analysis should meanwhile ensure the accuracy of interpretation. Second, it is troublesome to find out the crucial information from larger-scale storage in BC-IoT system than other common systems. Because each node in BC-IoT executes the smart contract when it is invoked and generates additional data, it is hard for PbSM to take advantage of such system runtime information to resolve policy conflict dynamically. Third, it is also difficult to identify which is the redundant part in the policy storage space, since the dependencies on each other among various smart contracts are so complex, and removing the redundancy should not negatively impact on the conflict resolution algorithm. To meet the abovementioned challenges, in this paper, we propose a novel PbSM method named FPICR, which focuses on realizing fast policy interpretation and high conflict resolution rate with lower storage cost in the BC-IoT system. First, we present a new policy interpretation method through utilizing system logs to describe system states. This interpretation meshes with the running characteristics of smart contract, policies thus in FPICR can be interpreted in a short time. Second, we propose a dynamic policy conflict resolution algorithm to improve resolution rate, which can exploit runtime state information such as policy weight, to provide more accurate decisions rather than the presetting. Third, we design a weighted directed graph structure that can help determine the importance of the dependency between each policy and facilitate to remove the redundancy conveniently. Our innovations and major contributions in FPICR are highlighted as follows:

(i) We present a novel policy interpretation method based on system log. This method can interpret a policy into a smart contract through simple notation parsing rather than complex semantic and lexical analysis. The speedup of the log-based interpretation achieving is about up to $2.1\times$ compared with XACML-based interpretation

(ii) We propose a weighted decision algorithm to resolve policy conflict by utilizing the policy execution times and other important information of system runtime. Such an algorithm can evaluate the importance of the security policies in fine-grained. This algorithm facilitates FPICR to increase the rate of conflict resolution up to 96.1%, which is higher than the traditional method by 6.2%

(iii) We design a weighted directed policy graph to store the dependencies among security policies. This policy graph can remove the redundant parts to reduce the ledger size in BC-IoT. Compared with the traditional method, the degradation of storage can be achieved by about 17%

2. Background and Motivation

In this section, we introduce the minimal background about policy-based security management (PbSM) in BC-IoT, followed by the discussion on the related work. Lastly, we conclude the challenges of achieving effective and efficient PbSM in BC-IoT.

2.1. Traditional PbSM. Traditional PbSM methods are inefficient for BC-IoT. In particular, the reasons in detail are as follows. They usually consist of four parts, policy administration point (PAP), policy decision point (PDP), policy information point (PIP), and policy enforcement point (PEP). PAP is responsible to describe policies by various high-level policy languages. Therefore, in order to make sense in the BC-IoT system, a policy should be first interpreted from high-level policy language to the codes by following the rules of smart contract. This process often suffers from high latency due to complex syntax and lexical analysis. Massive policies may cause policy conflict, and PDP provides the algorithms to solve policy conflicts. PIP can also help tune policy by retrieving system runtime-related information. However, the resolution results of existing algorithms in PbSM do not always work, since most of these algorithms utilize the limited static priority and policy comparison, they cannot make full use of the rich information of system runtime to address the conflicts dynamically. Once there are lots of conflicts to be resolved, they have to be handled manually. Consequently, traditional PbSM methods are so time-consuming and heavy labor works. Furthermore, because PIP stores the whole security policies which can generate lots of redundant smart contracts on blockchain, traditional PbSM will bring high storage cost if be integrated into the BC-IoT system directly. This heavy burden on storage will limit the ability of BC-IoT devices greatly. PEP is responsible for the execution and enforcement of policies. The first three time-consuming and resource-intensive parts in traditional PbSM also leads to the low efficiency of PEP, since PEP has to wait for the results of the preprocessing.

2.2. Related Work. FPICR is closely related to three aspects: policy interpretation, conflict resolution, and the smart contract as explained as follows.

The typical policy interpretation focuses on two main scenarios: network management and security management according to the application field's requirements [16]. Varadharajan et al. [17] propose a policy expression language based on the routing syntax, in order to realize the network packages routing. Lara and Ramamurthy propose an understandable language in OpenSec [18] which could realize the automatic reaction to the network events. Moreover, an expressive language is presented by Gember et al. in [19] to control data traffic flow, so as to protect the privacy information of devices. In the papers [20], a standard firewall rule is given to create a blacklist. Furthermore, in the paper [21], the authors describe the security policy relying on the BNF notation. Besides, the access control rule languages are mostly based on the XACML [20], P3P [22], and so on. However, these methods for policy interpretation usually utilize a time-consuming two-step process to realize interpreting policies from the high-level expression to the executable program.

Resolving policy conflict in traditional methods often take advantage of the most commonly used priority mechanism [23, 24], and this method is applied to deal with conflicts in many current architectures [25–29]. However, rationally assign priority to each policy is almost impossible. It is even challenging for a well-trained administrator in this field. There are also other methods for conflict resolution, such as several rules combining algorithms to support conflict resolution strategies in [30], the conflict matrix in [31], the policy conflict analysis for QoS in [32], and the context-aware conflict resolution in [33]. Moreover, there are several conflict resolution strategies provided in XACML (i.e., permit-overrides and deny-unless-permit). Nevertheless, differing from FPICR, these works cannot make use of system running information to resolve the conflicts dynamically.

Defending attacks in blockchain system by smart contracts is an effective method [34]. Zhang et al. [11] propose a smart contract-based framework to implement distributed access control, but they neglect using the possible dependencies between policies. To manage the endpoint devices in IoT system, Novo [35] presents a decentralized security management technique, which can utilize devices by defining operations in smart contracts. Besides, Alphand et al. [36] design IoT chain as the security management architecture, which allows the access token to be stored in the smart contract to ensure data correctness. In order to protect private data, Kosba et al. [37] provide a framework for building public and private smart contracts. They focus on the smart contract-based access control implementation, but lead to undesirable results due to lacking policy enforcement, especially conflict resolution.

2.3. Challenges and Goals. The PbSM needs to be more effective and efficient to meet the requirements of real applications. Improving the performance of PbSM can be realized through fast interpretation and resolving policy conflicts efficiently. We identify three fundamental challenges and our goals as follows.

First, it is difficult to realize fast policy interpretation without missing the accuracy of interpretation results. Policy interpretation needs complex and time-consuming analysis, removing the analysis may miss important information. Besides, policy interpretation in BC-IoT should consider the characteristics of smart contract (e.g., GAS).

Second, resolving policy conflicts according to system runtime information is nearly impossible in BC-IoT as discovering the crucial information from large-scale data in BC-IoT is challenging. BC-IoT has larger-scale data than the traditional systems because BC-IoT generates additional data related to blockchain and smart contract (e.g., blocks and encryption information). As a result, resolving policy conflicts dynamically is harder than the traditional PbSM systems.

Third, security policies have complex dependencies, which is important to the resolution of policy conflicts. It needs a larger storage space to store the dependency information, so the policy storage space in BC-IoT has many redundant parts. It is difficult to remove the redundant parts and the policy dependencies without negatively impacting on the resolution of policy conflicts.

3. System Overview

FPICR is designed to achieve fast interpretation and dynamic policy conflict resolution for general BC-IoT. FPICR consists of two key components (see Figure 1), log-based policy interpretation and policy graph reconstruction. The overall process in Figure 1 is as follows. When the BC-IoT system requires adding or updating policies to protect against attacks, FPICR first performs the policy conflict detection and resolution module to determine whether this new policy can be deployed on the system or not. A policy that can be deployed will be interpreted into a policy smart contract (or short, PSC) by a certain log-based template. The templates are obtained through analyzing the relationship between items among the system logs. Once the policy has been deployed, it will be inserted into the weighted directed graph to record the corresponding dependencies on each other policies. This policy graph then can be reconstructed and updated to generate a new policy graph contract (PGC). A PGC is responsible for saving the property data of PSC and provides parameters such as weight, the number of access, or other information that the conflict resolution algorithm desires to utilize.

4. Fast Interpretation

In this section, we introduce how to realize fast interpretation with system logs and templates in our FPICR. We first design the template structures by summarizing and simplifying the policy description which is based on the system logs. We then interpret the policy into the smart contract by the converting process relying on module matching. These processes are lower latency and less resource-consuming.

4.1. System Log. System log can reflect the system state and record various special events [38], so there are many important data that can be used in fine-grained for preventing security threats. Moreover, in general, there are also lots of regular formal logic in the system logs. Specifically, there are various kinds of modes, and we select the three typical modes in the system log to help us explain the details as follows:

Single implies that to define a threat can only use one entry of the log. For example, rebooting a device can generate a record in log, such as system status: locked, which represents the system is deadlocked currently.

4.1.1. Iteration. The iteration stream is a workflow of a certain process, thus, the iteration can also be used as the basis for judging whether the system has appeared exceptions or not.

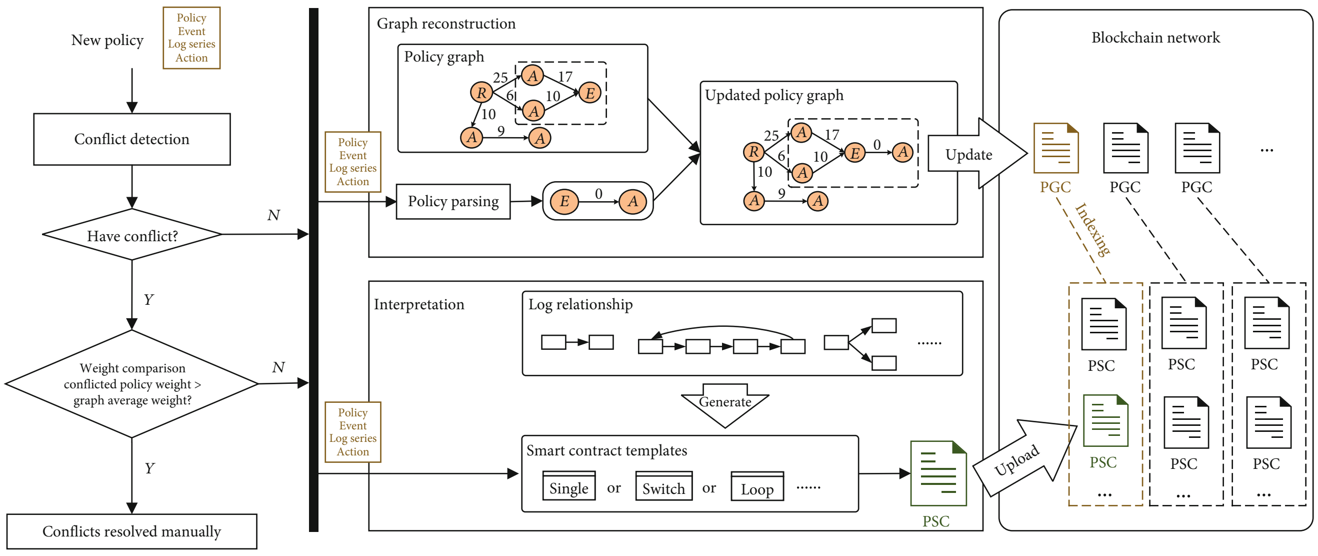


Figure 1: The FPICR architecture.

4.1.2. Loop. Loop refers to log entries repeating according to a certain regulation. When a loop appears in system log, the system may suffer from attacking or exceptions. For instance, a large repeat state of TCP retransmission means flooding attack happened. Based on the above explanations, in consequence, these observations facilitate us to skillfully combine the characteristic between system log and smart contract to speed up the interpretation.

4.2. Policy Template. In FPICR, an innovation is that we design templates based on system logs to describe various policies accurately. The policy template consists of four fields, MODE, CONDITION, ACTION, and CONF, respectively. Policy template has referenced the ECA (event-condition-action) paradigm [39]. A common template can be defined as follows:

$$LbP \coloneqq [\text{MODE}][\mathscr{T}, N][\$\text{SRC, ACT}][\text{CONF}]. \tag{1}$$

MODE refers to the type of relationships among the log entries. There are several alternative options such as single, iteration, and loop. T represents specific entries of the system log. N is denoted as the times that T appears. SRC represents the resource in the specific entries. ACT is the action on the corresponding resource SRC. CONF specifies some actions which are conflicted with the expecting action described in the field ACT. Hence, CONF is the basis for conflict detection. We give an instance to explain how the policy template works. A policy, such as when the number of TCP retransmission from the same host greater than 10, blocking this host, can be denoted as follows:

$$\begin{aligned} LbP_{eg} \coloneqq \quad & [\text{LOOP}]\left[{}'\$\text{SRC TCP\_Trans}', 10\right], \\ & \left[\$\text{SRC}, {}'\text{BLOCKING}'\right]\left[{}'\text{OPEN}'\right]. \end{aligned} \tag{2}$$

Moreover, FPICR supports complex policies owing to enabling the fields in the template to be nested. Because of meshing with the characteristics of smart contract, the interpretation based on policy template can be program-friendly, machine-readable, and time-saving.

We have summarized some common security threats and attacks in Table 1, and we can observe that the designed log-based templates have strong policy descriptive capacity.

4.3. Converting into Smart Contract. Policy interpretation in FPICR refers to convert the log-based policy into an executable smart contract that is PSC. As shown in Figure 2, a PSC is usually composed of three parts as follows:

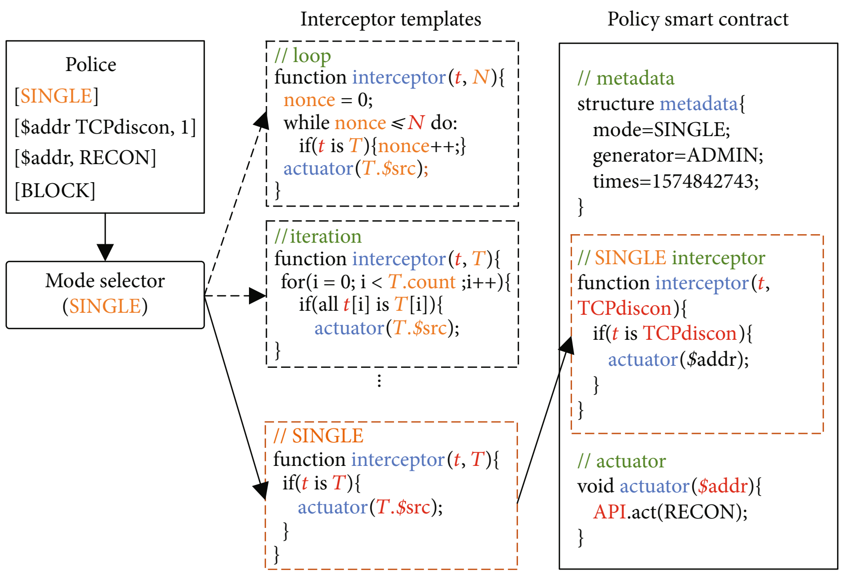
(i) Metadata is implemented as an extensible structure that can record some important property information of the policy. For instance, metadata may include the policy mode, the unique identification number, the timestamp, the policy generator, and the other parameters

(ii) Interceptor is used for gathering and checking the system log. Interceptor also has several templates, which are corresponding with the policy template, to facilitate fast converting for the policy interpretation. Once the gathered information in the interceptor matches with the parameter in CONDITION field of the policy, the corresponding actions described in ACT of the policy will be triggered

(iii) Actuator can activate specific actions when corresponding conditions fulfill the requirement. It should be noted that a common defense program usually only gives the abstract expressions rather than implementation for an action set. The reason is that devices in IoT system are actually vendor-dependent, thus, we have to call the hardware device APIs to instantiate an actuator

Table 1: Common BC-IoT security attacks and the countermeasures based on log-based policy.

Security threats	Mode	Condition	Action
Node capture attack	Single	['$num in Consecutive exception', t]	['SELF', 'ISOLATION']
Sleep deprivation	Single	['$dev TIME_SLEEP DEL', 1]	['$dev', 'REBOOT_TIME_SLEEP']
Flooding attack	Loop	['$SRC TCP_Transmission', 10]	['$SRC', 'BLOCKING']
Blackhole attack	Iteration	['$addr Data_recv, Data_recv', 100]	['$addr', 'BLOCKING']
Malicious injection	Single	['$str inserted abnormally', 1]	['$str', 'DELETE']
Homing attack	Loop	['$SRC ConnectReq', 20]	['$addr', 'BLOCKING']
Eavesdropping	Iteration	['$Addr EaringChel', 2]	['$addr', 'SHUTDOWN']
Password attack	Loop	['$Usr PwLogin', 5]	['$*USR*', 'NOLOGIN']
Hardware bugs	Single	['$bugid occured in $usr']	['$usr', 'EXIT']



```
Police
[SINGLE]
[$addr TCPdiscon, 1]
[$addr, RECON]
[BLOCK]
```

```
// loop
function interceptor(t, N){
  nonce = 0;
  while nonce ⩽ N do:
    if(t is T){nonce++;}
  actuator(T.$src);
}
```

```
//iteration
function interceptor(t, T){
  for(i = 0; i < T.count ;i++){
    if(all t[i] is T[i]){
      actuator(T.$src);
}
```

```
// SINGLE
function interceptor(t, T){
  if(t is T){
    actuator(T.$src);
  }
}
```

```
// metadata
structure metadata{
  mode=SINGLE;
  generator=ADMIN;
  times=1574842743;
}
// SINGLE interceptor
function interceptor(t,
TCPdiscon){
  if(t is TCPdiscon){
    actuator($addr);
  }
}
// actuator
void actuator($addr){
  API.act(RECON);
}
```

Figure 2: The policy interpretation based on template.

Mode selector in Figure 2 is used for choosing a desirable interceptor template in terms of the MODE parameters of the policy template. The other information in the policy will be filled into the selected template directly.

5. Conflict Resolution

To resolve policy conflict fast, we present weighted directed policy graph (WDPG) to organize deployed policies. WDPG is to store the dependencies between each policy and record the system runtime information. These dependencies and information can facilitate WDPG reconstruction to realize policy conflict resolution in FPICR.

5.1. Weight Directed Policy Graph.

5.1. Weight Directed Policy Graph. A WDPG consists of one or more policies. A policy in accordance with the ECA paradigm can be defined as follows:

$$\begin{cases} P = \langle V, E, \varphi \rangle, \\ V = \{v_{\text{event}}, v_{\text{action}}\}, \\ E = \{e_{\text{condition}}\}, \\ \varphi = \langle v_{\text{event}}, v_{\text{action}} \rangle. \end{cases} \tag{3}$$

P is a policy that consists of two vertexes and an edge. V represents the set of vertexes. E is the set of events. v_{event} and v_{action} represent an event and an action, respectively. $e_{\text{condition}}$ is the edge between the vertex v_{event} and the vertex v_{action} (see

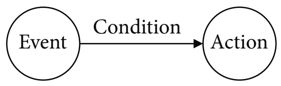


Figure 3: The security policy in policy graph.

Figure 3). φ represents to trigger an action execution when $e_{\text{condition}}$ meets requirement.

The weight of an edge in our WDPG can be defined in terms of the rich system runtime information. In other words, the meaning of the weight can be different in different applications, for example, the policy's priority, the execution time, the resource consumption, and so on. Because the execution frequency of action can reveal the importance of a policy, we denote weight by frequency in this paper. In system runtime, the frequency can be denoted as the called times of the corresponding smart contract which reflects the system state. Once a policy smart contract is involved, the relevant weight can be increased. The high frequency implies the policy contributes to a BC-IoT system relative greatly than other policies. Thus, WDPG can utilize the system runtime data to evaluate the importance of a policy as shown in Figure 4. The corresponding policies in the WDPG are listed in detail in the bottom of Figure 4. In the case, the policy $P1$ If NETFLOW is greater than 50, enable the firewall, with weight number 40, represents that this policy has been executed 40 times. As a result, the policy $P1$ is more important than the policy $P2$ which weight is 35. When the weight changes with the dynamic system runtime data, the importance of a policy to a BC-IoT system may be changed together. Therefore, WDPG is more dynamic and flexible than the other methods by presetting static priority for policies.

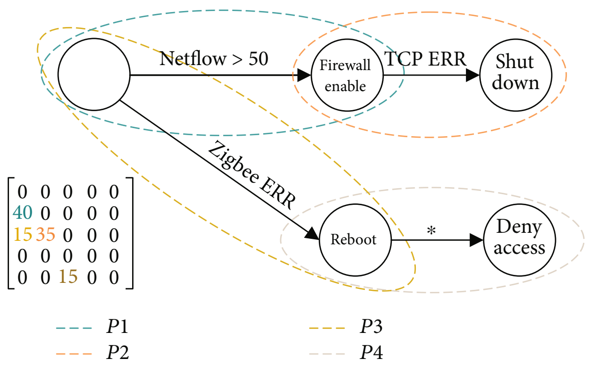
5.2. Graph Reconstruction and Updating. Policy conflict refers that there are inconsistent actions on the same resource. The conflicts are caused by multiple rules or rule instances. Policy conflict can make the state of system uncertain.

Because there are lots of dependencies between policies, it is challenging to process policy conflicts. In particular, adding or updating policy may generate redundant data, and deleting or deactivating a policy may cause the associated policies to be affected negatively. For such a difficult problem, we can address it through reconstructing WDPG, which is in charge of storing and updating the relationship between policies. In fact, the key process to solve policy conflicts is just reconstructing and updating the corresponding WDPG. We take an example to explain WDPG reconstruction flow (see Figure 5). We utilize an adjacent matrix as the storage structure of WDPG. We assume that when a new policy described by a pair of vertex and edge will be added into the existing WDPG, there is a conflict between policy A and F. A should be replaced by following certain requirement. Thus, we will delete A and add the new policy F into the WDPG. To do this, we first delete the row and column data related to A in the adjacent matrix which stores the previous WDPG. Then, we delete C, the successor vertex of A. At last, we add F into the WDPG and update the weight value in the corresponding adjacent matrix.

An observation to evaluate the importance of a policy in WDPG is to calculate the in degree and the out degree of each vertex, respectively. A high out degree means a relative strong dependence. We can define the concept named impact factor (IF). It is a value reflecting the influence and importance of policy. IF can be obtained through calculating the sum of in and out degrees. Besides, we can utilize the average of all policy impact factor (or short, AvgIF) as a threshold to determine whether the new conflict policy can be updated directly or not.

Therefore, we propose a policy conflict resolution algorithm by comparing a policy IF with AvgIF (see Algorithm 1); this algorithm helps simplify the conflict processing. For example, while a conflicted policy's IF is smaller than AvgIF, the policy will be replaced by the new one due to being not important enough for the whole BC-IoT.

```
Algorithm 1: Policy conflict resolution.Input: Policy graph G, new policy P
Output: BOOL
  function CheckG, P
    $N \longleftarrow count(G.node)$;
    $TotalIF \longleftarrow 0$;
    fori in Ndo
      forj in Ndo
        $TotalIF \longleftarrow G[i][j] + TotalIF$;
      end for
    end for
    $AvgIF \longleftarrow TotalIF/N$;
    $W \longleftarrow G[i][j]$; ; //Assume that $i$, $j$ is conflicted policy.
    if$W > AvgIF$then
      sendPolicy($P$); //send $P$ to application for resolving conflict.
      return FALSE;
    else
      GraphReconstruction($P$);
      return TRUE;
    end if
  end function
  function GraphReconstructionP
    fork in Ndo
      if$G[i][k]$ ==1 then
```



	V1	V2	E	Meaning	Weight
P1	*	Firewall enable	Netflow > 50	If Netflow > 50, enable the firewall.	40
P2	Firewall enable	Shutdown	TCP error	If TCP error is occured, then shutdown the device.	35
P3	*	Reboot	Zigbee error	If Zigbee error occurs, then reboot the device.	15
P4	Reboot	Deny access	*	If device reboots, then deny any access.	15

FIGURE 4: An example of policy graph.

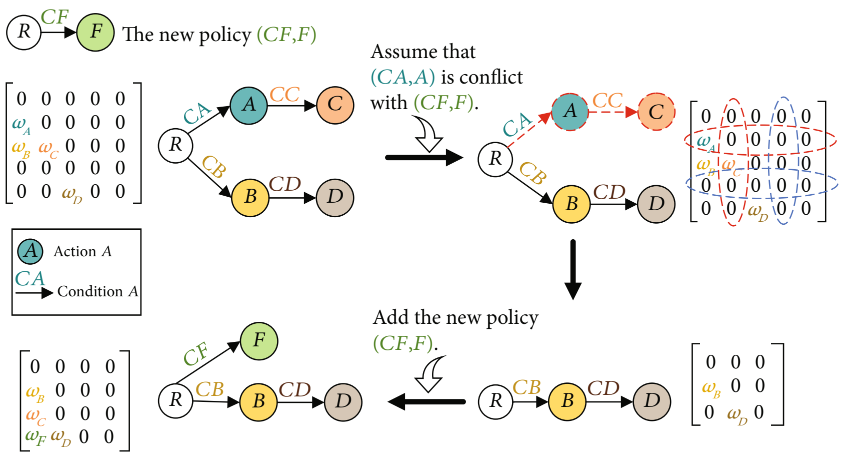

The new policy (CF,F)
Assume that (CA,A) is conflict with (CF,F).
Action A
Condition A
Add the new policy (CF,F).

FIGURE 5: The policy graph reconstruction.

```
            deleteColumn($G, k$); //delete the $k$ column from $G$.
            deleteRow($G,j$);
        end if
    end for
    if deleteColumn($G$, $i$) == TRUE then
        $G[i][j] \longleftarrow 0$; //initial weight is zero.
        $node[i], node[j] \longleftarrow P.event, P.action$;
        $edge[j] \longleftarrow P.condition$; //store the information of $P$.
        return TRUE;
    else
        return FALSE;
    end if
end function
```

6. Evaluation

FPICR is evaluated on enforcing policies over mobile devices in a real Ethereum platform to demonstrate the performance FPICR achieved. We have made a fair comparison with one of the state-of-the-art solutions, XACML framework. The objectives of the evaluation are threefold: (1) testing the performance improvement of FPICR over traditional method; (2) studying the impact of FPICR on the blockchain; (3)

```
Input: Policy graph $G$, new policy $P$
Output: BOOL
    function Check$G$, $P$
      $N \longleftarrow count(G.node)$;
      $TotalIF \longleftarrow 0$;
      for $i$ in $N$ do
          for $j$ in $N$ do
              $TotalIF \longleftarrow G[i][j] + TotalIF$;
          end for
      end for
      $AvgIF \longleftarrow TotalIF/N$;
      $W \longleftarrow G[i][j]$; ; //Assume that $i$, $j$ is conflicted policy.
      if $W > AvgIF$ then
          sendPolicy($P$); //send $P$ to application for resolving conflict.
          return FALSE;
      else
          GraphReconstruction($P$);
          return TRUE;
      end if
    end function
    function GraphReconstruction$P$
      for $k$ in $N$ do
        if $G[i][k]$ ==1 then
            deleteColumn($G$, $k$); //delete the $k$ column from $G$.
            deleteRow($G$,$j$);
        end if
      end for
      if deleteColumn($G$, $i$) == TRUE then
        $G[i][j] \longleftarrow 0$; //initial weight is zero.
        $node[i], node[j] \longleftarrow P.event, P.action$;
        $edge[j] \longleftarrow P.condition$; //store the information of $P$.
        return TRUE;
      else
        return FALSE;
      end if
    end function
```

Algorithm 1. Policy conflict resolution.

providing insights of FPICR's outperforming its peers in BC-IoT.

6.1. Experimental Setup

6.1.1. System Prototype. We build a real fully-equipped BC-IoT rather than a simulated testbed. We deploy five development boards Jetson-TK1s as endpoint devices, and three more servers as the security and mining server. All of these devices are under the same LAN, as shown in Figure 6. Besides, the miner server is set to be full mode, while others are set to be light mode. The security policies are enforced by sending transactions to Ethereum. The blockchain used is Go-ethereum (version 1.7), and the version of EVM we used is 1.7.0. The Ethereum official recommended evaluation framework Truffle is employed to evaluate the performance of blockchain. The hardware configurations in detail are shown in Table 2.

6.1.2. Data Set. We use a policy repository including 10,000 log-based policies and 10,000 XACML-based policies. The logs utilized are HDFS data collected from the real running system for policy template construction. The priority of each log-based policy and each XACML-based policy is assigned one random of the five values ranged from 1 to 5. The weight of each policy is initialized by random values. The data and code used to support the findings of this study have been deposited in the FPICR repository (https://github.com/nkicsl/FPICR).

6.1.3. Measure Metrics. The PbSM in the BC-IoT mainly requires low latency to handle the security problems in real-time, high throughput to process more access requests in a short time, and a small ledger size to save the blockchain storage space. Therefore, to evaluate the performance improvement, we choose the following metrics: (1) the overall performance of FPICR measured by latency, we test the time consumption of policy conflict resolution in FPICR; (2) we then care about the resolution rate and the breakdown latency spent in conflict detection, resolution, and interpretation phases, the throughput for interpreting policies; (3) to further understand FPICR making full use of blockchain,

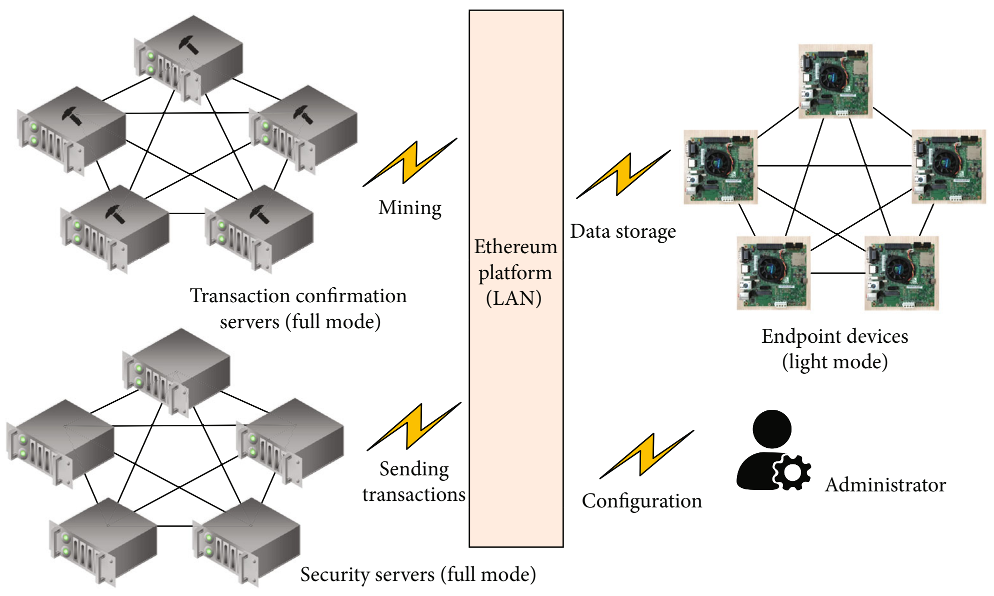

Mining
Transaction confirmation servers (full mode)
Ethereum platform (LAN)
Data storage
Endpoint devices (light mode)
Sending transactions
Configuration
Administrator
Security servers (full mode)

Figure 6: The prototype of FPICR.

Table 2: The hardware configuration in our experiment.

	Endpoint	Security
Hardware	Jetson-TK1	DELL tower
CPU	ARMv7	Xeon E5-2630
Memory	1.9 GB	96,566 MB
OS	Ubuntu 14.04	Ubuntu 16.04

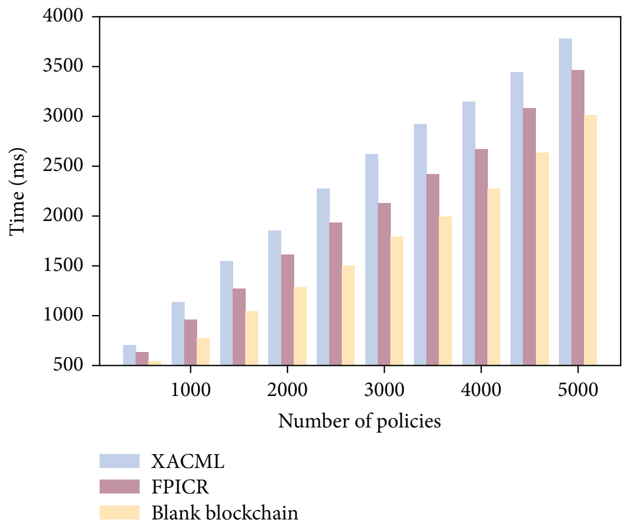


Figure 7: The overall performance of FPICR and XACML.

we also evaluate the resource utilization and overhead of blockchain by ledger size and CPU's overhead; (4) at last, we give a breakdown analysis to explain why FPICR outperforms other PbSM methods in BC-IoT.

6.2. FPICR Overall Performance. To evaluate the overall performance, we select 5,000 policies with 1,500 conflicted ones in the policy repository. So there are 30% of the security conflicted policies in each group. In the experiment, policies number size varies from 500 to 5000 with an increment of five hundred. As a comparison, the control group or blank blockchain group is set to facilitate evaluating the communication overhead of blockchain. Experimental results have been summarized as shown in Figure 7. Compared with XACML, FPICR can achieve latency reduction by 14.1%, even maximum by 18.73%. Therefore, FPICR in the BC-IoT system is outstanding, and FPICR can decrease time consumption from policy interpretation and conflict resolution.

6.3. Resolution Rate

6.3.1. Conflict Resolution Performance. The resolution rate in this paper refers to the percentage of the resolved conflict policies to the total conflict policies. One of the FPICR goals is resolving more policy conflicts automatically according to the system runtime information. High conflict resolution rate means FPICR can resolve more policy conflicts without manual decision. We select 2,000 policies with 400 existing conflict policies to evaluate the resolution rate of FPICR. We perform a comparison between FPICR and the baseline XACML by resolving policy conflicts for 500 times. The results of the resolution rate of two methods achieved are shown in Figure 8. We can observe that FPICR achieves a better policy conflict resolution rate by 91.6%-95.8%, reaching 93.7% on average, while the comparison one is only about 87.6%. It should be noted that though the resolution rate is higher than the comparison by 6%+ on average, FPICR processing speed can also be improved by up to 2.1×.

6.3.2. Latency. Next, we shift our attention to the time consumption FPICR takes to process the same number of policy conflicts. In this case, we measure the breakdown latency spent on the conflict resolution and interpretation phase. In our evaluation, conflict resolution consists of three steps roughly:

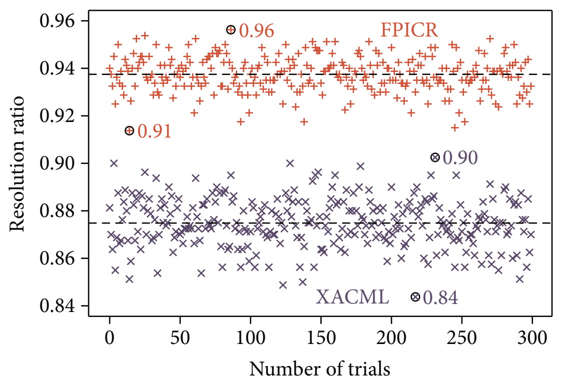


Figure 8: The resolution rate.

(i) *Proposal*. We proposed transactions to the target systems, and each transaction contains one security log-based policy

(ii) *Handling*. The uploaded policy can be detected whether there exist conflicts or not by a specific smart contract. Once the conflict has been detected, it will be resolved and the WDPG reconstruction will be performed. The new WDPG will be also updated into the PGC. If the conflict cannot be handled by FPICR or the comparison presetting static priority, this conflict will be handled manually

(iii) *Submit Manually*. The result for the unresolved conflict will be submitted to the BC-IoT system again. This process can lead to lots of writing operations in the blockchain network, thus, this step is very time-consuming. The worse is that too many conflicts will lead to a large of labor works

Compared with the static priority-based resolution by 15%, 30%, and 45% conflicted policies among the total policies, FPICR can efficiently reduce the latency by 17.44%, 23.93%, and 34.47%, respectively. Because FPICR facilitates solving conflicts by smart contract executing automatically, policy enforcement can regain more time. In contrast, unsolved conflict policy which requires to be handled manually will introduce many labor works; as a result, the time consumption will be greatly increased.

6.4. Interpretation Performance. FPICR aims to achieve fast policy interpretation without missing the accuracy of interpretation results. In this part, we interpret the proposed policy and the typical policy and compare the latency between them. There are two steps for the interpretation phase, the first is policy conversion, and the second is contract deployment. The conversion refers to converting policy into a smart contract. The deployment is to deploy smart contract on the blockchain. We prepare several groups of policies from 300 to 5000 to perform a comparison between FPICR and the baseline XACML. The first metric we are concerned about is conversion latency. We do not care about the deployment time, because it costs the same time in both FPICR and baseline. The comparison results in Figure 9(a) show that compared with XACML, FPICR can speed up by about 2.1× with the obvious advantages. FPICR can fast convert a single log-based policy by only using 0.051 ms on average.

6.4.1. Throughput. Throughput as another metric to evaluate the performance of interpretation. We have also made a comparison between FPICR and XACML to test the number of interpreting policies into smart contract per period. In Figure 9(b), the comparison results about throughput indicate that FPICR can convert and deploy more policies than XACML within 60 seconds. FPICR outperforms the baseline by 1% to 10% along with time. In addition, to further improve throughput, we make full use of a single machine to explore the limitation. We utilize several processes to work together and the results are shown in Figure 9(c). With the increment of the number of processes, the experimental BC-IoT system can enforce dozens of policies per second. When the number of processes increases from 15 to 30, the curve of enforced policy number tends to be gentle. However, when the number is over 30, the curve has begun to drop and flatten fast.

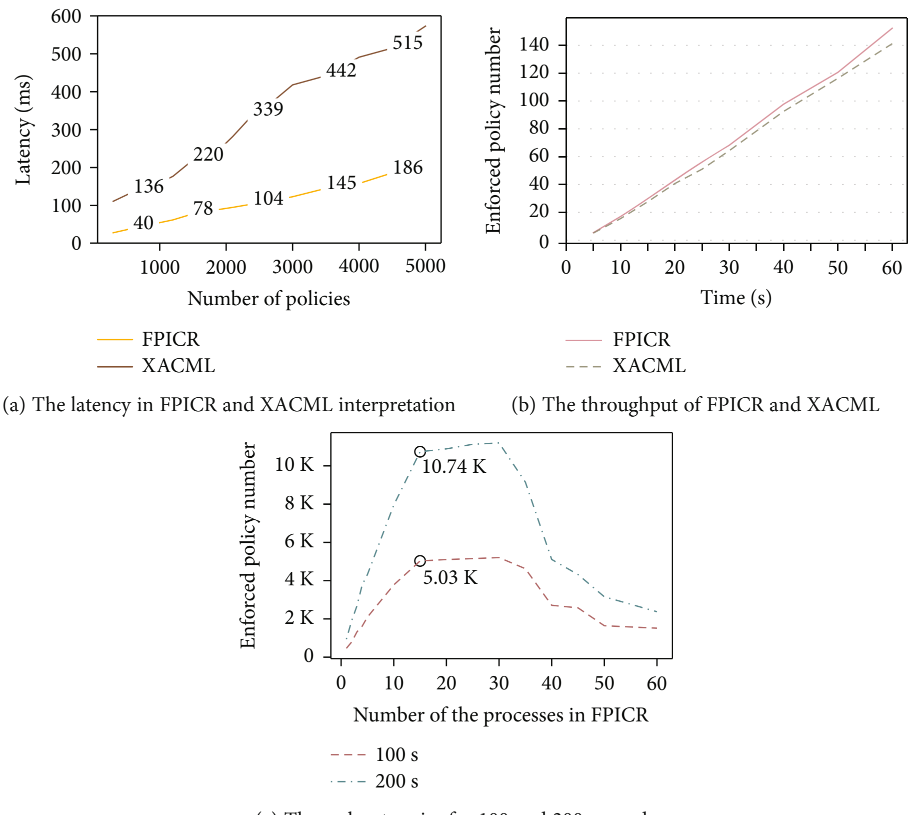
6.5. Blockchain Overhead. We care about the blockchain overhead of BC-IoT to measure the performance of WDPG. The experiments are performed from two aspects: storage reduction and CPU overhead. The reduction of blockchain storage space (or ledger size) is achieved by identifying and cutting the redundant parts brought by the policy dependencies.

6.5.1. Ledger Size. The main index of storage resources about blockchain is ledger size. In our evaluation, ledger size refers to the size of the Ethereum ledger file in runtime. In Figure 10, we can see that the ledger size of blockchain increases with the number of policies growing. Storage structure with WDPG can effectively reduce the size by 16.86% at most than the others.

6.5.2. CPU Overhead. At last, we also have evaluated the CPU overhead during the policy interpretation period. In light of the results in Figure 11, FPICR can interpret and upload 300 policies from 34 s to 110 s. During this period, the peak point of 8.89% represents the CPU occupied by the converting function and the process only lasts for a short time, thereafter, the CPU overhead decreases down to around 3.5%. By contrast, if the ordinary data is uploaded to the blockchain instead of deploying a contract, the overhead of the CPU fluctuates around 4.7%. Obviously, FPICR can effectively reduce CPU overhead during BC-IoT system running. The comparison also indicates that FPICR can be efficient enough to be installed in other real BC-IoT.

6.6. Performance Analysis. This subsection gives a breakdown analysis to explain why FPICR can achieve performance improvement than other PbSM methods in the BC-IoT system.

Because of system runtime information recording in our designed WDPG, FPICR can make full use of the real situation rather than speculation or presetting. The high-resolution rate of FPICR is also benefited from the design,



(a) The latency in FPICR and XACML interpretation
(b) The throughput of FPICR and XACML
(c) Throughput varies for 100 and 200 seconds

FIGURE 9: The performance of interpretation in prototype.

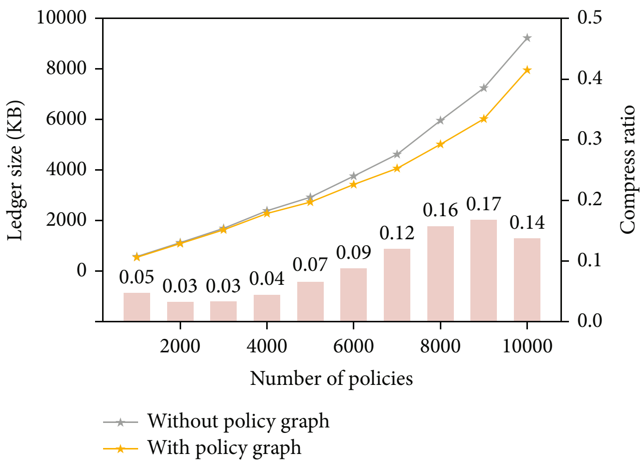


FIGURE 10: The ledger size with policy graph.

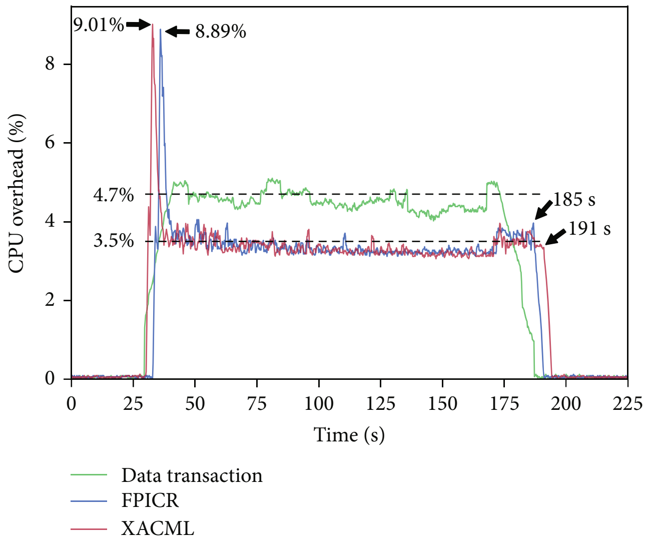
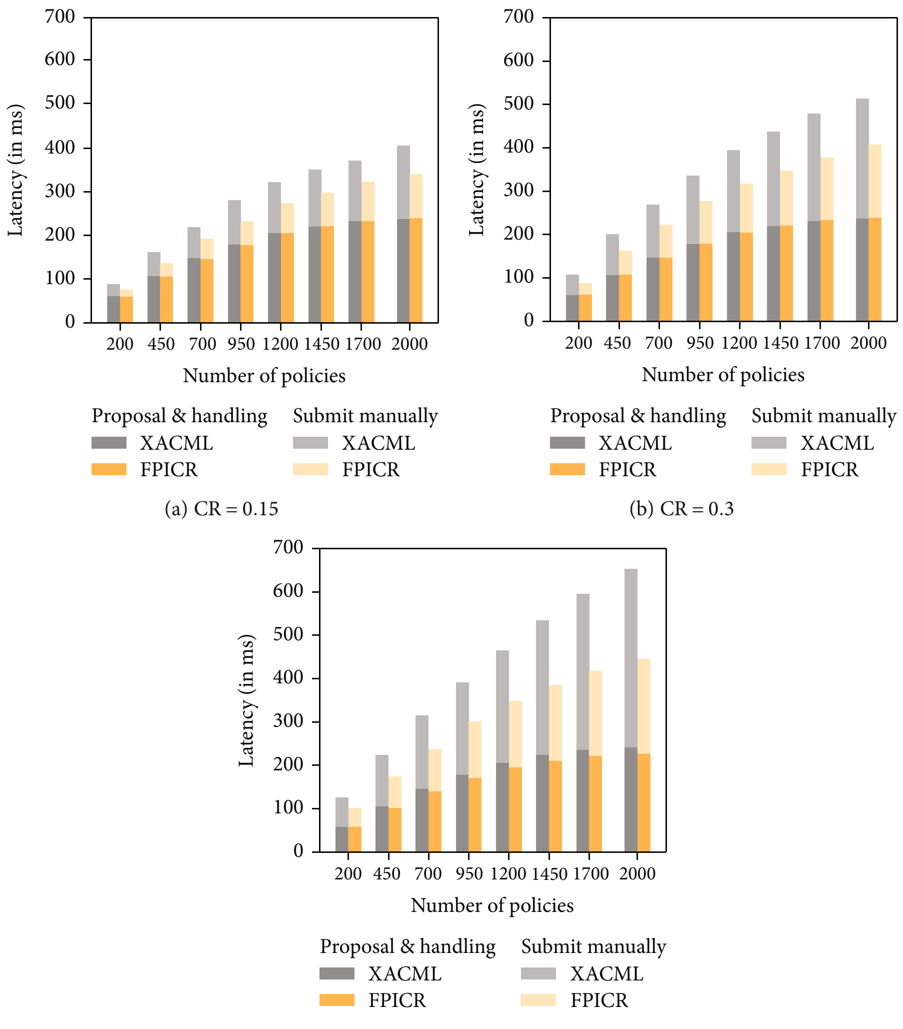


FIGURE 11: The CPU overhead during interpretation.

and more conflicts can be solved automatically, thus, FPICR can decrease heavy labor works. We can find results in the submit manually step in Figure 12, transactions by processing manually will be reduced relative greatly in FPICR.

As shown in Figure 9(a), the low latency metric results benefit from the log-based policy templates. The designed templates can match the smart contract template exactly. This facilitates that FPICR can fill the templates directly, instead of complex parsing. FPICR, in consequence, can only use one step to complete policy interpretation; otherwise, the other methods usually need to take two steps to process. They often require to extract the variables and notations (i.e., "+", ") from the labels and then execute the parsing module. And the parsing is very time-consumption as shown in Figure 9(a). Therefore, our FPICR can meet the requirement of low latency and fast interpretation process very well. These advantages help FPICR achieve high throughput without



(a) CR = 0.15 (b) CR = 0.3 (c) CR = 0.45

Figure 12: The performance of conflict resolution in FPICR.

blocking almost. The reason about storage reduction is that WDPG facilitates policies to only store a simple string rather than redundant uselessly information on blockchain. Therefore, the source data can be reduced and then the ledger size can be decreased. We also find that when the number of policies is increasing large, the compression rate of ledger size can be even further higher, as shown in Figure 10.

For CPU overhead, FPICR CPU resource is only occupied by 4% to 5%, compared with XACML. Because of removing the complex syntactic analysis, FPICR does not need additional computation and can reduce the consumption of CPU. This characteristic enables FPICR to be suitable for computational resource-limited IoT end devices.

7. Conclusion

In this paper, we present FPICR for fast interpretation and dynamic conflict resolution so as to implement effective and efficient PbSM in the BC-IoT system. We propose a new log-based policy interpretation method by extracting parameters directly only in one step. In addition, we present a weighted directed policy graph to organize the relationship for thousands of deployed policies. To solve policy conflict dynamically, the resolution algorithm is proposed based on WDPG reconstruction. FPICR can overcome the limitations of the BC-IoT system and meet the requirements of low latency and compression storage space. Evaluation on FPICR and the comparison results have proved that FPICR can reduce policy interpretation latency and the ledger size of blockchain. Therefore, FPICR can actually realize efficient and economic PbSM for blockchain-based IoT system.

Data Availability

The data and code used to support the findings of this study have been deposited in the FPICR repository (https://github.com/nkicsl/FPICR).

Conflicts of Interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

Acknowledgments

This work is partially supported by the National Key Research and Development Program of China (2018YFB2100300), Zhejiang Lab (2021KF0AB04), the Natural Science Foundation of Tianjin (20JCZDJC00610 and 19JCQNJC00600), and the National Natural Science Foundation (62002175 and 61872200).

References

[1] R. Petrolo, V. Loscri, and N. Mitton, "Towards a smart city based on cloud of things," in *Proceedings of the 2014 ACM international workshop on Wireless and mobile technologies for smart cities*, pp. 61–66, Philadelphia, Pennsylvania, USA, 2014.

[2] Y. He, J. Guo, L. Liu et al., "Iot for the power industry: recent advances and future directions with pavatar," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems, SenSys'18*, pp. 353-354, New York, NY, USA, 2018.

[3] L. Zhao and X. Dong, "An industrial internet of things feature selection method based on potential entropy evaluation criteria," *IEEE Access*, vol. 6, pp. 4608–4617, 2018.

[4] X. Han, L. Wang, S. Xu, D. Zhao, and G. Liu, "Recognizing roles of online illegal gambling participants: an ensemble learning approach," *Computers & Security*, vol. 87, p. 101588, 2019.

[5] A. A. Seif and N. El-Saber, "Scalable distributed-computing iot applied architecture with semantic interoperable gateway," in *Proceedings of the 3rd Africa and Middle East Conference on Software Engineering, AMECSE '17*, pp. 43-44, New York, NY, USA, 2017.

[6] Y. Zhao, Y. Li, X. Zhang, G. Geng, W. Zhang, and Y. Sun, "A survey of networking applications applying the software defined networking concept based on machine learning," *IEEE Access*, vol. 7, pp. 95397–95417, 2019.

[7] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. Maglaras, and H. Janicke, "Blockchain technologies for the internet of things: research issues and challenges," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2188–2204, 2019.

[8] Z. Yang, K. Yang, L. Lei, K. Zheng, and V. C. Leung, "Blockchain-based decentralized trust management in vehicular networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1495–1505, 2019.

[9] V. Scoca, R. B. Uriarte, and R. De Nicola, "Smart contract negotiation in cloud computing," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 592–599, Honolulu, HI, USA, 2017.

[10] Z. Shae and J. Tsai, "Ai blockchain platform for trusting news," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1610–1619, Dallas, TX, USA, 2019.

[11] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2019.

[12] G. Liang, S. R. Weller, F. Luo, J. Zhao, and Z. Y. Dong, "Distributed blockchain-based data protection framework for modern power systems against cyber attacks," *IEEE Transactions on Smart Grid*, vol. 10, no. 3, pp. 3162–3173, 2019.

[13] F. Moradi, A. Sedaghatbaf, S. A. Asadollah, A. Causevic, and M. Sirjani, "On-off attack on a blockchain-based iot system," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1768–1773, Zaragoza, Spain, 2019.

[14] Y. Wang, S. K. Lahiri, S. Chen et al., "Formal verification of workflow policies for smart contracts in azure blockchain," in *Working Conference on Verified Software: Theories, Tools, and Experiments*, pp. 87–106, Springer, 2019.

[15] S. Zhu, W. Li, H. Li, L. Tian, G. Luo, and Z. Cai, "Coin hopping attack in blockchain-based iot," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4614–4626, 2019.

[16] W. Han and C. Lei, "A survey on policy languages in network and security management," *Computer Networks*, vol. 56, no. 1, pp. 477–489, 2012.

[17] V. Varadharajan, K. Karmakar, U. Tupakula, and M. Hitchens, "A policy-based security architecture for software-defined networks," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 897–912, 2019.

[18] A. Lara and B. Ramamurthy, "Opensec: policy-based security using software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 30–42, 2016.

[19] A. Gember, C. Dragga, and A. Akella, "Ecos: leveraging softwaredefined networks to support mobile application offloading," in *2012 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 199–210, Austin, TX, USA, 2012.

[20] A. A. Jabal, M. Davari, E. Bertino et al., "Methods and tools for policy analysis," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–35, 2019.

[21] Z. B. Celik, G. Tan, and P. D. McDaniel, "Iotguard: dynamic enforcement of security and safety policy in commodity iot," in *Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, California, USA, 2019.

[22] L. F. Cranor, "P3p: making privacy policies more useful," *IEEE Security & Privacy*, vol. 1, no. 6, pp. 50–55, 2003.

[23] J. D. Moffett and M. S. Sloman, "Policy conflict analysis in distributed system management," *Journal of Organizational Computing and Electronic Commerce*, vol. 4, no. 1, pp. 1–22, 1994.

[24] H. X. Son and E. Chen, "Towards a fine-grained access control mechanism for privacy protection and policy conflict resolution," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 2, 2019.

[25] X. Du, Z. Lv, J. Wu, C. Wu, and S. Chen, "Pdsdn: a policy-driven sdn controller improving scheme for multi-tenant cloud datacenter environments," in *2016 IEEE International Conference on Services Computing (SCC)*, pp. 387–394, San Francisco, CA, USA, 2016.

[26] N. M. Hoang and H. X. Son, "A dynamic solution for fine-grained policy conflict resolution," in *Proceedings of the 3rd International Conference on Cryptography, Security and Privacy*, pp. 116–120, Kuala Lumpur, Malaysia, 2019.

[27] Y. Kim, J. Nam, T. Park, S. Scott-Hayward, and S. Shin, "Soda: a software-defined security framework for iot environments," *Computer Networks*, vol. 163, p. 106889, 2019.

[28] D. Wenxia, L. Chengyong, W. Ding, and F. Li, "Policy conflict resolution method and apparatus," US Patent 10,193,755, 2019.

[29] A. Molina Zarca, J. B. Bernabe, R. Trapero et al., "Security management architecture for nfv/sdn-aware iot systems,"

IEEE Internet of Things Journal, vol. 6, no. 5, pp. 8005–8020, 2019.

[30] S. Godik and T. Moses, *Oasis Extensible Access Control Markup Language*, OASIS Committee Secification cs-xacml-specification-1.0, 2002.

[31] B. Wu, X.-y. Chen, Y.-f. Zhang, and X.-d. Dai, "An extensible intra access control policy conflict detection algorithm," in *2009 International Conference on Computational Intelligence and Security*, pp. 483–488, Beijing, China, 2009.

[32] M. Charalambides, P. Flegkas, G. Pavlou et al., "Policy conflict analysis for quality of service management," in *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*, pp. 99–108, Stockholm, Sweden, 2005.

[33] C. Shin and W. Woo, *Conflict Resolution Method Utilizing Context History for Context-Aware Applications*, vol. 577, Cognitive Science Research Paper-University Of Sussex Csrp, 2005.

[34] B. Rodrigues, T. Bocek, A. Lareida, D. Hausheer, S. Rafati, and B. Stiller, "A blockchain-based architecture for collaborative ddos mitigation with smart contracts," in *Security of Networks and Services in an All-Connected World*, pp. 16–29, Springer, Cham, 2017.

[35] O. Novo, "Blockchain meets iot: an architecture for scalable access management in iot," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1184–1195, 2018.

[36] O. Alphand, M. Amoretti, T. Claeys et al., "Iotchain: a blockchain security architecture for the internet of things," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, Barcelona, Spain, 2018.

[37] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: the blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE symposium on security and privacy (SP)*, pp. 839–858, San Jose, CA, USA, 2016.

[38] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1285–1298, Dallas, TX, USA, 2017.

[39] J. Bailey, G. Papamarkos, A. Poulovassilis, and P. T. Wood, "An event-condition-action language for xml," in *Web Dynamics*, pp. 223–248, Springer, 2004.