# SWIFTGUARD: Enhanced Privacy and Efficiency in Blockchain-Based Fine-Grained Access Control for Cross-Domain Healthcare Collaboration

Mengke Zhang[†]
*College of Intelligence and Computing*
*Tianjin University*
Tianjin, China
mengkezhangcs@tju.edu.cn

Xiaohong Li
*College of Intelligence and Computing*
*Tianjin University*
Tianjin, China
xiaohongli@tju.edu.cn

Jie Zhang[†]
*College of Intelligence and Computing*
*Tianjin University*
Tianjin, China
jackzhang@tju.edu.cn

Zhe Hou
*School of Electrical Engineering and Computer Science*
*Griffith University*
Nathan, Australia
z.hou@griffith.edu.au

Guangdong Bai
*Faculty of Science and Engineering*
*The University of Queensland*
Brisbane, Australia
g.bai@uq.edu.au

Ruitao Feng[*]
*Faculty of Science and Engineering*
*Southern Cross University*
Bilinga, Australia
ruitao.feng@scu.edu.au

***Abstract*—As healthcare systems evolve and healthcare data grows, the need for cross-domain collaboration treatment has become more complex, necessitating fine-grained access control to enhance privacy and security. Blockchain provides a distributed trusted platform without third parties, but the current blockchain-based access control systems lack efficiency and sufficient privacy protection in cross-domain collaboration. To address these challenges, we propose SWIFTGUARD, an efficient and fine-grained access control system based on a master-slave chain to strengthen the security and privacy of cross-domain healthcare collaboration. SWIFTGUARD incorporates a zero-knowledge proof protocol for cross-domain authentication without exposing sensitive data and leverages quantitative attribute weights for efficient access control. Through game-based security proof, we demonstrate the zero knowledge and soundness of the system. Extensive experiments evaluate that SWIFTGUARD reduces the time complexity of access authorization from $O(n)$ to $O(\log n)$, with improved throughput and stable performance in cross-domain collaboration. Our comprehensive evaluation confirms that SWIFTGUARD provides a secure and efficient access control system for cross-domain healthcare collaboration.**

***Index Terms*—Access control, cross-domain collaboration, blockchain, zero-knowledge proof, smart healthcare**

## I. INTRODUCTION

In the digital era, data has become the cornerstone of advancements across fields such as homecare and healthcare [1]. In particular, the massive electronic health records (EHRs) drive the efficiency and precision of healthcare services. Secure and efficient sharing of EHRs can provide better healthcare services for patients. However, due to disparities in medical capabilities across regions, cross-domain collaboration in healthcare has become an inevitable trend. The security and privacy protection of the EHRs used in such collaboration, however, requires further research [2]. To tackle this, access control is widely applied in healthcare to safeguard EHRs. Attribute-Based Access Control (ABAC) has emerged as an effective fine-grained data control method due to its flexibility in adjusting access rights based on user attributes [3].

In practice, cross-domain collaboration based on ABAC [4]–[6] often rely on trusted third parties or centralized cloud servers for data and policy management, posing trust and security risks [7]. Especially in an untrustworthy environment, the failure or untrustworthiness of the centralized server can lead to data manipulation and leakage. Therefore, there is an urgent need for a decentralized access control system that addresses the trust crisis inherent in traditional systems.

With its decentralized ledger, blockchain offers potential solutions to address the above weakness, which can achieve cross-domain access control without a trusted third party [8]. However, despite the advantages of blockchain, it suffers from inherent performance limitations, particularly when using a single-chain structure for managing ABAC [9]. Researchers have already tried to utilize multi-chain structures to address these limitations [10], but these solutions often conflict with ABAC's need for high security and privacy due to cross-chain operations and the transparency of blockchain [11]. To address these conflicts, we introduce zero-knowledge proof (ZKP), which enables authentication without revealing sensitive information [12]. However, the additional high computing power consumption makes it challenging to implement the enhanced ABAC in cross-domain healthcare collaboration [13].

To address these challenges, we propose SWIFTGUARD, a fine-grained access control system tailored for the secure cross-domain sharing of EHR. SWIFTGUARD adopts a master-

Mengke Zhang and Jie Zhang contribute equally to this work.
Ruitao Feng is the corresponding author.

slave chain architecture to create a trusted platform without the need for a third party, It also implements a cross-domain authentication mechanism based on zero-knowledge proof to protect cross-domain privacy. To relieve the stress of the high delay caused by the ZKP protocol overhead, the system also implements a fast-response access control scheme by quantifying the degree of attribute decision on authorization. Ultimately, SwiftGuard ensures the privacy of cross-domain data access while significantly enhancing system efficiency and security. By Game-based security proof, we proved the zero knowledge and soundness of SwiftGuard. Extensive experiments underscore that SwiftGuard exhibits excellent efficiency in cross-domain collaboration and access control. The specific contributions are as follows:

- We implement a decentralized access control system based on the master-slave chain, which supports efficient cross-domain collaboration and ensures data privacy.
- SwiftGuard introduces a cross-domain collaboration authentication scheme, utilizing the proposed zero-knowledge proof protocol to protect privacy.
- SwiftGuard reduces authorization complexity from $O(n)$ to $O(\log n)$ by quantitative attribute decision-making ability and dynamic policy structure.

Therefore, SwiftGuard offers a novel solution for cross-domain data access control in healthcare, enabling robust data protection and efficient collaboration.

## II. Related Work

Attribute-based access control [3] offers fine-grained control in healthcare settings by leveraging complex contextual information and user attributes. However, traditional cross-domain implementations in ABAC typically rely on trusted third parties to authenticate cross-domain requests and make access decisions, leading to privacy and security risks. Bai et al. [4] developed a cross-domain ABAC model based on the attribute mapping center, where a trusted third party is required to transform attribute certificates during cross-domain authentication. While the addition of attribute certificates and mapping tables improves cross-domain efficiency, the reliance on a third party introduces attribute leakage and unauthorized access risks. Oliveira et al. [5] proposed AC-ABAC for healthcare and acute care, using cloud storage services instead of the attribute mapping center to manage EHR access. However, cloud storage is not entirely trustworthy and may expose data to leakage risks due to its lack of reliable cross-domain collaboration policies and authentication schemes.

Blockchain's decentralization and distributed ledger characteristics provide new ideas for many researchers. Liu et al. [14] and Damiano et al. [15] explored blockchain-based ABAC models to improve security and address issues of untrusted third parties. However, the single-chain architectures used in these studies still face scalability and storage efficiency challenges for large-scale data. This has sparked interest in multi-chain solutions. Zhang et al. [10] utilized a master-slave chain to store data in slave chains and record secure indexes on the master chain, thereby strengthening the potential for decentralized EHR management. Furthermore, considering the security and autonomy requirements in multi-domain IoT collaboration, Zhang et al. [9] proposed a master-slave chain-based decentralized cross-domain access control model. In this model, slave chains handle intra-domain access control, while the master chain facilitates cross-domain access, preserving each domain's independence and performance. However, they do not have a reliable design for cross-domain collaboration schemes. Due to blockchain's transparency, each cross-domain request and authentication process is publicly visible, creating a risk that adversaries could infer sensitive details. Wu et al. [11] proposed a new access control scheme using zero-knowledge proof (ZKP) with smart contracts to hide attributes or policies in authorization decisions. However, they did not address cross-domain collaboration and sharing. Additionally, despite offering privacy benefits, integrating ZKP increases computational complexity [12], potentially causing processing delays in time-sensitive settings like healthcare, where dynamic access efficiency is essential.

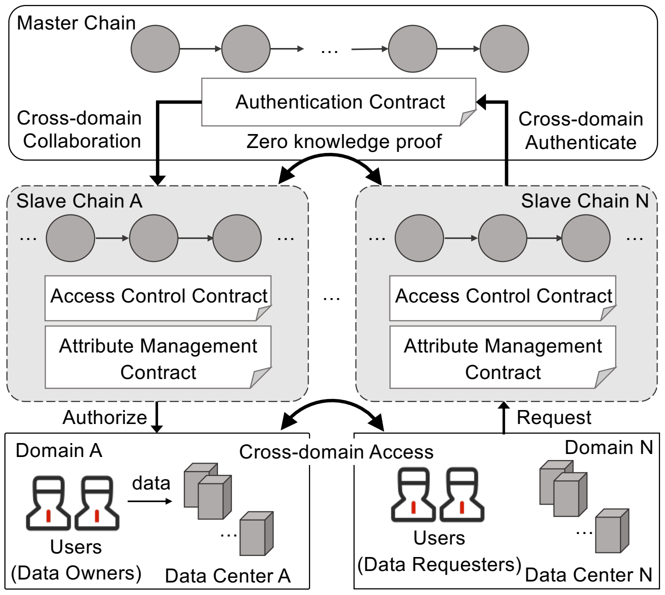


Fig. 1. System model of SwiftGuard

Current healthcare access control systems for cross-domain collaboration face key limitations. First, ensuring cross-domain privacy and securing data in untrusted environments is challenging. Second, many systems struggle to effectively manage cross-domain requests and data attributes, especially in ABAC during dynamic authorizations. Lastly, efforts to enhance security often reduce the efficiency of the authorization process.

## III. Proposed Model

### A. System Model

Fig. 1 outlines the key components of SwiftGuard, tailored for cross-domain access control in healthcare:

- **Data Owner (DO)**: DO owns patient data in the data center. It defines and manages access policies to ensure access is limited to authorized entities.

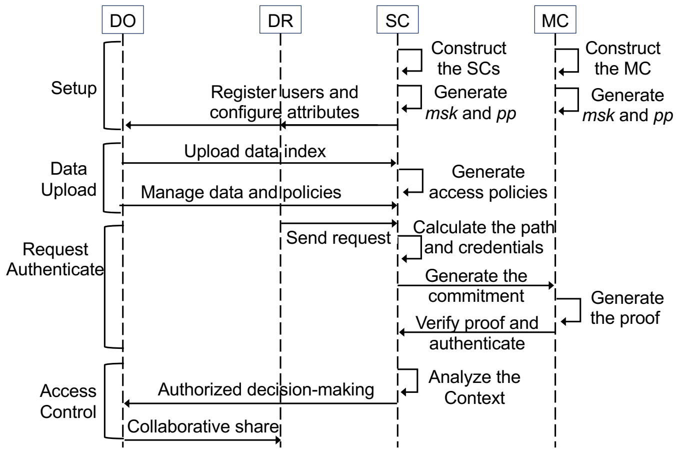


Fig. 2. The main workflow of SwiftGuard

- **Data Requester (DR)**: DR interacts with the SC to send data access or cross-domain collaboration requests, rather than directly interacting with DO.
- **Slave Chain (SC)**: Each domain maintains an autonomous SC, which ensures data security within the domain. It enforces attribute management and authentication for all DOs and DRs via smart contracts and executes intra-domain access control policies.
- **Master Chain (MC)**: MC coordinates the interaction between SCs, handling the authentication of collaborators, protecting sensitive information with zero-knowledge credentials, and issuing tokens for verified requests.

### B. Design Goals

Our design goals encompass the following key aspects:

**G1**. Ensure secure authentication schemes for cross-domain collaboration to ensure secure collaboration without compromising the security of any involved parties.

**G2**. Protect the privacy of sensitive information even during cross-domain interactions, mitigating risks related to unauthorized access or data leakage.

**G3**. Achieve efficiency and fine-grained access control that adapts to various roles, scenarios, and data sensitivity levels while maintaining high system performance.

## IV. Design of SwiftGuard

### A. Overview

Fig. 2 briefly illustrates the workflow of SwiftGuard, which comprises the following four phases.

*1) Setup:* This phase establishes the master-slave chain architecture and the basic settings of SwiftGuard based on Hyperledger Fabric. Firstly, the master chain $MC$ is initialized with $\{node_i\}_{i=1}^{M}$. The slave chain $SC_j$ for domain $D_j$ is constructed with $\{node_k\}_{k=1}^{N_j}$ and connected to $MC$. Secondly, SwiftGuard generates the master system key $msk$ and public parameters $pp$ by **Setup** and **KeyGen** functions. Finally, all users complete registration and attribute configuration via the Attribute Management Contract (AMC).

*2) Data Upload:* In this phase, DO calculates a secure index $eid = [H(metadata)]_{pk}$ based on the $metadata$ stored in the data center and uploads $eid$ to SC. Then SC initializes access policies through the contract AMC. DO can dynamically manage these policies following access requirements.

*3) Request Authenticate:* DR in $SC_i$ sends requests to access DO's data or initiates cross-domain requests to $SC_j$ when collaborative treatment is required. The request is structured as $req = \{uid, type, eid, SC_i, SC_j, \overrightarrow{context}\}$. Upon receiving $req$, $SC_i$ verifies it via the Access Control Contract (ACC), generating a $path$ based on $uid$'s attributes and $req$'s context (e.g., timestamp, environment). For cross-domain requests, $SC_i$ and $SC_j$ compute cross-domain credentials $cre_i$ and $cre_j$. Based on credentials, $SC_J$ calculates commitments to conceal intra-domain sensitive information, generates a proof $\pi$ by $MC$'s Authentication Contract (AC), and verifies the validity of the proof by $SC_i$ to complete authentication.

*4) Access Control:* In this phase, $SC_i$ conducts a context-aware analysis and judges the existence of $path$ in the policy by the contract ACC, dynamically making authorization decisions and issuing an access token. This token enables DR to access DO's data securely upon receiving authorization.

### B. Smart Contract Design

This section introduces the system's specific implementation of three smart contracts. To clearly illustrate the core operating logic, we additionally describe the functions involving complex calculations or multiple steps in algorithms 1.

*1) Attribute Management Contract (AMC):* In SwiftGuard, users and policies are composed of attribute tuples. To achieve modular management, we jointly design all operations involving attributes as AMC.

- $\textbf{AttrConfig}() \rightarrow A$: It configures all attribute parameters in the system and outputs the attribute set $A$.
- $\textbf{UserGen}() \rightarrow \overrightarrow{attr_{uid}}$: It registers users, assigns the attribute tuple $\overrightarrow{attr_{uid}} = (attr_1, \cdots, attr_n)$ to $user_i$.
- $\textbf{DataUpload}(ehr) \rightarrow eid$: When DO uploads $ehr$, this function runs $\textbf{hash}(ehr)$ and uses $pk$ to encrypt and get $eid$.
- $\textbf{InitPolicy}(eid) \rightarrow pid$: It initializes the policy $p$ of $eid$, which is composed of multiple attribute tuples and defined as $P = \{\overrightarrow{attr_{rule1}} \vee \cdots \vee \overrightarrow{attr_{rulen}} | \overrightarrow{attr_{rulei}} \subseteq A\}$, where the same $attr_k$ are shared in $\overrightarrow{attr_{rulei}}$ and the order of $attr_k$ is determined by attribute weights. Finally, it returns $pid$.
- $\textbf{AWCal}() \rightarrow W$: It regularly calculates the attribute weights list $W$. The calculation formula is as follows.

$$W = sort(-\sum_{attr_i \in A} P(X = attr_i)\log(P(X = attr_i))) \quad (1)$$

*2) Access Control Contract (ACC):* ACC is responsible for processing all access control requests and verifying permissions based on attributes and policies.

- $\textbf{VeriReq}(req) \rightarrow T/F$: It verifies whether the request received is legal. If it is legitimate, it returns true and proceeds to the next step, otherwise, it terminates the request.
- $\textbf{GetPath}(req) \rightarrow path$: Based on DR's attributes and the $req$'s context, it generates a $path = (\overrightarrow{attr_{DR}}, \overrightarrow{context_{req}})$.

**Algorithm 1** Smart Contract

```
    //Quantify the attribute decision-making and sort it as $W$
 1: function AWCal( )
 2:     Initialize an empty entropy list $entro_v$
 3:     for each $attr_i$ in $A$ do
 4:         Calculate the $entropy_i$ for $attr_i$ as formula 1
 5:         $entro_v \leftarrow (attr_i, entropy_i)$
 6:     end for
 7:     return $W \leftarrow sort(entro_v)$
 8: end function
    //Restructure the policy into a tree structure
 9: function Recon($P, W$)
10:     Initialize an empty $root$ to point to $P_W$
11:     for each rule $\overrightarrow{attr_{rule}}$ in $P$ do
12:         sort $\overrightarrow{attr_{rule}}$ based on $W$
13:         if not exist $root.child[attr_i \in attr_{rule}]$ then
14:             create $root.child[attr_i]$
15:         end if
16:     end for
17:     return $P_W$
18: end function
    //Check whether there is a RoottoLeaf path in $P_W$
19: function Access($path, P_W$)
20:     $root \leftarrow P_W$, $attr_i \leftarrow path.attr_1$
21:     for $root! = null$ && $attr_i! = null$ do
22:         if not exist $root.child[attr_i]$ then return $\perp$
23:         end if
24:         $root \leftarrow root.child[attr_i], attr_i \leftarrow attr_{i+1}$
25:     end for
26:     Generate the $token$
27:     return $token$
28: end function
```

- $\mathbf{Recon}(P, W) \rightarrow P_W$: It reconstructs policies $P$ into $P_W$ according to attribute weights in $W$. For $attr_i$, the higher weight $aw_i$ reflects more decision-making in restricting access, so it prioritizes matching to speed up the authorization.
- $\mathbf{Access}(path, P_W) \rightarrow token/\perp$: It is the core authorization function and reduces the time complexity of authorization from $O(n)$ to $O(\log n)$, formally expressed as the existence judgment of $path$ in $P_w$ shown in the Algorithm 1.
- $\mathbf{CreCal}(uid) \rightarrow cre$: It calculates the cross-domain collaboration credential $cre$ based on the formula $cre = \frac{C_s}{C_t} * \rho_1 + \frac{A_d}{A_t} * \rho_2 + CP * \rho_3$. $C_t$ is the number of requests initiated and participated in, and $C_s$ is the successful collaborations number. $A_t$ and $A_d$ denote the number of access requests and authorization times. $CP$ is computing power and $\rho_1$, $\rho_2$, $\rho_3$ are the weights assigned to the three parts and the total is 1.

*3) Authentication Contract (AC):* AC implements the cross-domain authentication scheme based on zero-knowledge proof.

- $\mathbf{Setup}(\lambda) \rightarrow pp$: Given a security parameter $\lambda$, it chooses two large prime numbers $p, q$ and computes $N = pq$ to satisfy $\lambda = lcm(p-1, q-1)/2$ and $\mathcal{L}(p) = \mathcal{L}(q) = \lambda$, where $\mathcal{L}(x) = \frac{x-1}{N}$. Then it randomly selects $\theta \in [1, N/4]$, generates a generator $g$ of order $(p-1)(q-1)$ and computes $h = g^{\theta} modN^2$. $\mathbb{Z}_p$ is a modulo $p$ residual class ring.
- $\mathbf{KeyGen}(\lambda) \rightarrow (pk, sk)$: It randomly selects $\theta_i \in [1, N/4]$ as $sk$ and generates $pk = (N, g, g^{\theta_1} modN^2)$.
- $\mathbf{Commitment}(m) \rightarrow y$: Given a cross-domain credential $m$ known to the prover $P$, the output is the commitment triple $y$. The function computes $\mathbf{M} = (M_1, \dots, M_n) \in \{0,1\}^n$ based on $m$, satisfying $\langle \mathbf{M}, \mathbf{2}^n \rangle = 2^m$, where $[a, n]$ is the valid scope of the of credentials. Let $\mathbf{s} = \mathbf{M}_{[a,n]}$ and $\mathbf{m} = \mathbf{M}_{[0,m+1]}$. $P$ generates random numbers $\alpha, \beta, \gamma, \delta, \varepsilon, x \in \mathbb{Z}_p$, and computes $T = g^m h^{\varepsilon}$, $C_1 = g^{\alpha} h^{\beta}$, $C_2 = g^{\langle \mathbf{s}, \mathbf{x}^{n-a} \rangle} h^{\gamma}$, and $C_3 = g^{\langle \mathbf{m}, \mathbf{x}^{(m+1)} \rangle} h^{\delta}$. Finally, it outputs $y = (C_1, C_2, C_3)$.
- $\mathbf{Proof}(y, c) \rightarrow (y, \pi)$: The verifier $V$ gives a challenge $c$, and $P$ constructs a proof $(y, \pi)$ according to the challenge $c$ and commitment triple $y$. It calculates $z_1 = \alpha + cm$, $z_2 = \beta + c\varepsilon$, $z_3 = g^{(c-1)\langle \mathbf{m}, \mathbf{x}^{m+1} \rangle} h^{\gamma}$ and $z_4 = g^{c(x^a - 1)\langle \mathbf{s}, \mathbf{x}^{n-a} \rangle} h^{\delta}$. Let $\pi = (z_1, z_2, z_3, z_4)$ and the final proof is $(y, \pi)$.
- $\mathbf{Verify}(y, \pi) \rightarrow 0/1$: Upon receiving the proof $(y, \pi)$, $V$ determines the validity of Equation 2 to ensure $m$ is not tampered with and checks Equation 3 to ensure $m$ meets the range of $[a, n]$. If both Equation 2 and Equation 3 are satisfied, the output is 1, indicating successful cross-domain authentication; otherwise, it is 0, signifying failure.

$$g^{z_1} \cdot h^{z_2} = T^e \cdot C_1 \tag{2}$$

$$C_2^c \cdot z_4 = C_3 \cdot z_3 \tag{3}$$

- $\mathbf{ColToken}(req) \rightarrow token$: After completing the function **Verify**, call this function, which authorizes access by calling the in-domain policy and generates access tokens.

## V. Security And Performance Analysis

### A. Security Proof

*Theorem 1 (Zero-knowledge):* The proposed authentication scheme satisfies zero-knowledge if the Decision Diffie-Hellman (DDH) assumption holds [16].

*Proof.* We will prove the theorem by constructing a simulator $\mathcal{S}$ based on a DDH problem instance $(g, g^{\varepsilon_0}, g^{\varepsilon_1}, Z)$ so that $\mathcal{A}$ cannot distinguish the witness from $\mathcal{S}$. If $\mathcal{A}$ can distinguish the knowledge $m_0$, $m_1$ from $(y, \pi)$ with a non-negligible advantage $\varepsilon$ so that $\mathcal{S}$ can break the DDH problem with a non-negligible probability.

**Setup.** $\mathcal{S}$ generates the system public parameter $pp$ by running the function $\mathbf{Setup}(\lambda)$.

**Query.** $\mathcal{A}$ submits two distinct knowledge $m_0, m_1$ to $\mathcal{S}$. $\mathcal{S}$ flips a random coin $b \in \{0,1\}$ and generates $y_b = (C_1', C_2', C_3')$ by running $\mathbf{Commitement}(m_b)$. Then $\mathcal{S}$ calculates $\pi_b$ based the challenge $c$, and sends $(y_b, \pi_b)$ to $\mathcal{A}$.

**Guess.** When the adversary $\mathcal{A}$ receives the pair $(y_b, \pi_b)$, it guesses a bit $b'$ and sends to the simulator $\mathcal{S}$. If $b' = b$ then $\mathcal{A}$ wins the game, otherwise it fails.

If $b' = b$, the adversary $\mathcal{A}$ gets the knowledge $m_{b'}$ and knows $T = g^{m_{b'}} h^{\varepsilon}$ from the pair $(y_b, \pi_b)$. Therefore, $\mathcal{A}$ can get $Z = g^{\varepsilon_0 \varepsilon_1} = h^{\varepsilon}$ and solve the DDH problem. The simulation is indistinguishable from the real attack. Thus, $\mathcal{A}$ has probability $\frac{1}{2} + \varepsilon$ of guessing correctly.

If $b' \neq b$, $\mathcal{A}$ only knows $T \neq g^{m_{b'}} h^{\varepsilon}$ and $Z \neq g^{\varepsilon_0 \varepsilon_1}$. It is equivalent to using a random number $Z$ independent of other parameters, without gaining any additional information. Thus, the probability that $\mathcal{A}$ guesses correctly is $\frac{1}{2}$.

Therefore, the probability that $\mathcal{A}$ solves the DDH problem is $\frac{1}{2}(\frac{1}{2}+\varepsilon+\frac{1}{2}) = \frac{1}{2}+\frac{1}{2}\varepsilon$. It is contrary to the DDH assumption.

*Theorem 2 (Soundness):* The proposed authentication scheme satisfies soundness if the Discrete logarithm assumption (DLA) holds [16].

*Proof.* We prove this theorem by showing the probability that $\mathcal{A}$ prevailing in a series of games against $\Omega$ is less than $negl(\lambda)$, as determined through the use of DLA. We construct an extractor $\Omega$ to fight against $\mathcal{A}$ and denote $Pr\left[Win_i\right]$ as the probability of $\mathcal{A}$ winning in the $Game_i$.

$\mathbf{Game_0}$. Suppose an attacker $\mathcal{A}$ attempts to forge a legitimate proof $(y_1, \pi_1)$ to deceive the $V$, while an extractor $\Omega$ endeavors to extract a valid witness from a fake proof. Initially, $\Omega$ receives $g^m h^{\varepsilon}$ and $\mathcal{A}$ generates $y_1 = (C_1, C_2, C_3)$. $\Omega$ generates a random challenge $c$ and $\mathcal{A}$ responds with a fake proof $(y_1, \pi_1)$ based on $c$. $\Omega$ verifies that equation 3 is not satisfied, then $\Omega$ rewinds to the challenge phase and sends a new challenge $c'$ to generate new $\pi'$. So $\Omega$ can calculate:

$$\begin{cases} m = \dfrac{z_1 - z_1'}{c - c'} \\ \varepsilon = \dfrac{z_2 - z_2'}{c - c'} \end{cases} \tag{4}$$

If $\mathcal{A}$ wins $Game_0$, then the discrete logarithm can be computed, which is contrary to the DLA. Thus we can get

$$Pr[Win_0] \leqslant negl(\lambda) \tag{5}$$

$\mathbf{Game}_1$. Suppose $\mathcal{A}$ attempts to forge a secret value $m_1$ as input to complete the proof. In the commitment phase, $\mathcal{A}$ chooses $\alpha_1$,$\beta_1$,$\gamma_1$,$\delta_1$,$\varepsilon_1$,$x_1 \in \mathbb{Z}_p$ randomly to generate

$$y_1 = \left(g^{\alpha_1} h^{\beta_1}, g^{\langle \mathbf{s_1}, \mathbf{x_1}^{n-a} \rangle} h^{\gamma}, g^{\langle \mathbf{m_1}, \mathbf{x_1}^{(m_1+1)} \rangle} h^{\delta}\right) \tag{6}$$

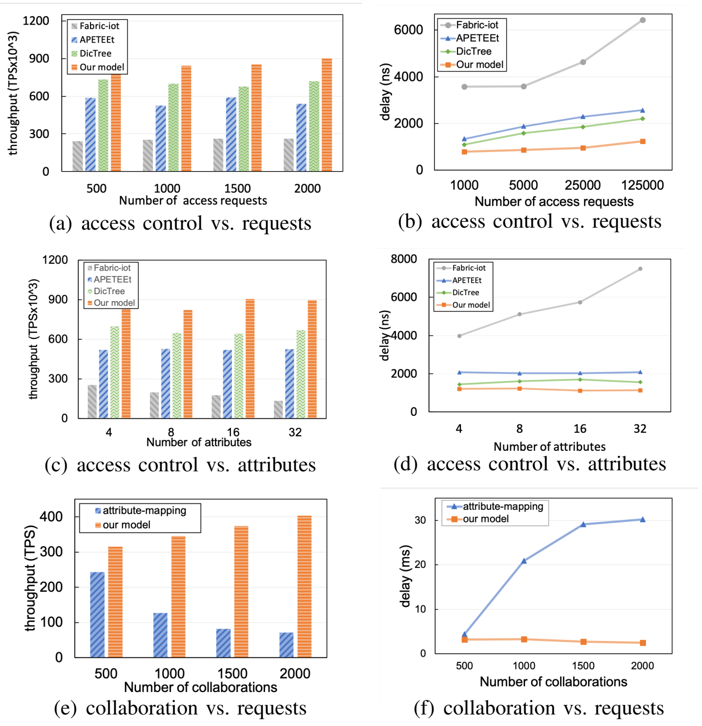
$\Omega$ generates a random challenge $c$ and $\mathcal{A}$ generates $z_1 = \alpha_1 + cm_1$, $z_2 = \beta_1 + c\varepsilon_1$, $z_3 = g^{(c-1)\langle \mathbf{m_1}, \mathbf{x_1}^{m_1+1} \rangle} h^{\gamma_1}$ and $z_4 = g^{c(x_1^a - 1)\langle \mathbf{s_1}, \mathbf{x_1}^{n-a} \rangle} h^{\delta_1}$, along with generating the proof $(y_1, \pi_1')$. $\Omega$ verifies that Equation 2 does not hold, then rewinds to the challenge phase and sends a new challenge $c'$. Similar to $\mathbf{Game}_0$, $\mathcal{A}$ generates $\pi_1' = (z_1', z_2', z_3', z_4')$ and returns a new proof $(y_1, \pi_1')$. So $\Omega$ can calculate

$$\begin{cases} m = \dfrac{z_1 - z_1'}{c - c'} \\ \varepsilon = \dfrac{z_2 - z_2'}{c - c'} \end{cases} \tag{7}$$

Therefore, if $\mathcal{A}$ can win $\mathbf{Game}_1$ with a non-negligible advantage over $\mathbf{Game}_0$, then $\Omega$ can use it to break the DLA. Thus we can get

$$Pr[Win_1] - Pr[Win_0] \leqslant negl(\lambda) \tag{8}$$

In conclusion, we prove that $\mathcal{A}$ cannot forge legitimate evidence to deceive $V$, establishing the soundness of the proposed protocol. The proof concludes here.



(a) access control vs. requests (b) access control vs. requests
(c) access control vs. attributes (d) access control vs. attributes
(e) collaboration vs. requests (f) collaboration vs. requests

Fig. 3. Evaluation of SwiftGuard for throughput and delay

## B. Performance Analysis

SwiftGuard is implemented using Hyperledger Fabric v2.2, Docker v20.10, Docker-compose v1.29, and GoLang v1.18. All experiments are conducted on an Apple M1 Pro machine with 16GB memory running on MacOS Monterey v12.3.1. To evaluate the design goals of SwiftGuard, we conducted the following experiments, each repeated ten times, and the average was the final result.

Firstly, we evaluate throughput and delay on the system's access control based on the number of requests and attributes, comparing the results with Fabric-iot [14], APETEEt [17] and DicTree. Fabric-iot [14] is an ABAC model implemented on the Hyperledger Fabric to evaluate ABAC performance. APETEEt [17] enforces ABAC policies using a height-balanced tree deployed in a trusted environment to improve efficiency. DicTree uses dictionary trees to replace the policy structure in our model to explore the effectiveness of updating attributes in SwiftGuard. Fig. 3(a) illustrates the impact of increasing request number on throughput. SwiftGuard demonstrates a significant advantage in off-chain authorization, with higher throughput to handle more requests than other baselines. As shown in Fig. 3(b), the delay of Fabric-iot [14] increases exponentially as the number of requests grows, indicating a time complexity of $O(n)$. APETEEt [17], DicTree and our model's delay remain nearly linear as the number of requests increases, among which our model has the best performance. This indicates that our system effectively reduces time complexity from $O(n)$ to $O(\log n)$. Fig. 3(c) shows that as the attributes' number exponentially grows, the performance gap in throughput between the models becomes

TABLE I
OUR PROTOCOL VS. BULLETPROOFS

| method | time (ms/op) | byte (kB/ op) | mem alloc (alloc/op) |
|---|---|---|---|
| Our protocol | 2,196.45 | 56.83 | 1,571.7 |
| Bulletproofs [18] | 19,031.93 | 303.51 | 8,329.5 |

TABLE II
STEPS EVALUATION

|  | UserGen | DataUpload | VeriReq | Recon |
|---|---|---|---|---|
| throughput(TPS) | 61.282 | 695.787 | 764772.788 | 34141.82 |
| delay(ms) | 16358.7347 | 1497.72129 | 1.8853 | 89.8737 |

more pronounced. Fig. 3(d) shows that attribute exponential growth has minimal effect on SWIFTGUARD's response speed.

Second, we also compared cross-domain collaboration requests with Bai's cross-domain access control scheme relies on the attribute mapping center based on a trusted third party [4]. To assess the advantages of SWIFTGUARD, we replicated Bai's cross-domain scheme [4] within our architecture, maintaining consistent system settings. Fig. 3(e) illustrates that SWIFTGUARD's throughput significantly increases as the number of concurrent collaborations grows while Bai's [4] decreases. This indicates that SWIFTGUARD's advantage in handling more transactions with greater privacy and security. As shown in Fig. 3(f), the delay of SWIFTGUARD remains almost constant and slightly decreases, attributed to the fixed operational scale in SWIFTGUARD's cross-domain process. While Bai's scheme [4] experiences increased delay as the number of collaborations increases, due to the frequent attribute mappings and certificate transformations required.

Additionally, to evaluate our proposed proof protocol, we compared it with Bulletproofs [18] in terms of time, bytes, and memory allocation. The result is shown in Table I and indicates that our protocol processes perform better.

Finally, to ensure the system's usability, we progressively evaluate SWIFTGUARD the cost (throughput and delay) outside the core function. The averaged result is shown in Table II. Since the functions **UserGen** and **DataUpload** involve frequent database operations, we believe that the overhead is reasonable and acceptable. In the request authenticate phase, we evaluated **VeriReq** separately. We tested the **Recon** used for policy reconstruction in the access control phase.

Our comprehensive evaluation concludes that SWIFTGUARD performs well and can meet the needs for cross-domain collaboration in healthcare.

## VI. CONCLUSION

This paper proposed SWIFTGUARD, an attribute-based access control model designed to enhance privacy and security in cross-domain access control. By leveraging a master-slave blockchain architecture and integrating a zero-knowledge proof protocol, SWIFTGUARD enables secure, privacy-preserving cross-domain authentication without exposing sensitive information. To achieve efficient access control, it also quantifies the degree of decision-making in multi-attribute access control and implements efficient authorization to reduce the time complexity from $O(n)$ to $O(\log n)$. Through security analysis, we prove the security zero-knowledge and soundness in SWIFTGUARD. By implementing SWIFTGUARD on Hyperledger Fabric and comparing it with other existing schemes, we demonstrate its performance excellence. In future work, we aim to test it in a large-scale production environment and conduct a thorough security analysis of potential attacks.

## ACKNOWLEDGEMENT

This work is supported by the National Key Research and Development Program under Grant 2021YFF1201102.

## REFERENCES

[1] M. Sookhak, M. R. J. Sattari, N. S. Safa, and F. R. Yu, "Blockchain and smart contract for access control in healthcare: A survey, issues and challenges, and open issues," *J. Netw. Comput. Appl.*, vol. 178, p. 102950, 2021.

[2] S. K. Memon, N. I. Sarkar, A. Al-Anbuky, and M. A. Hossain, "Preemptive admission control mechanism for strict qos guarantee to life-saving emergency traffic in wireless lans," *J. Netw. Comput. Appl.*, vol. 199, p. 103318, 2021.

[3] V. C. Hu, D. F. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to attribute based access control (abac) definition and considerations," 2014.

[4] L. Bai, K. Fan, Y. Bai, X. Cheng, H. Li, and Y. Yang, "Cross-domain access control based on trusted third-party and attribute mapping center," *Journal of Systems Architecture*, vol. 116, p. 101957, 2021.

[5] M. T. de Oliveira, G. Verginadis, L. H. A. Reis, E. Psarra, I. Patiniotakis, and S. D. Olabarriaga, "Ac-abac: Attribute-based access control for electronic medical records during acute care," *Expert Syst. Appl.*, vol. 213, p. 119271, 2022.

[6] M. A. Lail, M. Moncivais, R. Benton, and A. J. Perez, "Cloud-based access control including time and location," *Electronics*, 2024.

[7] Y. Zhang, M. Yutaka, M. Sasabe, and S. Kasahara, "Attribute-based access control for smart cities: A smart-contract-driven framework," *IEEE Internet of Things Journal*, vol. 8, pp. 6372–6384, 2020.

[8] W. Viriyasitavat and D. Hoonsopon, "Blockchain characteristics and consensus in modern business processes," *J. Ind. Inf. Integr.*, vol. 13, pp. 32–39, 2019.

[9] Z. Zhang, X. li Wu, and S. Wei, "Cross-domain access control model in industrial iot environment," *Applied Sciences*, 2023.

[10] A. Zhang and X. Lin, "Towards secure and privacy-preserving data sharing in e-health systems via consortium blockchain," *Journal of Medical Systems*, vol. 42, pp. 1–18, 2018.

[11] N. Wu, L. Xu, and L. Zhu, "A blockchain based access control scheme with hidden policy and attribute," *Future Gener. Comput. Syst.*, vol. 141, pp. 186–196, 2022.

[12] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie, and X. Peng, "A survey on zero-knowledge proof in blockchain," *IEEE Network*, vol. 35, pp. 198–205, 2021.

[13] A. E. Mensah, S. Zhou, Y. Liao, E. Antwi-Boasiako, and I. A. Obiri, "Authentication scheme based on non-interactive zero-knowledge proof for mobile health," *2022 IEEE 24th Int Conf on High Performance Computing & Communications*, pp. 1690–1696, 2022.

[14] H. Liu, D. Han, and D. Li, "Fabric-iot: A blockchain-based access control system in iot," *IEEE Access*, vol. 8, pp. 18 207–18 218, 2020.

[15] D. di Francesco Maesa, P. Mori, and L. Ricci, "A blockchain based approach for the definition of auditable access control systems," *Comput. Secur.*, vol. 84, pp. 93–119, 2019.

[16] J. Duan, L. Wang, W. Wang, and L. Gu, "Trct: A traceable anonymous transaction protocol for blockchain," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 4391–4405, 2023.

[17] P. Godhani, R. Bharadhwaj, and S. Sural, "Poster: Apeteet – secure enforcement of abac policies using trusted execution environment," *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies*, 2023.

[18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 315–334, 2018.