# Detection of Compromised Entities in Cross-Domain Communication Systems

SHIQING LI, UTKU TEFEK, and ERTEM ESINER, Illinois Advanced Research Center at Singapore, Singapore, Singapore
ZBIGNIEW KALBARCZYK and DEMING CHEN, University of Illinois Urbana-Champaign, Urbana, Illinois, USA

Cross-domain communication systems are important in multiple areas such as smart factories and automatic driving. Derived from real-life scenarios, each domain has a local network and accesses the public network via a central server, which makes it easy to manage the local network and protect entities in the domain. Based on this, a cross-domain communication system follows the "*device-server-server-device*" pattern. For each communication session, there are two steps: authentication and communication. The authentication phase helps two parties to build a secure communication channel. If the server is compromised, the authentication request can be sent to the attacker and then the attacker can impersonate the original receiver. As a result, it is necessary to effectively solve this server-compromise case. Further, anomaly detection is required to monitor the behaviors of entities in domains. Current deep learning solutions deploy autoencoder-based models to reconstruct input data and then detect anomalies. However, they feed raw input data into the models, which makes it hard to reconstruct evolving data streams.

In this work, we focus on the detection of compromised entities for cross-domain communication systems. Specifically, we split the problem into two parts: the server-compromise case in the authentication phase and anomaly detection for the whole system. Toward the server-compromise case, we propose a blockchain-based "double verification" scheme to prevent the server from making decisions on its own. Specifically, nodes in the blockchain network evaluate submitted records using the public key infrastructure. Meanwhile, the distributed nature of blockchain allows us to deploy more machines as verifiers to monitor the behavior of servers. For anomaly detection, we propose to feed the degree of change of items instead of raw item values into deep learning models, which is more adaptive to evolving data streams. We propose to use the linear combination of historical data and recent data to compute the degree of change. Finally, we analyze the security properties of the proposed system and evaluate the proposed anomaly detection method using real datasets. We build a simple cross-domain communication system using the Fabric framework to simulate the "double verification" scheme and the proposed anomaly detection method achieves around 0.11 accuracy improvement on average.

CCS Concepts: • **Security and privacy → Distributed systems security**;

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme.
Authors' Contact Information: Shiqing Li (corresponding author), Illinois Advanced Research Center at Singapore, Singapore, Singapore; e-mail: shiqing.li@iarcs-create.edu.sg; Utku Tefek, Illinois Advanced Research Center at Singapore, Singapore, Singapore; e-mail: u.tefek@iarcs-create.edu.sg; Ertem Esiner, Illinois Advanced Research Center at Singapore, Singapore, Singapore; e-mail: e.esiner@iarcs-create.edu.sg; Zbigniew Kalbarczyk, University of Illinois Urbana-Champaign, Urbana, Illinois, USA; e-mail: kalbarcz@illinois.edu; Deming Chen, University of Illinois Urbana-Champaign, Urbana, Illinois, USA; e-mail: dchen@illinois.edu.

ACM 2378-9638/2026/1-ART7
https://doi.org/10.1145/3748819

Additional Key Words and Phrases: Cross-domain communication systems, Compromised entities, Authentication, Verification

**ACM Reference format:**
Shiqing Li, Utku Tefek, Ertem Esiner, Zbigniew Kalbarczyk, and Deming Chen. 2026. Detection of Compromised Entities in Cross-Domain Communication Systems. *ACM Trans. Cyber-Phys. Syst.* 10, 1, Article 7 (January 2026), 21 pages.
https://doi.org/10.1145/3748819

## 1 Introduction

Cross-domain communication systems have become indispensable for enabling seamless information exchange across different domains [28]. These systems, which bridge disparate networks and often involve multiple organizations, are increasingly critical in finance [6], healthcare [29], and critical infrastructures [34]. For example, data sharing between multiple administrative domains of smart factories is becoming inevitable [9]. However, the complexity and openness inherent in such systems introduce significant security challenges, particularly the risk of compromise.

Malicious attackers can compromise entities[1] using techniques such as side-channel attacks [7], phishing attacks [1], and **Man-in-the-Middle (MITM)** attacks [23]. Once attackers get credentials, the security of the entire system is at risk. When an entity is compromised, it can lead to unauthorized access, data breaches, and potentially catastrophic disruptions to system operations. The detection of these compromised entities is crucial for maintaining the integrity and security of the overall system. Effective detection mechanisms, such as intrusion detection systems, continuous monitoring, and anomaly detection algorithms, are vital for identifying compromised entities in real time. By swiftly detecting and responding to these security incidents, organizations can mitigate the impact of attacks, prevent damage, and protect sensitive information from falling into the wrong hands.

In a typical scenario, organizations (e.g., companies) use local area networks within the domain and access the public network via the central server. As a result, the basic cross-domain communication system includes regular devices and central servers of each domain. Each communication session consists of two phases: authentication and communication. Specifically, the authentication phase establishes a secure communication channel between two parties, and then the two parties can communicate. Let's say Alice wants to communicate with Bob. On the one hand, if the server in Alice's domain is compromised, the authentication request can be sent to an attacker instead of Bob. Then, the attacker can impersonate Bob. On the other hand, if Bob is compromised, the attacker can communicate with Alice and may show abnormal behaviors that are different from the real Bob. Both of these cases would put Alice in danger, and thus, it is necessary to detect compromised entities.

Detecting compromised entities in such a cross-domain communication system is challenging. For the server compromise case in the authentication phase, the key problem is that the server is fully responsible for managing user authentication. Previous cross-domain authentication works [26–28] propose to use the blockchain technique [20] to enable a secure authentication. However, they [26–28] do not consider the server compromise case. For anomaly detection, deep learning-based anomaly detection methods [3, 17, 32, 33] have been investigated. Specifically, **Autoencoders (AEs)** are widely used to detect anomalies by learning to reconstruct input data. However, some existing works [11, 14] are designed for an offline setting in which the models do not change at

[1]In this article, we use a generic term entities to depict both devices (i.e., end-user machines that can communicate with other machines) and servers.

runtime and thus cannot effectively cope with evolving data streams. For example, [11] proposes to use hidden activation values of an input and its reconstruction by the AE to improve the accuracy of the AEs. ARCUS [32] proposes to deploy multiple models and update the model pool based on a metric called reliability, which indicates whether the current model pool is reliable. If reliability is less than a threshold, ARCUS adds a new AE model to adapt to the evolving data stream. ARCUS uses AEs to reconstruct incoming data. However, when the raw data distribution shifts, these AEs struggle to maintain accuracy. Thus, how to adapt to evolving data streams requires more effective solutions.

To improve the overall security considering the compromised servers in the authentication phase and anomaly detection, we focus on the detection of compromised entities in such a cross-domain communication system. For the server compromise case in the authentication phase, we propose to use the blockchain technique and deploy additional machines as verifiers in each domain to monitor the behavior of each server. Verifiers check whether a record the server submits to a blockchain is valid/authentic or the record is a "lie" generated by a compromised server. For anomaly detection, current methods [11, 14, 32] use AE-based models to reconstruct input data and use the difference between input data and the reconstructed one as the anomaly score. In other words, current methods require stable input data to maintain the performance. Our idea is based on the fact that the related data of normal subjects should not frequently undergo drastic changes. Consequently, the **Degree of Change (DoC)** of the input data in evolving data streams is more stable than the raw input data itself, and we propose to feed the DoC of the input data rather than the raw input data to AE-based models. In summary, our key contributions are as follows.

- —Design and evaluate an approach to detect and mitigate security attacks in cross-domain communication systems. Our strategy consists of: (i) detecting compromises of user authentication servers and (ii) anomaly detection in the cross-domain communication system.
- —Develop a method for detecting compromised authentication servers. We propose a "double verification" scheme to prevent servers from making decisions on their own. Specifically, we develop a blockchain network to evaluate submitted records using the public key infrastructure. The distributed nature of the blockchain allows us to deploy additional machines as verifiers to monitor the behavior of servers. When compromised servers submit wrong records or send authentication requests to the attacker, verifiers in the two related domains attempt to detect the compromise.
- —Develop an **Machine Learning (ML)**-based adaptive (to evolving data streams) method for anomaly detection in the cross-domain communication system. Specifically, the ML model is fed with data representing the DoC in the input (i.e., data transferred among the entities in the cross-domain communication system) compared to historical data rather than raw input data. We use the linear combination of historical and recent data to compute the DoC. The experimental results show that the proposed anomaly detection method achieves around 0.11 accuracy improvement on average compared to the state-of-the-artwork [32].

This article is organized as follows. In Section 2, we introduce the background of cross-domain communication systems, consortium blockchain, and anomaly detection. In Section 3, we discuss our assumptions and detail the "double verification" authentication scheme. In Section 4, we show the proposed anomaly detection methodology. Section 5 shows the experimental results and analysis. Finally, we conclude this work in Section 6.

## 2 Background and Related Works

In this section, we first give a brief introduction to the cross-domain communication system model. Then, we introduce the consortium blockchain and some existing works targeting cross-domain

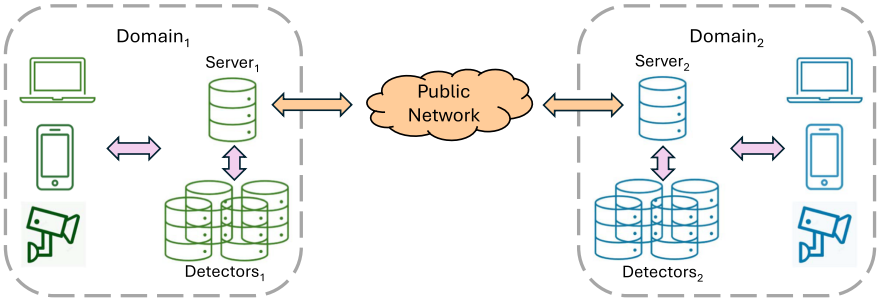



Fig. 1. The cross-domain communication system model.

authentication using blockchain. Finally, several deep learning-based anomaly detection methods are discussed.

### 2.1 Cross-Domain Communication System

Figure 1 shows the basic cross-domain communication system model. This model is derived from the reality that domains (e.g., companies or universities) have their own local networks and access the public network via a central server. Previous cross-domain authentication work [26, 27, 30] also targets similar systems. The detectors play the role of verifiers in the authentication phase (see Section 3) and anomaly detectors (see Section 4) in the communication phase. Once detectors detect an attack, they inform the whole domain. Let's say Alice wants to communicate with Bob in the other domain. The authentication phase establishes a secure communication channel between Alice and Bob, and then the two parties can communicate. Servers play an important role in the authentication phase. Specifically, if the server in Alice's domain is compromised, it can transfer the authentication request to an attacker Eve. As a result, Eve can impersonate as Bob and then conduct an attack. If any entity (i.e., servers or devices) is compromised, the attacker can pass the authentication phase. In this case, the attacker's activity may lead to abnormal system behavior, which could be detected by an anomaly detector.

In this work, we focus on the detection of compromised entities in cross-domain communication systems. We split the problem into the server-compromise case in the authentication phase and anomaly detection for the whole system. For the server-compromise case, we propose a "double verification" scheme to prevent the server from making decisions on its own. For anomaly detection, we propose to feed the DoC of the input data instead of the raw input data into models, which is more adaptive to evolving data streams.

### 2.2 Cross-Domain Authentication and Consortium Blockchain

Robust and efficient authentication mechanisms based on consortium blockchain across different domains have been studied in the literature [26, 27, 30]. Consortium blockchains have emerged as a powerful solution, offering a decentralized yet controlled environment where multiple organizations can collaborate to securely authenticate identities and transactions. Unlike public blockchains, which are open to anyone, or private blockchains, which are restricted to a single entity, consortium blockchains provide a middle ground by allowing a group of pre-approved entities to maintain and validate the network collectively. Consortium blockchains for cross-domain authentication leverage the strengths of decentralization and shared governance to ensure a high level of trust, transparency, and security. The immutable transaction history also benefits postprocessings such as manual checking and data analysis. By distributing the authentication process across multiple trusted organizations, these blockchains eliminate single points of failure and reduce the risk of fraudulent

activities. This collaborative approach not only enhances the reliability of the authentication process but also ensures that no single entity has complete control, thus maintaining a secure system.

In [26], the authors propose a blockchain-based lightweight message authentication scheme for multi-receiver cross-domain Industrial Internet of Things. The security proof shows that the proposed scheme satisfies confidentiality and unforgeability and the experimental results show that the proposed scheme is highly secure, especially considering data confidentiality and anonymity. The work in [30] proposed a blockchain-based vehicular authentication scheme for multidomain scenarios and presented a proxy-assisted pseudonym distribution method to guarantee security and privacy protection. Specifically, the blockchain as a trust bridge stores and shares the cross-domain information to facilitate the pseudonym service and authentication. Authors in [27] design a blockchain-based cross-domain data-sharing protocol by combining broadcast encryption and proxy re-encryption techniques. The protocol realizes secure and flexible data sharing across domains without leaking the privacy of smart devices, making it suitable for the scenario where smart devices cannot access cross-domain data directly. However, all these works assume that servers are not compromised. In this work, to avoid this single-point failure, we propose to deploy more machines as verifiers to monitor the behavior of the central server. With the help of blockchain, verifiers can verify the authentication request in the sender's and receiver's domains.

### 2.3 Anomaly Detection

If a device is compromised, the attacker can pass the authentication phase without being detected. In this case, further attacker activities may lead to abnormal system behavior. An anomaly detector is necessary to detect these abnormal behaviors in real time. An anomaly is identified as a data point that deviates from the majority, representing a novel observation, a system failure, unexpected noise, or other irregularities within a system of interest. Currently, complex data streams are commonplace, especially from IoT devices and cloud-based infrastructures. These streams often consist of data items with hundreds of features, unknown correlations, and heterogeneous data types, continuously arriving in real time. This complexity presents a significant challenge for anomaly detection. The challenge becomes even more pronounced when the data stream evolves over time. This phenomenon, known as concept drift [16], occurs when the properties of the target domain change unpredictably. Such complex and evolving data streams are prevalent in various real-world scenarios, such as monitoring the related data (e.g., the wing-beat frequency) of insects under different temperatures [24].

Anomaly detection based on a deep neural network has proven to handle the complexity effectively, better than classical methods (e.g., $k$ nearest neighbors [12]) [21]. In particular, AE-based models [11, 14, 35] have been widely studied, as they are appropriate for an unsupervised setting that is natural for anomaly detection with rare labels. There are AE-based methods [11, 35] designed for an offline setting and thus they cannot effectively cope with evolving data streams. There are a few recurrent neural network-based methods proposed for time series anomaly detection [17, 33]. However, they focus on learning temporal relationships inside local sequences and incrementally updating a single model, which is not suitable for handling arbitrarily evolving data streams. In ARCUS [32], the authors propose to train multiple models to adapt to the concept drift. However, they feed the raw data into the model which may change too quickly for the model to handle well. In this work, we propose to feed the DoC of the input data into models. The key idea is that the related data of normal subjects should not frequently undergo drastic changes.

## 3 The Proposed System

In this section, we provide an overview of our system and then detail the proposed "double verification" scheme.

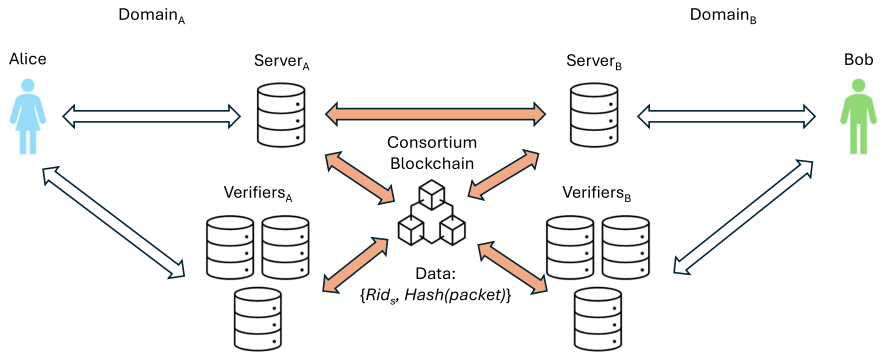



Fig. 2. The proposed cross-domain communication system.

Table 1. Notations and Definitions Used

| Notations | Definitions |
|---|---|
| $Server_A$ | The central server in domain $A$ |
| $Verifiers_A$ | Verifiers in domain $A$ |
| $PK_d^A$ | The long-term public key of entity $d$ in domain $A$ |
| $S_d^A$ | The long-term private key of entity $d$ in domain $A$ |
| $pk_d^s$ | The temporal public key of entity $d$ for session $s$ |
| $s_d^s$ | The temporal private key of entity $d$ for session $s$ |
| $Rid_s$ | The ID of the authentication request of session $s$ |
| $Sign_{s_d^s}()$ | The signature using the private key $s_d^s$ |
| $Hash(Msg)$ | The hash code of $Msg$ |

### 3.1 System Overview and Assumptions

As shown in Figure 2, there are two domains ($Domain_A$ and $Domain_B$) and a consortium blockchain network. Each domain has a central server ($Server_A$ and $Server_B$), multiple dedicated machines named verifiers, and some regular devices (e.g., Alice and Bob). We list notations in Table 1, and our threat model is as follows.

- —The adversary can eavesdrop on, block, replay, or forge messages on any communication channel between servers, verifiers, and devices.
- —The centralized server in any domain may be compromised: the attacker can read or modify its secret state (including private keys), inject arbitrary messages, or disrupt its behavior. Similarly, any device may be captured or compromised, giving the attacker access to its stored keys and allowing it to impersonate that device.
- —Each domain contains four independent verifiers. We assume that the attacker may compromise up to one verifier per domain (that is, $f = 1$ out of $n = 4$ where $f, n$ indicate the number of compromised entities and the number of all entities), following the standard **Byzantine Fault Tolerance (BFT)** bound $3 * f + 1 = n$ [5]. As long as no more than one verifier is faulty, honest verifiers will reach agreement on each authentication decision. Following a BFT threat model is advantageous because it formally bounds the number of compromised nodes while still guaranteeing both safety (no two honest verifiers decide on conflicting values) and liveness (honest verifiers eventually reach agreement). Because BFT does not require trusting

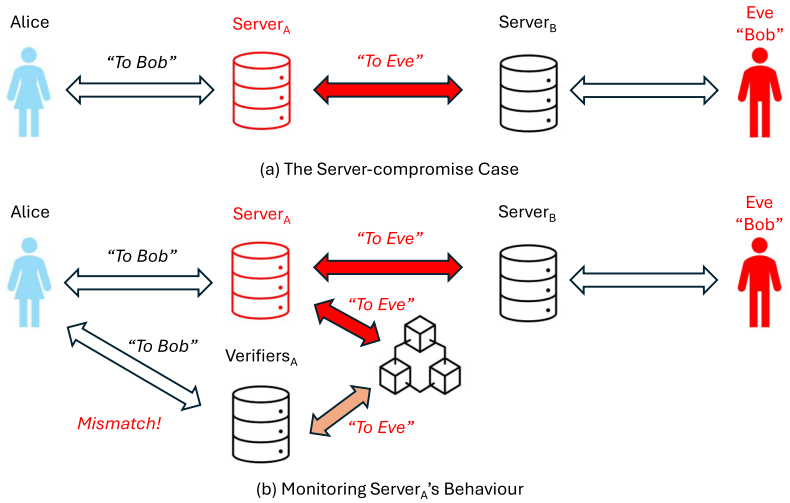



Fig. 3. Motivation for the server-compromise case.

any single verifier, it provides a well-studied and robust foundation for secure, distributed consensus in environments where some verifiers may be actively compromised.

Since blockchain relies on cryptography to validate submitted records, our system uses the **Elliptic Curve Crypto (ECC)** technology [13]. The key concepts of ECC are listed below:

— *Elliptic Curve Discrete Logarithm Problem (ECDLP)*: $Q = xP$, where $P, Q$ are points on curve $E$. Given $Q$, $P$, it is computationally hard for a **Probabilistic Polynomial-Time (PPT)** adversary to calculate $x$. Here, $x$ is the secret key, and $Q$ is the public key.
— *Elliptic Curve Diffie-Hellman Problem (ECDHP)*: $X = xP, Y = yP$, where $X, Y, P$ are points on curve E. Given $X = xP$ and $Y = yP$, it is computationally hard for a PPT adversary to calculate $xyP$. However, the entity with $x$ can easily get $xyP$ given $Y$.

In this work, we assume that the attackers can control all the communication channels and can compromise servers. Meanwhile, attackers are PPT adversaries, and thus they cannot crack current public key infrastructures.

### 3.2 The "Double Verification" Scheme

*3.2.1 Motivation.* As shown in Figure 3, we use the case that $Server_A$ is compromised by an attacker Eve in domain $B$ as our motivational example. In this case, $Server_A$ can transfer the authentication request which is from Alice to Bob to Eve and then Eve can impersonate Bob. The fundamental problem is that $Server_A$ itself executes the authentication phase. To solve this case, we use blockchain and deploy verifiers to monitor the server's behavior. As shown in Figure 3, if $Server_A$ submits the wrong record (i.e., "To Eve") to the blockchain, $Verifiers_A$ can detect the mismatch between the one (i.e., "To Bob") from Alice and the record (i.e., "To Eve") in the blockchain. In addition, the compromised server can submit "To Bob" to the blockchain but send the request to Eve. We discuss this case later in this section.

*3.2.2 The "Double Verification" Scheme.* Cross-domain authentication in our system includes the following steps. Let's say Alice in domain $A$ wants to communicate with Bob in domain $B$.

- Step 1: Alice first generates a temporal key pair $s^s_{Alice}, pk^s_{Alice}$ and inputs three other components: $Rid_s$, the sender (i.e., Alice), and the receiver (i.e., Bob). The $Rid_s$ is a *string* variable and includes a timestamp to prevent the replay attack. Alice signs the three components using $s^s_{Alice}$ and the Elliptic Curve Digital Signature Algorithm. Here, Alice has the basic request *packet* which includes the $Rid_s$, the sender, the receiver, $pk^s_{Alice}$, and $Sign_{s^s_{Alice}}()$. Finally, Alice signs the packet using her long-term secret key $S^A_{Alice}$ and then sends the packet with the signature to $Server_A$. To reduce the communication overhead and conceal the true information, the packet is hashed to a 256-bit string using `SHA-256`. Overall, Alice sends $Rid_s$, $Hash(packet)$, and the corresponding signature to $Verifiers_A$.
- Step 2: $Server_A$ first verifies the two signatures using $PK^A_{Alice}$ and $pk^s_{Alice}$. If the verification fails, the server can alert Alice about this. Otherwise, $Server_A$ submits the request to the blockchain in the format of $Rid_s$, $Hash(packet)$. Meanwhile, $Server_A$ sends the request and $Sign_{S^A_{Server_A}}()$ to $Server_B$.
- Step 3: $Verifiers_A$ first verifies the signature using $PK^A_{Alice}$. Then, they keep trying to fetch the submitted record of $Server_A$. Once they fetch the record, they compare it with the one received from Alice. Finally, the compared results with the signature $Sign_{S^A_{Veri}}()$ are sent to Alice.
- Step 4: Alice first verifies the signature using $PK^A_{Verifiers_A}$. Alice can alert domain $A$ if `False` is received. Meanwhile, Alice can also identify whether verifiers are in good condition by checking whether verifiers send the same response. For example, if we have three verifiers, two verifiers return `True` and the third one returns `False`. In this case, the third verifier may be compromised or crashed.
- Step 5: Once $Server_B$ receives the packet and verifies the signature using $PK^A_{Server_A}$. If the verification passes, $Server_B$ sends the request and the corresponding signature $Sign_{S^B_{Server_B}}()$ to Bob.
- Step 6: Bob first verifies the signature using $PK^B_{Server_B}$. Then, Bob sends $Rid_s$, $Hash(packet)$, and the corresponding signature $Sign_{S^B_{Bob}}()$ to $Verifiers_B$.
- Step 7: $Verifiers_B$ first verifies the signature using $PK^B_{Bob}$. Then, they keep trying to fetch the submitted record of $Server_A$. Once they fetch the record, they compare it with the one received from Bob. Finally, the compared results with the signature $Sign_{S^B_{Veri}}()$ are sent to Bob.
- Step 8: Bob does the same operations as Alice does in Step 4.

If everything goes well, Bob gets $pk^s_{Alice}$. Then, Bob repeats the above steps, and Alice can get $pk^s_{Bob}$. Based on the ECDHP assumption, Alice and Bob can build a secure channel using their temporal key pairs. Specifically, Alice can get the session key by $s^s_{Alice} * pk^s_{Bob}$, and Bob can get the session key by $s^s_{Bob} * pk^s_{Alice}$. With the help of the consortium blockchain, we implement a secure cross-domain communication system. First, all the submitted records are evaluated by other nodes in the blockchain network, which is part of the consensus algorithm [20]. Second, the distributed nature of blockchain allows us to deploy more machines to monitor the behavior of servers. Third, since the consortium blockchain is a permissioned blockchain, we can trust the other organizations and thus do not need certificate authorities. This is analogous to how banks use the consortium blockchain to collaborate with each other [6].

Why do we need verifiers instead of verifying submitted records by the device itself? There are two reasons. First, the verification part incurs a computation overhead. Deploying these verifiers supports the case that devices are low-performance embedded devices. Second, deploying more verifiers increases the security level of the whole system. Since all the verifiers compare the record

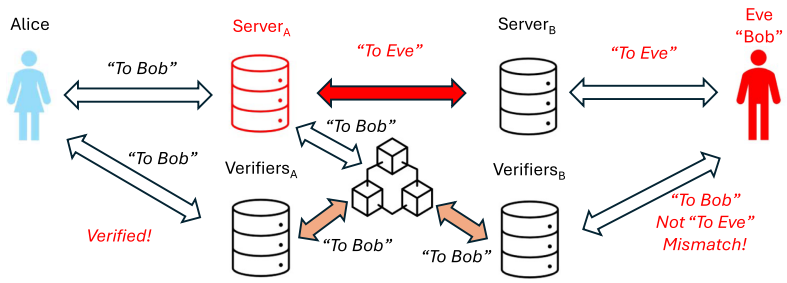



Fig. 4. The other case in the motivational example.

from Alice and the one from the blockchain, they should return the same response to Alice. Following our threat model, Alice follows the decision of the majority of verifiers to judge whether there are compromised entities.

### 3.3 Security Analysis

In this subsection, we analyze the security of the proposed authentication scheme. First, we show how the proposed scheme solves the case that the compromised $Server_A$ sends "To Bob" (mentioned in Section 3.2.1).

As shown in Figure 4, if $Server_A$ submits "To Bob" to the blockchain but actually it sends the request to Eve to $Server_B$, $Verifiers_A$ verifies the submitted record "To Bob" and thus $Server_A$ can deceive $Verifiers_A$. However, $Verifiers_B$ can find that the receiver does not match and detect the compromised entities. As shown in Figure 4, $Verifiers_B$ receives the request from Eve but fetches the blockchain record "To Bob." As a result, $Verifiers_B$ detects this mismatch between the legitimate receiver, Bob, and the impostor, Eve. The "mismatch" shown in Figure 4 indicates this mismatch. Following the system model in Section 2.1, $Verifiers_B$ will notify the whole domain. This is also why we name the proposed scheme a "double verification" scheme. Only if verifiers in both domains verify the request, the two parties confirm that they are communicating with the correct entity. We discuss the following typical malicious attacks.

*3.3.1 Brute Force Attacks.* Attackers attempt to guess a device's secret key by systematically trying all possible answers until the correct one is found. Since we use 256-bit ECC-based keys, the difficulty of solving the ECDLP adds a layer of complexity that protects against brute-force attacks. Besides, there are $2^{128}$ possibilities to solve the ECDLP.

*3.3.2 MITM Attacks.* Attackers intercept communication between a user and the authentication server, potentially stealing login credentials or injecting malicious data. Since all the devices, servers, and verifiers hold a long-term public key list of others in each domain, all the messages are signed (i.e., $Sign_{privatekey}()$ in the steps) with their long-term private keys. As a result, Alice and Bob can get the other's temporal public key and build a secure channel. When they communicate, the attacker still cannot decrypt messages based on the assumption of ECDHP.

*3.3.3 Replay Attacks.* Attackers capture and reuse a valid data transmission, such as an authentication token or password, to gain unauthorized access. To prevent replay attacks, Alice generates a unique authentication request ID in the format of "Alice" plus the current timestamp. When $Server_A$ submits the record to the blockchain network in Step 2, the ID acts as a key. If the key exists, the blockchain returns an error indicating "the key exists" and the authentication process terminates. In this way, we can prevent replay attacks.

*3.3.4 Session Hijacking.* Attackers steal or predict session tokens, which allows them to impersonate a user within an ongoing session. In the proposed system, Alice and Bob use temporal key pairs (i.e., $s^s_{Alice}, pk^s_{Alice}, s^s_{Bob}, pk^s_{Bob}$) for each session. As a result, the attackers cannot reuse session keys even if they get the previous ones.

*3.3.5 Impersonate Attack.* Attackers compromise the server to send the authentication request to their devices and then impersonate the original receiver as shown in Figure 3. We use blockchain and deploy verifiers to ensure that the server does the right thing. Besides, we also propose a novel anomaly detection method in Section 4 to detect any potentially compromised entities.

## 4 Anomaly Detection for the System

In the previous section, we target the situation in which $Server_A$ is compromised in the authentication phase. The next question is: What will happen if Alice is compromised? If Alice is compromised, the attacker can pass the authentication phase and build a secure channel with the victim. As long as Alice is compromised, the attacker can pass the authentication phase regardless of whether $Server_A$ is compromised. As a result, we only discuss the situation in which Alice is compromised. In this situation, the attacker's activity may lead to abnormal behavior with respect to network traffic. Thus, an efficient anomaly detection method is required in the communication phase.

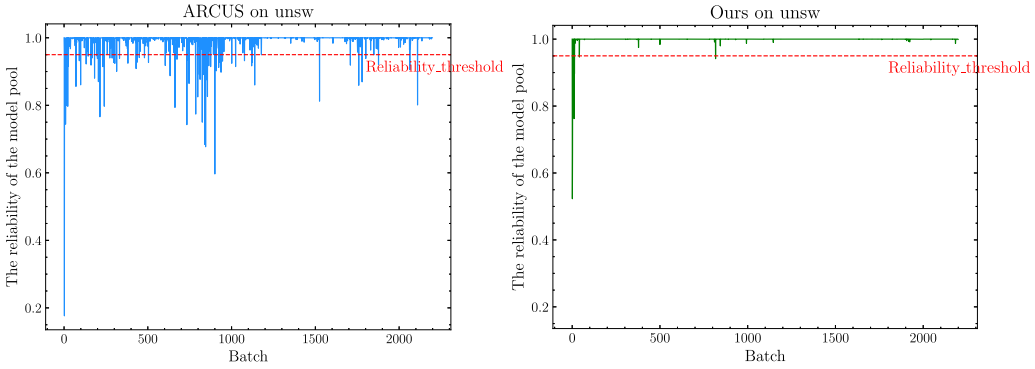
### 4.1 Preliminaries

Since we aim to detect anomalies timely in the system as shown in Figure 2, the anomaly detection model processes the input data in a live-streaming fashion. Given a sequence of inputs $< \ldots, x_{t-1}, x_t, x_{t+1} \ldots >$ in a data stream, an anomaly detection model $M$ determines the anomaly score of each data input and reports the data points whose anomaly scores exceed a predefined threshold indicating abnormal behaviors. Since we do not have labeled data in real-life scenarios, the anomaly detection model should work in an unsupervised fashion. The model first scores data points in the newly coming batch (i.e., a fixed number of consecutive data points) and then updates its parameters. Among current deep learning models, AE-based models achieve great success. As shown in Equation (1), an AE model has an encoder and a decoder. It uses the two components to reconstruct the input data, and the loss is regarded as the anomaly score. A higher loss indicates that it is hard to reconstruct the input and thus is likely an anomaly:

$$loss = MSE(x_{in}, Decoder(Encoder(x_{in}))), \tag{1}$$

where $MSE$ refers to the mean squared error, $x_{in}$ refers to the input vector, and *Encoder and Decoder* refer to the layers in the model (e.g., traditional AEs use fully connected layers).

The basic AE model can handle stable data streams. However, it fails to process evolving data streams that outdate the trained models. This phenomenon is referred to as concept drift [16]. As a concept refers to a certain distributional or statistical property of data in a domain, concept drift refers to an extrinsic phenomenon of the concept changing arbitrarily over time. The concept drift can be formulated as $P_t(X, y) \neq P_{t+1}(X, y)$ where $P_t(X, y)$ refers to the probability of an input item $X$ labeled as $y$ at time $t$ based on profiling. (For example, insects show different behaviors under different temperatures.) When concept drift occurs, the current anomaly detection model may generate a wrong anomaly score and should learn the new concept. To handle this, ARCUS [32] proposes to deploy multiple models to adapt to the concept drift. Specifically, it includes two steps: concept-driven inference and concept drift-aware update. When the new batch of data arrives, ARCUS first evaluates the reliability of the model pool according to Hoeffding's Inequality-based Mean Difference Bound [8]. The reliability of a single model is derived as in Definition 1 [32].




(a) The reliability of models using ARCUS on unsw (b) The reliability of models using ours on unsw

Fig. 5. The motivational example of our proposed anomaly detection.

Based on Definition 1, given a model pool $P = \{M_1, \ldots, M_k\}$ ($P$ only has $M_1$ at the beginning), the reliability $r_P$ of $P$ is $1 - \prod_{i=1}^{k}(1 - r_{M_i})$.

*Definition 1.* The model reliability $r_M$ of $M$ for the current batch $B_{Curr}$ is defined as

$$r_M = e^{\frac{-b\epsilon^2}{(s_{max}-s_{min})^2}}, \tag{2}$$

where $b$ is the length of each batch, $\epsilon$ refers to the difference of the average anomaly score between the current batch and the last batch, and $s_{max}$ and $s_{min}$ refer to the maximum anomaly score and the minimum anomaly score in the latest two batches.

Based on the above concepts, ARCUS monitors the reliability of the current model pool and triggers the update of the pool with a significance level $1 - \alpha$. A model pool will be kept unchanged when it has at least a single highly reliable model (i.e., $r_P > \alpha$), but it will be adjusted when the models in the pool have only neutral reliability values (i.e., $r_p \ll \alpha$). The default value of $\alpha$ is set as 0.95, which is commonly used in the statistical significance test, meaning that only 5% of the possibility that all models are not reliable is allowed. Once the update of the model pool is triggered, ARCUS creates a new model with the current batch and this new model is added to the model pool.

### 4.2 Motivation

Considering that the data stream is evolving, the values of the input data may keep changing. However, ARCUS feeds the raw input data into the model, which makes it hard to learn the implicit pattern in the evolving data stream. To prove that the unstable raw data may affect the reliability of the model pool, we evaluate ARCUS on the *unsw* [18] network traffic dataset. The *unsw* dataset is widely used in the area of anomaly detection and is similar to the environment of cross-domain communication systems. It was created by the IXIA PerfectStorm tool in the Cyber Range Lab of UNSW Canberra for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviors. As shown in Figure 5(a), the reliability varies along batches and is often lower than the reliability threshold (i.e., 0.95 used in ARCUS by default). In other words, the model pool cannot maintain its performance and ARCUS creates 78 models to adapt to the data stream. Instead, we propose to feed the DoC of the input data to models. We calculate DoC between two scalars $A$ and $B$ via $(A - B)/B$ and refer to it as *DoC(A, B)*. As shown in Figure 5(b), our proposed method has a stable reliability curve except for the beginning part. Meanwhile, we only create eight models to adapt to the data stream.

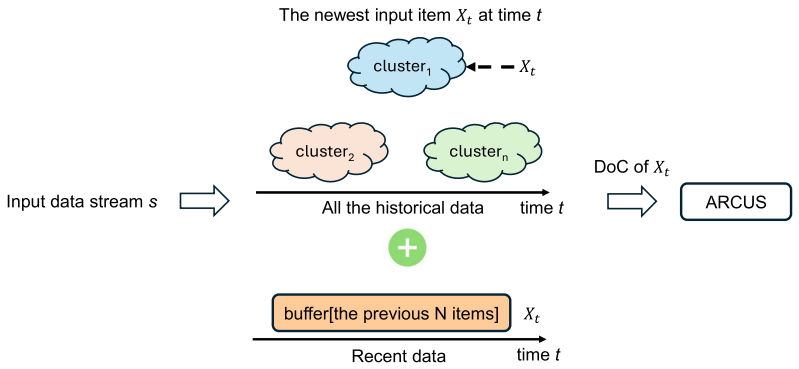



Fig. 6. The overall workflow of the proposed anomaly detection method.

### 4.3 Our Proposed Data Preprocessing Method

How do we detect the compromised entities by analyzing the related data (e.g., the network traffic)? The basic principle of AE-based models is to reconstruct the input data. However, since the specific values of the raw data may change over time, the trained models can be outdated and unable to reconstruct the input data. Our proposed DoC normalization constrains the input values to a smaller range, mitigating this issue. For example, if the input data are [100, 150, 200], the corresponding DoC is [0, 0.5, 0.33]. DoC is more stable if the data are changing at a certain rate. For example, in a monotonically increasing data stream [0.1, 0.15, 0.225, 0.3375, 0.50625, 0.759375], the raw data change over time, but the DoC between the current value (e.g., 0.15) and the previous value (e.g., 0.1) is always 50%. However, which value should we use to calculate the DoC for the coming input data? On the one hand, the values of all the historical data may form clusters under different situations. For example, the road traffic data are high when people are going to work and returning home and lower during the remaining hours. In this case, two clusters are formed. On the other hand, recent data reflect the current trend. Consequently, we propose to combine these two parts to get the DoC and feed it into models. The overall workflow is shown in Figure 6. As shown in Figure 6, we first preprocess the input data following Algorithm 1 and then feed the processed data into ARCUS. Please refer to further details of ARCUS in [32].

Note that the input data $x$ (e.g., the input data in benchmarks shown in Table 2) is a multi-dimension vector and we process each dimension separately. As mentioned above, we need to compute the DoC based on historical data (denoted as $DoC_{cluster}$) and recent data (denoted as $DoC_{recent}$) for each scalar $x_d$ (i.e., the $d$th dimension of the input vector $x$). To compute the final $DoC_{x_d}$, we use a linear combination of $DoC_{cluster}$ and $DoC_{recent}$ as shown in Equation (3):

$$DoC_{x_d} = \alpha * DoC_{cluster} + (1 - \alpha) * DoC_{recent}, \tag{3}$$

where $\alpha$ is the proportion that $DoC_{cluster}$ accounts for and we empirically try different values to determine the value of $\alpha$ and show the influence in Section 5.4.

As shown in Algorithm 1, given an input data stream $s$, we first initialize a list to store the current clusters, a list to store the DoC results, and $buffer_{recent}$ which stores the previous $step$ items for each dimension of the input item (line 1). We hold two pieces of data for each cluster: $mean_c$ (i.e., the mean of elements in the cluster) and the number of elements in the cluster. For a cluster $c$, $DoC(x_d, c)$ is actually $DoC(x_d, mean_c)$ and is calculated by $(x_d - mean_c)/mean_c$. We consider $DoC(x_d, c)$ as the distance between $x_d$ and cluster $c$ and a smaller $DoC(x_d, c)$ indicates that $x_d$ is closer to $c$. For each scalar $x_d$, we traverse current clusters to find the closest cluster $c$ which has the minimum $DoC(x_d, c)$ (line 4). In addition, we do not trust clusters with too few elements.

**Algorithm 1:** The Proposed Data Preprocessing Algorithm

**Input:** The input data stream $s$, $\alpha$, $thres$, $step$
**Output:** The degree of change $DoC$ of each item $x$ in the input stream $s$

1: Initialize clusters, $DoC$, and $buffer_{recent}$ with $step$ elements for each dimension of $x$
2: **for** the newest input vector $x$ in $s$ **do**
3:  **for** $x_d$, the dimension $d$ of $x$ **do**
4:  Find the closest cluster $c$ for $x_d$
5:  **if** the number of items in $c$ is less than 5 **then**
6:  $DoC_{cluster}$ equals the DoC of the closest cluster with more than 5 elements
7:  **else**
8:  $DoC_{cluster}$ equals the DoC of $c$
9:  **end if**
10:  Update clusters according to $thres$
11:  Calculate $DoC_{recent}$ using $x_d$ and the mean of $buffer_{recent}$
12:  Update $buffer_{recent}$ in a FIFO manner
13:  Calculate $DoC_{x_d}$ using $\alpha$, $DoC_{cluster}$, and $DoC_{recent}$
14:  Append $DoC_{x_d}$ to $DoC$
15:  **end for**
16: **end for**
17: **Return** $DoC$

Table 2. Real Data Benchmarks for Anomaly Detection

| Dataset | Description | # Obj | # Dim | Anomaly Target |
|---|---|---|---|---|
| INSECTS-Abr | Optical sensor values for insects | 44,569 | 33 | The southern house mosquito |
| INSECTS-Inc | Optical sensor values for insects | 48,086 | 33 | The southern house mosquito |
| INSECTS-IncGrd | Optical sensor values for insects | 20,367 | 33 | The southern house mosquito |
| INSECTS-IncRec | Optical sensor values for insects | 67,455 | 33 | The southern house mosquito |
| GAS | Gas sensor values | 12,114 | 128 | Acetaldehyde |
| RIALTO | Building images around a bridge | 74,848 | 27 | The building class 0 |
| unsw | Network traffic at UNSW | 2,540,044 | 122 | Network packet |
| CREATE | Building management system at CREATE | 18,689 | 10 | Sensor data |

Thus, we empirically set the threshold as five elements. If $c$ has less than five elements, we assign the $DoC_{cluster}$ of $x_d$ to the $DoC_{cluster}$ of the closest cluster from $x_d$ with more than five elements. Otherwise, we assign $DoC_{cluster}$ of $x_d$ to $DoC(x_d, c)$ (lines 5–9). To update clusters, we add $x_d$ to $c$ if $DoC(x_d, c)$ is lower than $thres$ (line 10). Here, $thres$ indicates the distance threshold, which determines whether $x_d$ can be added into cluster $c$. We empirically try different values to determine the value of $thres$ and show the influence of $thres$ in Section 5.4. If $DoC(x_d, c)$ exceeds $thres$, we create a new cluster for $x_d$. Meanwhile, $DoC_{recent}$ is computed using $x_d$ and the mean of elements stored in $buffer_{recent}$ (line 11). Then, we update $buffer_{recent}$ in a FIFO manner (line 12). Finally, we compute the $DoC_{x_d}$ (line 13). After this, we will repeat the entire process for the next dimension of $x$ (i.e., $x_{d+1}$). In addition, each dimension has its corresponding clusters and $buffer_{recent}$.

## 5 Experiments

In our experiments, we build a cross-domain communication system including two domains. Each domain has one device (represented by $PC_2$ and $Laptop$), one server (represented by $PC_1$ and $PC_3$), and verifiers (docker containers in $PC_1$ and $PC_3$) as shown in Figure 7, which is a simple prototype of the proposed system shown in Figure 2. In this section, we first show the experimental setup and

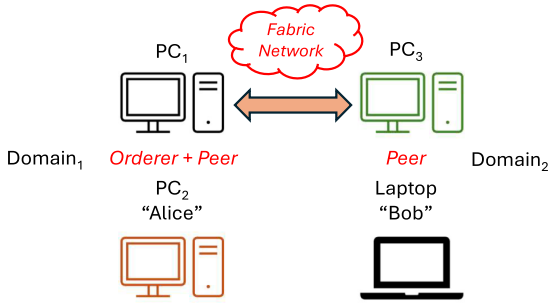



Fig. 7. The experimental environment.

evaluate the computation and communication overhead of the proposed authentication scheme. Then, we show the performance of the proposed anomaly detection method.

### 5.1 Experimental Setup

To evaluate the performance of the proposed system in a real scenario, we use the Hyperledger Fabric [2] framework to build the blockchain network. Hyperledger Fabric is an open source enterprise-grade permissioned distributed ledger technology platform. Specifically, we use Java as the main programming language and implement chaincodes (i.e., smart contracts in other blockchain platforms (e.g., Ethereum [4])) and applications. Nodes communicate through an overlay network built with Docker Swarm. In the experiment, the cryptographic tool library is the Java Security package and we use 256-bit keys (implemented by `KeyPairGenerator.getInstance("EC");` `keyGen.initialize(256);`). We use $PC_1$ running the Ubuntu 18.04.6 OS to simulate the server and verifiers of the first domain; this PC is equipped with Intel Core i9-9900 CPUs @3.6 GHz and 64 GB of memory. The device of the first domain is acted by $PC_2$ that runs Windows 11 Pro system and is equipped with Intel i7-12700 CPUs @2.1 GHz and 32 GB of memory. Besides, we use a laptop to simulate the device in the second domain. This laptop runs the Windows 11 Pro OS and uses 1.8 GHz Intel i7-1365U CPUs and 16 GB of memory. Similarly, $PC_3$ equipped with E5-1620 CPUs @3.5 GHz and running the Ubuntu 18.04.6 OS acts as the server and verifiers of the second domain. $PC_1$ runs the orderer nodes and four peer nodes and $PC_3$ runs four peer nodes. For the anomaly detection part, we follow the open source code of ARCUS[2] and train the preprocessed data using a Nvidia GeForce RTX 2080Ti GPU and cuda-11.7. We use the **Area Under Receiver Operating Characteristic (AUC)** as the accuracy metric.

We use all the real data benchmarks in ARCUS to make a fair comparison as shown in Table 2. In addition, we also add an extra network traffic benchmark unsw [18] which is more similar to the environment of communication systems. These benchmarks are split into two categories: known drifts and unknown drifts. For the insect-related datasets [24], since researchers manually control the temperature to generate these datasets, the drift is known. For GAS [25], RIALTO [15], and unsw, the data are collected naturally without much human intervention. In addition, we collect data from the building management system in our CREATE building and form a new dataset called CREATE. While collecting CREATE, we introduce manual disturbance to simulate compromised-entity scenarios. Specifically, we write a program to change the thermostat every 10 minutes to control the temperature in an office. In our experiments that include real-world data with inherent noise, we have not performed a controlled sensitivity analysis across varying noise

[2]https://github.com/kaist-dmlab/ARCUS.

Table 3. Execution Time Breakdown of the Authentication Phase in Milliseconds

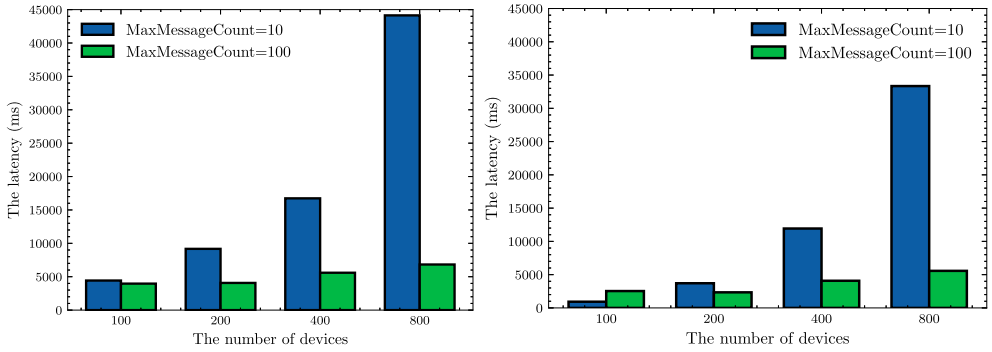
| Step | Total Time | Prepare Time | Write to Blockchain | Verifiers | Others |
|---|---|---|---|---|---|
| *Alice* ⟶ *Bob* | 2,560 | 62.2 (2.43%) | 2,025.4 (79.20%) | 385.4 (14.99%) | 87 (3.38%) |
| *Bob* ⟶ *Alice* | 2,630 | 83.75 (3.18%) | 2,485.75 (94.52%) | 4.75 (0.18%) | 55.75 (2.12%) |

intensities. Likewise, evaluating adversaries that dynamically adjust their tactics or developing online adaptation methodology remains an important direction for future research.

### 5.2 The Computation and Communication Overhead of the Authentication Phase

We simulate the authentication phase between Alice (i.e., $PC_2$) and Bob (i.e., the laptop). Table 3 shows the execution time breakdown of the authentication phase. Overall, we take around 5,190 ms to finish the authentication phase. The prepare time refers to the time of generating the authentication request packet (i.e., Step 1 in Section 3.2). The verifiers part in the table refers to the verification time spent in the receiver's domain. The reason is that the time spent in the sender's domain can be overlapped by the remaining operations (e.g., Steps 5–8 in Section 3.2). As shown in Table 3, the extra time induced by verifiers is around 7.52% of the overall time. In addition, the main bottleneck is the time spent on writing to the blockchain. Toward the communication overhead, we send $Hash(packet)$ instead of the full packet to reduce the communication overhead and the overhead mainly refers to communications with verifiers. Specifically, the network packet includes a 14-byte $Rid_s$, a 64-byte hashcode in *String*, and a 96-byte signature. Besides, verifiers also respond to Alice or Bob with their signatures. Specifically, the packet includes a 1-byte binary result and a 96-byte signature. The verifiers also fetch records from the blockchain using $Rid_s$ as the key. Assume that we have $N$ verifiers in each domain. The communication overhead is around $(N * 174 + N * 97 + N * 174 + N * 110) * 4$ in two steps. In this small system, we simulate three verifiers and the overhead is around 6,660 bytes. Since all the messages should be signed to ensure safety, signatures consume 4,608 bytes. Note that this overhead depends on the length of the keys which is related to the required safety level.

We further simulate large-scale scenarios. In this demo scenario, we extend the blockchain network with two additional domains. We run two server applications on each of the two PCs ($PC_1$ and $PC_3$) to act as central servers of the four domains and leverage the `java.util.concurrent` library to create multiple threads that act as devices in domains. Specifically, we evaluate configurations with 100, 200, 400, and 800 threads. As shown in Figure 8, we see that the average latency (i.e., 4,427, 9,173, 16,737, 44,137) is approximately proportional to the number of threads. Similarly, the latency (i.e., 933, 3,708, 11,932, 33,343) of the `submit` operation (i.e., a node submits a record to the blockchain network) has the same trend. To optimize performance, we change three parameters in the blockchain network. The first two are *MaxMessageCount* that indicates the maximum number of messages to permit in a block, and the corresponding *PreferredMaxBytes* that indicates the preferred maximum number of bytes allowed for the serialized messages in a block. Since the requests follow the same chaincode, *PreferredMaxBytes* is proportional to *MaxMessageCount*. The third one is *BatchTimeout* that indicates the amount of time to wait before creating a block. Specifically, we change *MaxMessageCount* from 10 to 100, *PreferredMaxBytes* from 512 KB to 5,120 KB, and *BatchTimeout* from 2 s to 500 ms. We use *Base* to refer to the original setting and *Optimized* to refer to the changed setting. As we can see from Figure 8, the latency for the Optimized setting is significantly reduced to (3,966, 4,070, 5,596, 6,829) and the latency of the `submit` operation is reduced to (2,534, 2,342, 4,081, 5,570). The main reason is that we reduce the number of blocks by increasing the size of each block (i.e., increasing *MaxMessageCount*). In addition, the hardware platform (e.g., CPUs, disks, and network), the chaincode program language, and the consensus




(a) The average latency of the process (b) The average latency of the submit operation

Fig. 8. The performance comparison between two blockchain settings.


(a) $PC_1$ is compromised




(b) $PC_1$ and two verifiers are compromised

Fig. 9. Responses of verifiers.

algorithm can affect the performance. When we deploy this scheme in real-life scenarios, all these settings should be considered according to the performance requirement and experimental setup.

### 5.3 What Will Happen If $Server_A$ Is Compromised?

In Section 3.2, we show that the proposed "double verification" scheme can detect compromised servers using two cases when $Server_A$ is compromised. Specifically, $Server_A$ may submit a "lie" to the blockchain or submit a "truth" but send the authentication request to an attacker. In this subsection, we show the screenshots of responses from verifiers in Alice's domain in Figure 9(a) when $PC_1$ (i.e., $Server_A$) submits a "lie." When $PC_1$ submits a "lie," the three verifiers in this simple prototype find the mismatch between the submitted authentication request and the request sent by Alice. As shown in Figure 9(a), three verifiers respond "False" to Alice. In the "truth" case, verifiers in Bob's domain can find the mismatch between the submitted request "To Bob" and the request received from Eve. The responses are the same as the ones shown in Figure 9(a). In addition, we also show the case when $PC_1$ and two verifiers in Alice's domain are compromised. As shown in Figure 9(b), there is one response with the content "False" (i.e., the submitted record does not match the one sent by Alice) from the uncompromised verifier. Besides, two compromised verifiers respond "True" to deceive Alice. Thanks to the uncompromised verifier, Alice can detect the compromised server and the two compromised verifiers. However, a limitation of this work is that it is hard to secure the system if all the verifiers are compromised. This is why we mention deploying more verifiers can increase the security level of the whole system in Section 3.2.

### 5.4 Anomaly Detection

We first compare the accuracy (i.e., AUC) with other streaming anomaly detection works including sLSTM-ED [17], sREBM [33], STARE [31], RRCF [10], MiLOF [22], DILOF [19], and MStream [3]

Table 4. Overall Accuracy Comparison

| Dataset | Ours | ARCUS | sLSTM-ED | sREBM | STARE | RRCF | MiLOF | DILOF | MStream |
|---|---|---|---|---|---|---|---|---|---|
| INSECTS-Abr | **0.753** | 0.631 | 0.749 | 0.471 | 0.555 | 0.695 | 0.393 | 0.730 | 0.709 |
| INSECTS-Inc | 0.706 | 0.600 | 0.696 | 0.383 | 0.559 | 0.669 | 0.415 | **0.757** | 0.593 |
| INSECTS-IncGrd | 0.753 | 0.641 | **0.795** | 0.575 | 0.594 | 0.719 | 0.395 | 0.746 | 0.628 |
| INSECTS-IncRec | **0.743** | 0.634 | 0.709 | 0.491 | 0.551 | 0.680 | 0.381 | 0.743 | 0.637 |
| GAS | **0.88** | 0.878 | 0.408 | 0.506 | 0.635 | 0.804 | 0.589 | 0.470 | 0.480 |
| RIALTO | **0.875** | 0.784 | 0.617 | 0.492 | 0.532 | 0.731 | 0.456 | 0.742 | 0.699 |
| unsw | **0.607** | 0.466 | NA | NA | NA | NA | NA | NA | NA |
| CREATE | **0.881** | 0.67 | NA | NA | NA | NA | NA | NA | NA |

The bold one indicates the highest accuracy.

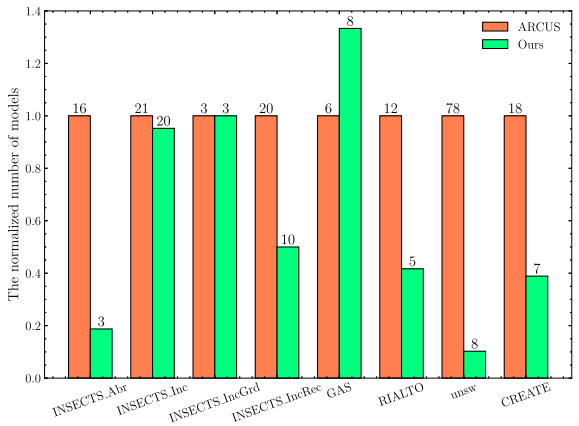



Fig. 10. The number of deployed models.

in Table 4. As shown in the table, our proposed method can achieve the top-2 accuracy in all the benchmarks. Especially, we have increased around 0.11 AUC on average compared to ARCUS. We also show the heatmap of `INSECTS-Abr` and `RIALTO` in Figures 12 and 13. Recall that $\alpha$ represents the weight of historical data, $1-\alpha$ is the weight of recent data, *Thres* represents the DoC threshold between a new item and existing clusters, and *Steps* represents the number of buffered recent items. When $\alpha$ is 0.0 or 1.0, we only consider recent data or historical data. As shown in the figures, we cannot achieve the highest accuracy if we only consider one part. Especially, if we only consider the historical data, the performance is pretty low as shown in Figures 12(e) and 13(e). Besides, different benchmarks have different features. As shown in Figure 12, `INSECTS-Abr` prefers a smaller $\alpha$. However, `RIALTO` shows similar performance with different values of $\alpha$ and prefers to buffer around 50 recent items as shown in Figure 13.

We also show the number of models that we deploy in Figure 10. As shown in the figure, our proposed method can reduce the number of deployed models, which indicates that the DoC is more stable than the raw data. As shown in Figure 10, we have a relatively poor performance on `GAS` in terms of the number of deployed models. We infer that the following reasons may apply. First, `GAS` includes data items that have high SD. The highest one is around 72,985 while the SD of most of the other benchmarks are below 1. The second reason is that `GAS` has little data. For instance, `unsw`'s data volume is 209 times that of `GAS`. By using fewer models, the proposed method requires less memory than ARCUS. For example, we reduce the number of models by 89.74% (i.e., 8 models

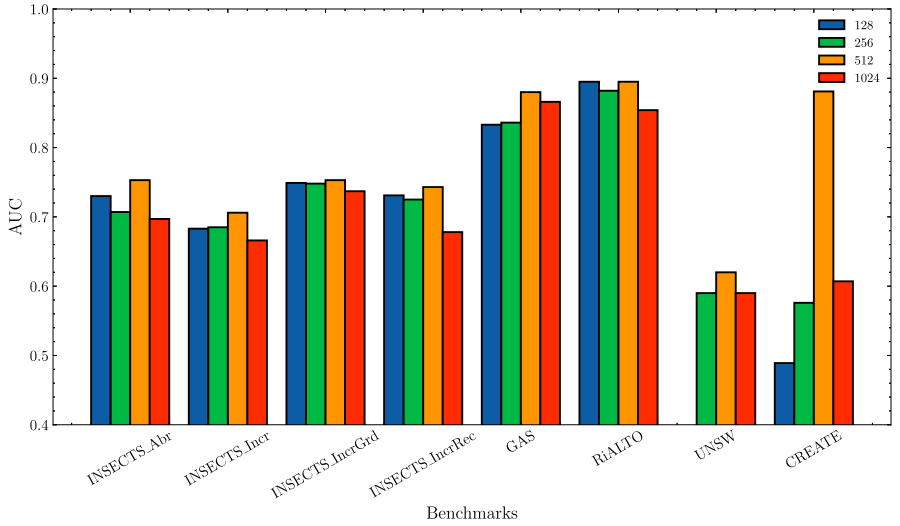
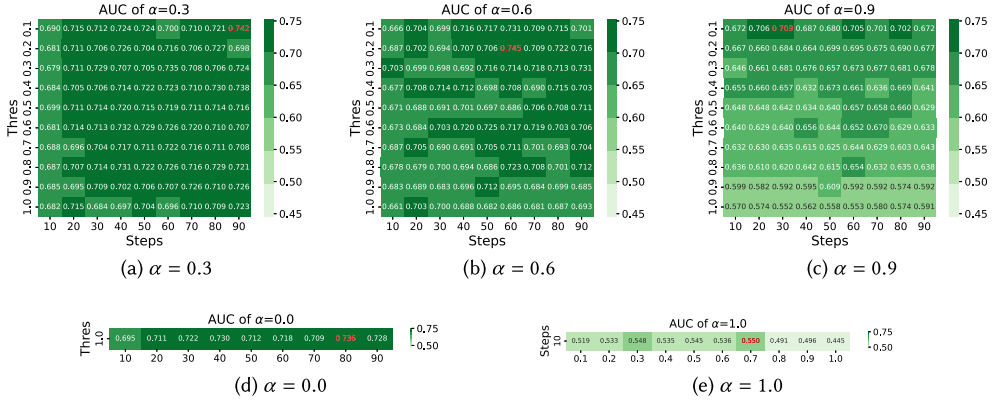



Fig. 11. AUCs under different batch sizes.




(a) $\alpha = 0.3$ (b) $\alpha = 0.6$ (c) $\alpha = 0.9$

(d) $\alpha = 0.0$ (e) $\alpha = 1.0$

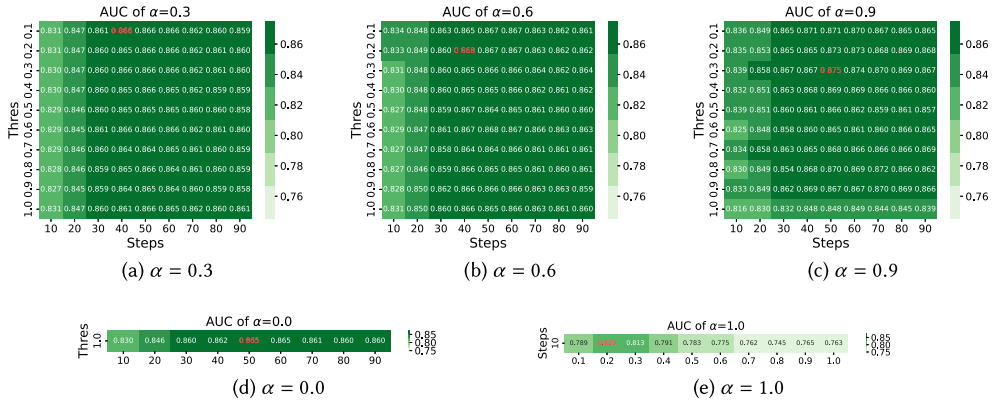
Fig. 12. The AUC of different *alpha* on INSECTS-Abr.

used by us versus 78 models used by ARCUS) for unsw, which is equivalent to a similar amount of reduction on memory consumption. Such a significant amount of improvement demonstrates one of the key advantages of our solution for much reduced design complexity, resource utilization, and system cost.

Figure 11 displays the AUCs for various batch sizes. As the figure indicates, a batch size of 512 delivers the best performance across all benchmarks. Because both our reliability calculations and model retraining operate on mini-batches, using substantially smaller or larger batch sizes may degrade accuracy.

## 6 Conclusion

In this work, we focus on detecting compromised entities in cross-domain communication systems. The basic communication model follows the "device-server-server-device" pattern. Each communication session includes the authentication phase and the communication phase. Since servers play an important role in the authentication phase, we use the consortium blockchain technology




(a) $\alpha = 0.3$ (b) $\alpha = 0.6$ (c) $\alpha = 0.9$

(d) $\alpha = 0.0$ (e) $\alpha = 1.0$

Fig. 13. The AUC of different *alpha* on RIALTO.

and propose a "double verification" scheme that deploys some machines as verifiers to monitor the behavior of servers. When compromised servers submit wrong records or send the authentication request to the attacker, verifiers in the two related domains can detect them. To detect potentially compromised entities, we propose to feed the DoC of the input data instead of the raw input data into models, which is more adaptive to evolving data streams. Finally, we analyze the security properties of the proposed system and evaluate the proposed anomaly detection method using real datasets. The experimental results show that the proposed anomaly detection achieves around 0.11 accuracy improvement on average.

## Acknowledgment

We thank Hengchuan Tan for collecting the CREATE dataset.

## References

[1] Zainab Alkhalil, Chaminda Hewage, Liqaa Nawaf, and Imtiaz Khan. 2021. Phishing attacks: A recent comprehensive study and a new anatomy. *Frontiers in Computer Science* 3, (2021), 563060.

[2] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the 13th EuroSys Conference*, 1–15.

[3] Siddharth Bhatia, Arjit Jain, Pan Li, Ritesh Kumar, and Bryan Hooi. 2021. Mstream: Fast anomaly detection in multi-aspect streams. In *Proceedings of the Web Conference 2021*, 3371–3382.

[4] Vitalik Buterin. 2013. Ethereum white paper. *GitHub Repository* 1, (2013), 22–23.

[5] Miguel Castro and Barbara Liskov. 1999. Practical byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design And Implementation*, Vol. 99, 173–186.

[6] China Everbright Bank. 2024. *Blockchain Forfaiting Transaction Platform*. Retrieved August 29, 2024 from http://www.cebbank.com/site/ceb_homepage33/corporate_banking1/trade_finance47/146085353/index.html

[7] Mampi Devi and Abhishek Majumder. 2021. Side-channel attack in Internet of Things: A survey. In *Applications of Internet of Things: Proceedings of ICCCIOT 2020*. Jyotsna K. Mandal, Somnath Mukhopadhyay and Alak Roy (Eds.), Springer, 213–222. Retrieved from https://www.amazon.com/Applications-Internet-Things-Proceedings-Networks/dp/9811562008

[8] Isvani Frias-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustin Ortiz-Diaz, and Yailé Caballero-Mota. 2014. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering* 27, 3, (2014), 810–823.

[9] GE. 2024. *Brilliant Manufacturing from GE Digital*. Retrieved August 29, 2024 from https://www.ge.com/digital/sites/default/files/download_assets/Brilliant-Manufacturing-Solutions-for-Conveyor-and-Assembly-Operations.pdf

[10] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. 2016. Robust random cut forest based anomaly detection on streams. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2712–2721.

[11] Ki Hyun Kim, Sangwoo Shim, Yongsub Lim, Jongseob Jeon, Jeongwoo Choi, Byungchan Kim, and Andre S. Yoon. 2019. Rapp: Novelty detection with reconstruction along projection pathway. In *Proceedings of the International Conference on Learning Representations*.

[12] Edwin M. Knox and Raymond T. Ng. 1998. Algorithms for mining distancebased outliers in large datasets. In *Proceedings of the International Conference on Very Large Data Bases*. Citeseer, 392–403.

[13] Neal Koblitz, Alfred Menezes, and Scott Vanstone. 2000. The state of elliptic curve cryptography. *Designs, Codes and Cryptography* 19, (2000), 173–193.

[14] Chieh-Hsin Lai, Dongmian Zou, and Gilad Lerman. 2019. Robust subspace recovery layer for unsupervised anomaly detection. arXiv:1904.00152. Retrieved from https://arxiv.org/abs/1904.00152

[15] Viktor Losing, Barbara Hammer, and Heiko Wersing. 2016. KNN classifier with self adjusting memory for heterogeneous concept drift. In *Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 291–300.

[16] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2018. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering* 31, 12, (2018), 2346–2363.

[17] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. 2016. LSTM-based encoder-decoder for multi-sensor anomaly detection. arXiv:1607.00148. Retrieved from https://arxiv.org/abs/1607.00148

[18] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS)*. IEEE, 1–6.

[19] Gyoung S. Na, Donghyun Kim, and Hwanjo Yu. 2018. Dilof: Effective and memory efficient local outlier detection in data streams. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1993–2002.

[20] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. Retrieved from https://bitcoin.org/

[21] Guansong Pang, Longbing Cao, Ling Chen, and Huan Liu. 2018. Learning representations of ultrahigh-dimensional data for random distance-based outlier detection. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2041–2050.

[22] Mahsa Salehi, Christopher Leckie, James C. Bezdek, Tharshan Vaithianathan, and Xuyun Zhang. 2016. Fast memory efficient local outlier detection in data streams. *IEEE Transactions on Knowledge and Data Engineering* 28, 12, (2016), 3246–3260.

[23] Osman Salem, Khalid Alsubhi, Aymen Shaafi, Mostafa Gheryani, Ahmed Mehaoua, and Raouf Boutaba. 2021. Man-in-the-Middle attack mitigation in internet of medical things. *IEEE Transactions on Industrial Informatics* 18, 3, (2021), 2053–2062.

[24] Vinicius M. A. Souza, Denis M. dos Reis, Andre G. Maletzke, and Gustavo Eapa Batista. 2020. Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery* 34, 6, (2020), 1805–1858.

[25] Alexander Vergara, Shankar Vembu, Tuba Ayhan, Margaret A. Ryan, Margie L. Homer, and Ramón Huerta. 2012. Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical* 166, (2012), 320–329.

[26] Fengqun Wang, Jie Cui, Qingyang Zhang, Debiao He, Chengjie Gu, and Hong Zhong. 2024. Blockchain-based lightweight message authentication for edge-assisted cross-domain industrial internet of things. *IEEE Transactions on Dependable and Secure Computing* 21, 4 (Jul.–Aug. 2024), 1587–1604.

[27] Fengqun Wang, Jie Cui, Qingyang Zhang, Debiao He, and Hong Zhong. 2024. Blockchain-based secure cross-domain data sharing for edge-assisted industrial internet of things. *IEEE Transactions on Information Forensics and Security* 19 (2024), 3892–3905.

[28] Miaomiao Wang, Lanlan Rui, Yang, Yang Zhipeng Gao, and Xingyu Chen. 2022. A blockchain-based multi-CA cross-domain authentication scheme in decentralized autonomous network. *IEEE Transactions on Network and Service Management* 19, 3, (2022), 2664–2676.

[29] Lingyan Xue, Haiping Huang, Fu Xiao, and Wenming Wang. 2022. A cross-domain authentication scheme based on cooperative blockchains functioning with revocation for medical consortiums. *IEEE Transactions on Network and Service Management* 19, 3, (2022), 2409–2420.

[30] Yuhan Yang, Lijun Wei, Jing Wu, Chengnian Long, and Bo Li. 2021. A blockchain-based multidomain authentication scheme for conditional privacy preserving in vehicular ad-hoc network. *IEEE Internet of Things Journal* 9, 11, (2021), 8078–8090.

[31] Susik Yoon, Jae-Gil Lee, and Byung Suk Lee. 2020. Ultrafast local outlier detection from a data stream with stationary region skipping. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1181–1191.

[32] Susik Yoon, Youngjun Lee, Jae-Gil Lee, and Byung Suk Lee. 2022. Adaptive model pooling for online deep anomaly detection from a complex evolving data stream. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2347–2357.

[33] Shuangfei Zhai, Yu Cheng, Weining Lu, and Zhongfei Zhang. 2016. Deep structured energy based models for anomaly detection. In *Proceedings of the International Conference on Machine Learning*. PMLR, 1100–1109.

[34] Yan Zhang, Bing Li, Jiaxin Wu, Bo Liu, Rui Chen, and Jinke Chang. 2022. Efficient and privacy-preserving blockchain-based multifactor device authentication protocol for cross-domain IIoT. *IEEE Internet of Things Journal* 9, 22, (2022), 22501–22515.

[35] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep autoencoding Gaussian mixture model for unsupervised anomaly detection. In *Proceedings of the International Conference on Learning Representations*.

Received 15 October 2024; revised 7 June 2025; accepted 8 July 2025