Hindawi
Security and Communication Networks
Volume 2022, Article ID 6128155, 17 pages
https://doi.org/10.1155/2022/6128155

Research Article

Blockchain-Assisted Distributed Fog Computing Control Flow Attestation

Hongchao Li,[1] **Tao Shen**,[1] **Fenhua Bai**,[1] **and Bei Gong**[2]

[1]*Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Jingming South Street, Kunming 650093, Yunnan, China*
[2]*Faculty of Information Technology, Beijing University of Technology, Nanmofang Street, 100124 Beijing, China*

Correspondence should be addressed to Tao Shen; shentao@kust.edu.cn

Received 3 May 2022; Revised 8 July 2022; Accepted 1 August 2022; Published 28 August 2022

Academic Editor: AnMin Fu

The control flow hijacking attack poses a serious threat to the integrity of the software. The attacker exploits the loophole to hijack the control flow of the running program to achieve the purpose of the attack. Remote control flow attestation is a method for embedded devices to ensure the integrity of the software. With the continuous development of Internet of Things (IoT) technology, embedded devices have exploded. None of the existing control flow attestation schemes can adapt to the real-time attestation requests of such massive embedded devices. This paper proposes a blockchain-assisted distributed fog computing control flow attestation scheme BDFCFA to deal with this scenario. The scheme uses a simplified control flow representation model, which can effectively represent the control flow of the program and reduce the runtime overhead of the prover in the attestation process. We use SGX technology to protect the integrity and confidentiality of verifier and prover data during the attestation process. Our proposed bidirectional control flow attestation protocol based on the elliptic curve can greatly protect the communication security between verifiers and provers without incurring excessive performance overhead and communication cost. We evaluate the performance of BDFCFA through the SNU real-time benchmark and demonstrate that BDFCFA has better performance. Finally, compared to the existing remote control flow attestation scheme, the results show that BDFCFA has the highest security.

1. Introduction

With the rapid development of IoT technology, a large number of embedded devices appear in our lives. A large part of it is deployed in critical information infrastructure and plays an important role. Once these embedded devices are maliciously attacked, it will pose a great threat to our lives. In recent years, control flow hijacking attacks have caused great security threats to embedded devices by tampering with the runtime behavior of programs. Control flow integrity (CFI) [1] was proposed to defend against this threat. The CFI obtains the control flow graph (CFG) of the program by analyzing the normal control flow of the program so that the control flow is transferred within the range limited by the control flow graph, and the execution process of the program is guaranteed to be safe and credible.

Remote attestation is a method of verifying the integrity of software on a remote device. It usually consists of two entities, a verifier who wants to know the state and one or more provers who provide reports of their state. Typically, there is agreement between a verifier and a prover. The verifier accepts a report of the hash value of the running state of the software to be executed signed by the security chip inside the device (such as the Trusted Platform Module, TPM) sent by the prover to verify whether the software state meets expectations. Remote attestation transfers the most expensive part of the entire attestation process to the verifier, thereby reducing the performance overhead of the prover.

In the early years, people used static measurements to verify the state of programs. In this way, the prover obtains a static measurement that is usually a signature or MAC calculated from the hash value of the program code and

sends them to the verifier. However, static attestation cannot detect return-oriented programming (ROP) [2] and jump-oriented programming (JOP) [3] in control flow hijacking attacks. The ROP attack is the most common code reuse attack today. It does not need to call the complete function block but calls the program code segment and the gadget that ends with the ret instruction in the dynamic link library code. By operating the program running stack, the program control flow is controlled so that the program jumps to the corresponding instruction segment when the execution function returns to achieve the purpose of attack. The JOP attack is achieved by using a chain of gadgets that ends with a Jmp instruction. In recent years, it has been proposed to measure and attest the integrity of the program runtime control flow through the prover [4–6] in order to more accurately verify the runtime program control flow. This adds overhead as it gets more contextual information at the basic block level. To balance runtime overhead and control flow security, some people have proposed a mutable control flow attestation scheme based on probability prediction [7] and a granularity adaptive control flow attestation scheme based on Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [8]. A log-based control flow attestation scheme [9] has also been proposed to deal with ROP [2] attacks and uses Physical Unclonable Functions (PUF) as a lightweight root of trust for the prover.

None of the above control flow attestation schemes can adapt to today's explosive growth of embedded devices. They are all single-server and single-database architectures that cannot handle real-time attestation requests from massive embedded devices, and because they are stand-alone databases, they cannot resist a centrally managed database being tampered with. Once the database is tampered with, it will affect the attestation results of the entire system. Moreover, the resources of the embedded devices are limited. Although MGC-FA [7] and GACFA [8] reduce runtime overhead to a certain extent, it comes at the cost of reducing control flow security. We use a simplified control flow representation model to effectively reduce the runtime overhead without reducing the security of the control flow, making the whole scheme more suitable for resource-constrained embedded devices. Moreover, the control flow attestation schemes mentioned above all use the challenge-response method for control flow attestation, and only perform one-way identity authentication, that is, the verifier authenticates the prover, which will cause great security risks.

In recent years, blockchain technology has been widely used in the Internet of Things [10, 11]. A blockchain is a distributed ledger distributed throughout a distributed system where multiple nodes maintain the same information without requiring a central authority. Therefore, this technology can not only mitigate the tampering attack of the centrally managed database but also reduce the communication overhead between the data center and the regional manager.

In this paper, we propose a blockchain-assisted distributed fog computing control flow attestation scheme to alleviate the above problems. The main contributions of this paper are as follows:

(1) We propose a blockchain-assisted distributed fog computing control flow attestation scheme, which can adapt to today's explosively growing embedded devices and mitigate centralized database tampering attacks.

(2) We use fog computing to deploy verifiers and blockchain nodes to the edge of the network, thereby reducing the communication overhead between verifiers and provers and improving the real-time nature of control flow attestation.

(3) We propose a simplified control flow representation model by simply using the <source address, destination address, number of jumps> of the jump instruction to represent the program control flow, thereby effectively representing the control flow of the program and reducing the runtime overhead of the prover during the attestation process.

(4) We propose a lightweight bidirectional control flow attestation protocol based on elliptic curves, which can greatly ensure the communication security between the verifier and the prover and does not generate excessive performance overhead and communication cost.

(5) We use SGX technology to protect the control flow remote attestation, and protect the integrity and confidentiality of the data of the verifier and the prover during the attestation process.

2. Related Work

2.1. Remote Attestation. Remote attestation obtains the running status of software on a resource-constrained prover through a resource-rich verifier. This reduces the attestation overhead for resource-constrained devices. The early remote attestation is mostly based on static, which can only protect the binary code when the program is started, but cannot prevent the control flow hijacking attack during the program execution. Later, the proposed C-FLAT [4] realized a more comprehensive runtime remote attestation of the program, and completed the work that could not be done by static remote attestation. The program control flow is protected from alteration by computing the cumulative hash of the basic blocks of program code. During the running of the program, all control flow instructions are intercepted by the runtime tracker trampoline and transmitted to the safe area for hash operation, and finally a hash value representing the execution state of the current program control flow is obtained. The resulting LO-FAT [5] uses a microcontroller to intercept instructions, instead of using a runtime tracker tool for software to intercept instructions in C-FLAT, allowing C-FLAT to be implemented with a lower performance overhead. ATRIUM [6] is a hardware-based runtime attestation protocol that not only checks the control flow of the program but also checks the specific instructions. It provides resilience against software- and hardware-based Time of Check Time of Use (TOCTOU) attacks while incurring minimal area and performance overhead. Although the hardware-based control flow attestation scheme reduces the

attestation overhead of the prover, the hardware implementation increases the cost and reduces the scalability of the scheme. MGC-FA [7] uses the probability model in machine learning to predict the fragile probability of each function in the program, thus distinguishing the normal part from the fragile part of the program. It is then subjected to lightweight coarse-grained checks and expensive fine-grained checks, respectively, balancing runtime overhead and control flow security. GACFA [8] optimizes the security and performance overhead of the attestation of the control flow of the program using the NSGA-II algorithm and then adaptively performs coarse- and fine-grained checks on the functions in the program according to the optimization results, thus reducing the performance overhead. Liu et al. [9] proposed to record the control flow through a program status log, and the verifier can effectively verify whether the target program has been damaged using the information in the log. At the same time, for more secure storage, they used a lightweight root of trust based on on-chip SRAM Physical Unclonable Functions in the attestation. However, this scheme log records function pointers and function return addresses, so it may be powerless against JOP attacks and attacks against decision data (branch variables or loop variables) in the program.

However, in today's environment of explosive growth of embedded devices, none of these solutions can adapt to new challenges—real-time attestation requests of massive embedded devices and attacks on centralized databases are tampered with. Hence, we use blockchain technology and fog computing to deal with these problems. Moreover, there are certain problems in their attestation protocols, so we propose a lightweight bidirectional control flow attestation protocol based on elliptic curve encryption.

2.2. Blockchain Technology. The blockchain is a chain data structure in which blocks are linked in chronological order and are protected by cryptographic algorithms, and the security of the ledger in the entire blockchain network is jointly maintained by means of distributed accounting. At present, according to the open authority of the network [12], the blockchain system can be divided into the public blockchain, the alliance blockchain, and the private blockchain. In this paper, we mainly study the consortium blockchain that is jointly managed by several different organizations or institutions. Unlike the public blockchain, the entities of the consortium blockchain are no longer a single individual but multiple organizations, so as long as most of the organizations in the consortium obtain a consensus, the data can be operated and managed. In a consortium blockchain, it is not the nodes participating in the chain that have permission to access data, but need to be preapproved by the institution to gain access, and because the consortium blockchain is semi-centralized, it is more efficient than the public blockchain.

In the past few years, due to the rapid development of blockchain technology, blockchain technology has been widely used in a large number of fields, such as finance, medical care, Internet of Things, edge computing, etc., for example, the application of blockchain technology in the field of cross-data center authentication of vehicular fog services (VFSs) [13]. By effectively combining modern cryptography and blockchain technology, the communication between service managers during user authentication is eliminated because the records of all service managers are updated synchronously and can effectively resist the attack of tampered database managed by a central because the public ledger is maintained by all service managers. There are also applications of blockchain technology to distributed data systems IoT [14]. To solve the conflict between the operation performance and security of the blockchain system, the conflict between transparency and privacy, and the compatibility problem of a large number of IoT devices that run together, a distributed data system for IoT based on blockchain technology is proposed. It provides a new system architecture for different industrial IoT devices to deploy high-performance blockchain systems in many scenarios. There is also the application of blockchain technology to the radio frequency identification (RFID) supply chain authentication protocol in the 5G mobile edge computing environment [15]. By applying 5G and blockchain technology to the supply chain, the supply chain process can be simplified and allow automatic payment upon receipt of goods. It will help save the company millions of dollars in operating costs by eliminating the need for distributors who have to handle accounts receivable and pay bill with department personnel to track unpaid invoices. Additionally, this will also help avoid the legal fees that it may incur in disputes. However, our extensive literature survey shows that no one has applied blockchain technology to the realm of program control flow attestation. This article adopts blockchain as the database to improve the security and performance of BDFCFA.

Consensus algorithms are used mainly in distributed systems to ensure data consistency. The consensus algorithm used in blockchain is to solve the "block conflict" problem that may arise when a new transaction block is added to the blockchain. Nowadays, common consensus algorithms in blockchain include proof of work (PoW) [16], proof of stake (PoS) [17], DPoS (delegated proof of stake), practical byzantine fault tolerance algorithm (PBFT) [18] and RAFT algorithm [19], and so on. These five consensus algorithms have their own advantages, and their performance comparisons are shown in Table 1. First of all, since the management departments in BDFCFA are all credible, the consensus algorithm we choose does not need to consider Byzantine fault tolerance, only crash fault tolerance. Second, since the blockchain in BDFCFA is mainly managed by multiple management departments, the selected consensus algorithm does not need to have a high degree of decentralization. Finally, due to the high real-time requirement of control flow attestation, the selected consensus algorithm should have a lower communication complexity, faster verification speed, and higher throughput. After considering the Byzantine fault tolerance, crash fault tolerance, degree of decentralization, communication complexity, verification speed, and throughput of each algorithm, we chose the RAFT algorithm. Although the scalability of the RAFT

Table 1: Comparison of the consensus algorithm.

Consensus algorithm	PoW	PoS	DPoS	PBFT	RAFT
Byzantine fault tolerance	50%	50%	50%	33%	N/A
Crash fault tolerance	50%	50%	50%	33%	50%
Decentralization	High	High	Low	Low	Low
Resource consumption	High	Medium	Low	Low	Low
Scalability	Strong	Strong	Strong	Weak	Weak
Communication complexity	$O(n)$	$O(n)$	$O(n)$	$O(n2)$	$O(n)$
Verification speed	>100 s	<100 s	<100 s	<10 s	<10 s
Throughput (TPS)	<100	<1000	<1000	<2000	>10 k

algorithm is not strong, its resource consumption is low, and its other properties are also very suitable for the high real-time and high security scenarios of control flow attestation.

2.3. Fog Computing. The concept of fog computing was first proposed by Cisco. By allowing devices deployed at the edge of the network to provide computing, storage, and network transmission services for terminal devices in a small area, the efficiency of data analysis and processing can be improved, and latency and network transmission pressure can be reduced. Today, the rapidly developing IoT faces many new challenges, such as strict latency requirements, network bandwidth constraints, resource-constrained devices, etc. [20], which cannot be adequately addressed by today's cloud computing and mainframe computing models alone. Therefore, to adapt to these new challenges, we adopt fog computing in BDFCFA to deploy verifier and blockchain nodes to the edge of the network, thereby improving the real-time performance of control flow attestation and alleviating the performance bottleneck of the central authority.

2.4. Intel SGX Technology. SGX (Software Guard Extensions), the Intel Software Guard Extensions, is a set of instruction sets supported by Intel since 2013 [21]. SGX provides an isolated Trusted Execution Environment TEE (Trusted Execution Environment) called enclave, which protects the safe operation and data of legitimate software from malicious attacks. No one, but the CPU, can access the code and data in it. SGX can now provide a trusted execution environment for many application scenarios, such as by introducing a trusted execution environment in edge computing, that is, software protection extension technology, to ensure the confidentiality of the medical IoT data analysis process [22]. There are also applications of SGX to verifiable confidential cloud computing, which guarantees code and data confidentiality, as well as the correctness and integrity of its results [23]. Confidentiality and integrity are preserved even when large components such as Hadoop, the operating system, and the hypervisor are compromised, and it outperforms Hadoop without SGX protection. There is also the use of SGX technology in the authentication of the IoT end device, enabling shielded execution of measurements and attestation procedures. The sensitive data in the authentication process are hidden through the specific key of the SGX enclave, which ensures the security of the sensitive data [24]. We enable SGX on the verifier and the prover, and the verifier and the prover process the remote control flow attestation related data in the enclave area, such as attestation request, program control flow data, and attestation report. Because the data processing process is carried out in the enclave area provided by SGX, malicious programs outside the enclave area cannot view the data, let alone tamper with them.

2.5. Elliptic Curve. The application of elliptic curves in cryptography first appeared in 1986, proposed by Miller [25] and Koblitz [26] based on the elliptic curve logarithm problem. After that, elliptic curve cryptography (ECC) became a popular research direction in cryptography and gradually became the mainstream of public-key cryptography. The security of ECC relies on the discrete logarithm problem of the elliptic curve (ECDLP), that is, for two points P, Q on an elliptic curve over a finite field, solve k so that $k \cdot P = Q$ holds. On a classical computer, the time complexity of solving the discrete logarithm problem of the elliptical curve is exponential [27], which guarantees the security of the encryption of the elliptical curve. Compared to encryption techniques such as DSA, RSA, DH, etc., ECC is a more efficient security encryption technique that uses a shorter key to provide the same level of security. Therefore, ECC is often used in the authentication of the identity of devices constrained by resources, such as the Internet of Things. For example, Mahmood et al. [28] applied elliptic curve encryption to the communication authentication of smart grid. Their scheme not only provides mutual authentication with low computational and communication costs but is also resistant to most attacks, while also providing anonymity and privacy. However, Sadhukhan et al. [29] proposed an ECC-based lightweight remote user authentication scheme for IoT devices and remote user authentication, which uses three-factor authentication to protect user privacy and data confidentiality from multiple attacks. Rostampour et al. [30] proposed an efficient scheme to secure communication between IoT edge devices and cloud servers, where the authors used an ECC-based authentication protocol. We design a lightweight two-way control flow attestation protocol based on an elliptic curve. By using elliptic curve encryption, we can effectively ensure the security of communication between the verifier and the prover and at the same time reduce the performance overhead and communication cost as much as possible.

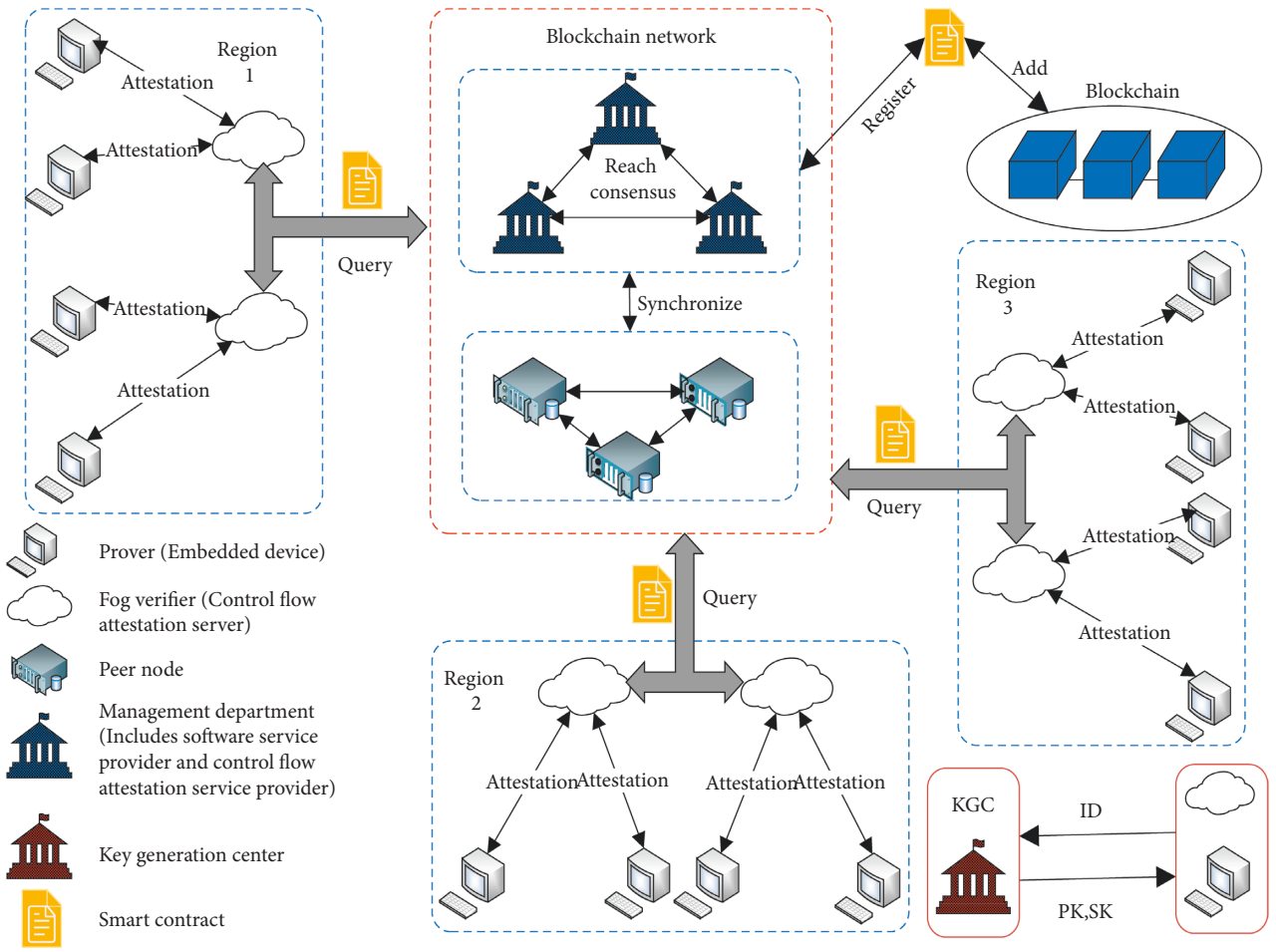

Figure 1: System framework.

3. Problem Statement and Assumptions

At present, the implementation of control flow hijacking attacks mainly includes controlled data attacks and non-controlled data attacks. Among them, control data attacks can be divided into code injection attacks and code reuse attacks according to the source of the attack code. In code injection attacks, the attacker uses the input operation of the software to inject malicious code into the memory space of the target software and then tampers with the control flow data to make the program jump to the malicious code. In code reuse attacks, attackers use existing instructions in programs and shared libraries to achieve attack goals, such as ROP [2] and JOP [3]. Non-controlled data attack is to modify the user identity data, configuration data, user input data, decision data, and other data in the program and then hijack the program control flow to complete the attack, such as (Data-oriented Programming, DOP) [31].

There are two assumptions about our scheme. First, we assume that the embedded device deploys the DEP (Data Execution Prevention) scheme that makes the executable area read-only, which is a built-in protection scheme commonly used on embedded platforms. Therefore, we do not consider code injection attacks. Second, we assume that attackers can carry out code reuse attacks and attacks on decision data (branch variables or loop variables) in the program as well as replay attacks, man-in-the-middle attacks, and impersonation attacks in network attacks. However, the attacker cannot perform any other attack such as pure data attack and physical attack.

4. BDFCFA System Model

Figure 1 shows the system framework designed in this paper, which consists of multiple areas. Each zone includes a peer node, multiple fog verifiers, and multiple embedded devices. There are five types of entities in the system.

4.1. Management Department. The management department is a completely credible institution and is a member of the blockchain network. There are multiple management departments in the entire system. The management sector includes software service providers and control flow attestation service providers. The software service provider is responsible for registering programs that require attestation of the control flow. When registering a program, first instrument the target program, obtain the control flow data of each possible control flow path, and measure the control flow data of each possible control flow path, respectively, to

obtain the expected measurement value. Finally, all the expected measurement values of the program and the program input range corresponding to the expected measurement values are uploaded to the local blockchain through smart contracts, consensus is reached among various management departments, and finally the data are synchronized to each peer node,

$$\begin{cases} H(E_1) = H(E_1, H(0)), & n = 1, \\ H(E_n) = H(H(E_{n-1}), E_n), & n > 1, \\ H = H(H(En), F), & F = \{f_i | i \in n\}. \end{cases} \quad (1)$$

The measurement calculation formula is shown in equation (1), where H is the final measurement value, and $H(0)$ is the initial hash value, which is set by the management department, and is generally 0. E represents the jump edge in the control flow graph, the data in the edge includes the source address of the jump instruction and the destination address of the jump instruction, and $F = \{f_i | i \in n\}$ is the set of execution times of all the jump instructions in the current path (see Section 5.1 for details).

4.2. Key Generation Center (KGC). The key generation center is another trusted entity in the system, responsible for generating a public-private key pair for the fog verifier and prover in BDFCFA, which is used for identity authentication between a fog verifier and prover in control flow attestation. For the specific key generation process, see Section 5.3.

4.3. Peer Node. A peer node is a member of the blockchain network. In BDFCFA, not every fog verifier needs to maintain the blockchain ledger but establishes a peer node in an area to join the blockchain network, reducing the number of nodes in the blockchain network, thereby reducing the cost of deploying the infrastructure. Through smart contract settings, peer nodes cannot push data to the blockchain in the blockchain network but can only query the data on the blockchain, thereby reducing the possibility of peer nodes becoming malicious nodes. Peer nodes provide query services to fog verifiers by providing services.

4.4. Fog Verifier. The fog verifier provides program control flow attestation services for embedded devices within a certain physical range. Before performing control flow remote attestation, fog verifiers need to register with the key generation center. During attestation, the fog verifier needs to query the blockchain for all expected measurement values of the program to be verified and the program input range corresponding to the expected measurement values. The query process is shown in Figure 2.

First, the fog verifier sends a query request to the peer nodes in the area to which it belongs, and then randomly selects to send a query request to 2n nodes in the blockchain network. Finally, after receiving all the results of the request, the final result is determined according to most principles, and according to this result, we attest to the target program. There is a special case, where the randomly selected nodes include the nodes of the management department, and the final result is the result of the node of the management department because the management department is a completely trusted authority. Through such a query mechanism, the harm of attackers tampering with a single database can be greatly alleviated.

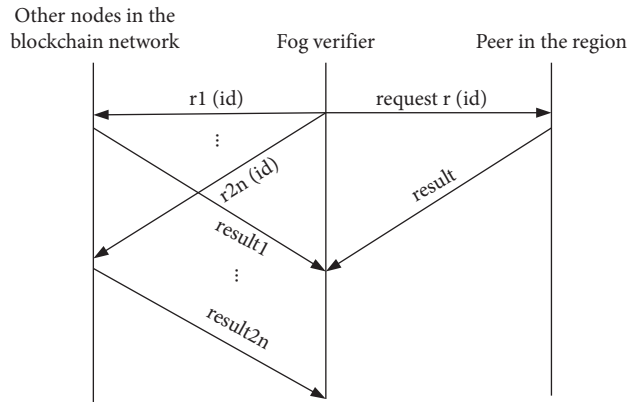

Figure 2: Query mechanism.

4.5. Prover. The prover is an embedded device that needs to attest the control flow of the program running on it. It can be industrial control equipment and other important embedded devices deployed in critical information infrastructure. The program on the prover is provided by the software service provider. Before remote attestation of the control flow, it needs to be registered with the key generation center.

5. BDFCFA Design

5.1. Simplified Control Flow Representation Model. To authenticate the program control flow, the verifier needs to measure the program running path on the prover. However, it is obviously not feasible for the prover to directly transmit every executed instruction to the verifier. This requires the prover to store lengthy control flow data, resulting in a large performance overhead and attestation delay. To reduce the complexity of program control flow representation, thereby reducing the performance overhead and attestation delay of the prover, this paper proposes a simplified control flow representation model. The model is specifically defined as follows:

Definition 1. The simplified control flow representation model is a directed graph G that represents the control flow of a program, represented by a quadruple $\langle V, S, D, F \rangle$.

Definition 2. $S = \{s_i | i \in N\}$, S is the set of source addresses of all jump instructions in the directed graph G; $D = \{d_i | i \in N\}$, D is the set of destination addresses of all jump instructions in the directed graph G; and $F = \{f_i | i \in N\}$, F is the set of execution times of all jump instructions.

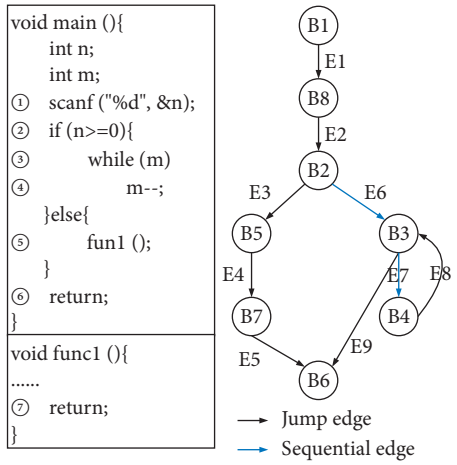

Figure 3: Control flow diagram for a simple program.

Definition 3. $V = \{v_i | i \in N\}$, V is the set of all vertices in the directed graph G, and a node represents a basic block. The basic block is a sequence of code instructions with only one entry and one exit without branching, so the basic block ends with a jump instruction. Hence, s_i is the exit of v_i and d_i is the entrance of v_{i+1}.

Definition 4. : $E = \{v_i \longrightarrow v_{i+1} | i \in N\}$, E is the set of all edges in the directed graph G; there are two types of edges, jump edges and sequential edges. The jump edge indicates that the jump instruction at the exit of the basic block executes the jump, while the sequence edge indicates that the jump instruction at the exit of the basic block does not jump but executes the next instruction. A jump edge is represented by a two-tuple $\langle S, D \rangle$.

Theoretically, the two-tuple $\langle S, D \rangle$ can fully represent the control flow of a program. However, a large number of basic block jump edges will be generated by loops and recursive calls during program operation, resulting in a large number of repetitions of control flow data information and increasing performance overhead. Therefore, when we represent the control flow, we choose to use the triple $\langle S, D, F \rangle$ to represent it.

Taking Figure 3 as an example, we explain the control flow model and its measurement results.

We can see $V = \{B1, B2, B3, B4, B5, B6, B7, B8\}$ from Figure 3, where $B8$ is the basic block of the scanf function that contains the return instruction, and there is only one basic block of $B7$ in the func1 function. $E = \{E1, E2, E3, E4, E5, E6, E7, E8, E9\}$, where $E6$ and $E7$ are sequential edges.

As shown in Figure 3, the control flow graph has two paths, and their expected measurement results are as follows:

$$\begin{aligned} &B1 \longrightarrow B2 \longrightarrow B5 \longrightarrow B6\text{: } H_1 = H\left(H\left(H\left(H\left(H\left(E1, H\left(0\right)\right), E2\right), E3\right), E4\right), E5\right), F_1\right), \\ &B1 \longrightarrow B2 \longrightarrow B3 \longrightarrow B6\text{: } H_2 = H\left(H\left(H\left(H\left(E1, H\left(0\right)\right), E2\right), E8\right), E9\right), F_2\right), \end{aligned} \tag{2}$$

where F_1 and F_2 are the set of execution times of all jump instructions in the two paths, respectively. Since in the second path, $E6$ and $E7$ are sequential edges and no jumps are performed, they are not added to the measurement calculation. The specific measurement calculation formula is mentioned in Section 4.

Then, we use the simple program shown in Figure 3 as an example to demonstrate the detection of five types of attacks, including the four types of attacks mentioned in Section 3:

(1) ROP attack: the attacker tampered with the return address of the function func1 so that the jump edge $E5$, which should have returned $B7$ from the exit of the node to the entry of the $B6$ node, pointed to an illegal malicious code address. Therefore, the destination address of the edge $E5$ will be different from expected, thus detecting that the program has been attacked by control flow hijacking.

(2) JOP attack: the attacker tampered with the jump address of the Jmp class instruction so that the jump edge $E3$, which should have jumped from the exit of node $B2$ to the entrance of node $B5$, pointed to an illegal malicious code address. Therefore, the destination address of the edge $E3$ will be different from expected, thus detecting that the program has been attacked by control flow hijacking.

(3) Branch variable attack: the attacker tampers with the prover's input so that the sequential edge that should be executed sequentially from node $B2$ to node $B3$ becomes a jump edge to jump to the entry of node $B5$, causing the control flow to enter an unexpected but legal path. However, the input of the prover does not conform to the program input range corresponding to this legal path, so it can also be detected that the program has been attacked by control flow hijacking.

(4) Loop variable attack: the attacker altered the value of the loop control variable m, causing the number of loops to change, resulting in a different number of $E8$ than expected. Finally, it was detected that the program was attacked by control flow hijacking.

(5) Function pointer attack: the attacker tampers with the code pointer so that the jump edge $E1$, which should be called from the exit of node $B1$ to the entry of node $B8$, points to an illegal malicious code

address. The destination address of the edge $E1$ will be different than expected, thus detecting that the program has been attacked by control flow hijacking.

In summary, our method can detect these five attacks.

5.2. SGX-Based Control Flow Attestation. As shown in Figure 4, SGX-based control flow attestation includes two roles of the fog verifier and prover. Each role is divided into a secure area and an insecure area. The SGX enclave is the security area inside the role. We deploy the request generation part and the report verification part of the fog verifier in the SGX enclave. Guarantee the security of the fog verifier request and report the verification process and key information during the remote attestation process of the control flow. At the same time, we deploy the runtime tracking part of the program in the unsafe area in the prover, and deploy the authentication, counting, measurement, and report generation part in the SGX enclave to protect the security of the attestation report, key information, and control flow measurement process.

First, the fog verifier generates the attestation request and session key in the request generator and sends the session key to the report verifier and then the attestation request to the prover through the network communicator. The request contains the timestamp, program id, program input in, authentication information, and auxiliary information of encrypted session key. After the prover's network communicator receives the attestation request, it forwards the request to the authenticator. The authenticator first verifies whether the timestamp meets a certain threshold, otherwise the session is terminated. Then verify the identity information of the fog verifier and ensure that the verification does not pass the session termination. Finally, the auxiliary information of the session key will be decrypted through the attestation request to calculate the session key, which will be forwarded to the report generator, and the runtime tracker will execute the corresponding program and instrument program according to the program id and program input in the attestation request. When the application runs to the jump instruction, the interceptor in the runtime tracker will intercept it, and then judge whether the instruction is executed for the first time; if so, send the instruction address and destination address to the measurer. The measurer performs hash operation according to formula (1); if not, it sends a signal that the instruction count is incremented by one to the counter, and the counter increments the execution times of the target instruction by one according to the signal. When the program ends, the counter sends the set of execution times of the jump instruction to the measurer for the final hash operation to obtain the final measure, which is then sent to the report generator. The report generator generates an attestation report and sends it to the fog verifier through the network communicator. The fog verifier's network communicator receives the attestation report and sends it to the report verification. The report verification first verifies whether the timestamp meets a certain threshold, otherwise the session is terminated. Then verify the identity information of the prover, and the verification does not pass the session termination. Then use the session key calculated by the request generator to verify the information in the attestation report, and if the verification fails, the session is terminated. Finally, query the blockchain for the expected measurement value corresponding to the input of the program to be verified and check whether the measurement value calculated according to the attestation report is the same as the expected measurement value. The specific calculation process in the above process is shown in Section 5.3.

The key modules are described in the following:

(1) Request generator: this part uses the fog verifier's private key, the prover's public key, timestamp, program id, program input, and a random number to generate the attestation request and session key and send to the network communicator and report verification device, respectively.

(2) Report verification: authenticates the attestation report received by the network communicator from the prover and verifies whether the control flow measurement value meets the expectations, thereby determining whether the program on the prover is subject to a control flow hijacking attack.

(3) Network communicator: the part of the fog verifier and prover that performs network communication, providing network communication and data forwarding functions for modules in the SGX enclave.

(4) Authenticator: the part of the prover that verifies the identity of the fog verifier. The identity of the fog verifier is verified through the attestation request, and the session key is further calculated to generate the attestation report.

(5) Runtime tracker: this part is used to track the target application. We rewrite the Pin tool in Intel Pin-3.15 as a tool for intercepting jump instructions when the program is running.

(6) Interceptor: intercepts the jump instruction when the target application is running, and judges whether it is a jump instruction that has been intercepted. If it is, it will send a signal to the counter that the number of executions of the instruction is incremented by one. If not, it will intercept the received address information of the instruction being sent to the measurer for measurement.

(7) Counter: the part that obtains the execution times of the jump instruction and puts it in the enclave container to protect the execution times of the jump instruction from being tampered with. Because the address information will be hashed after interception and the set of execution times needs to be completed after the program runs, the counter is deployed in the safe area.

(8) Measurer: when a hash operation is performed in the enclave container, the security of the measurement process is guaranteed. After the final measurement h is calculated, it is sent to the report generator.

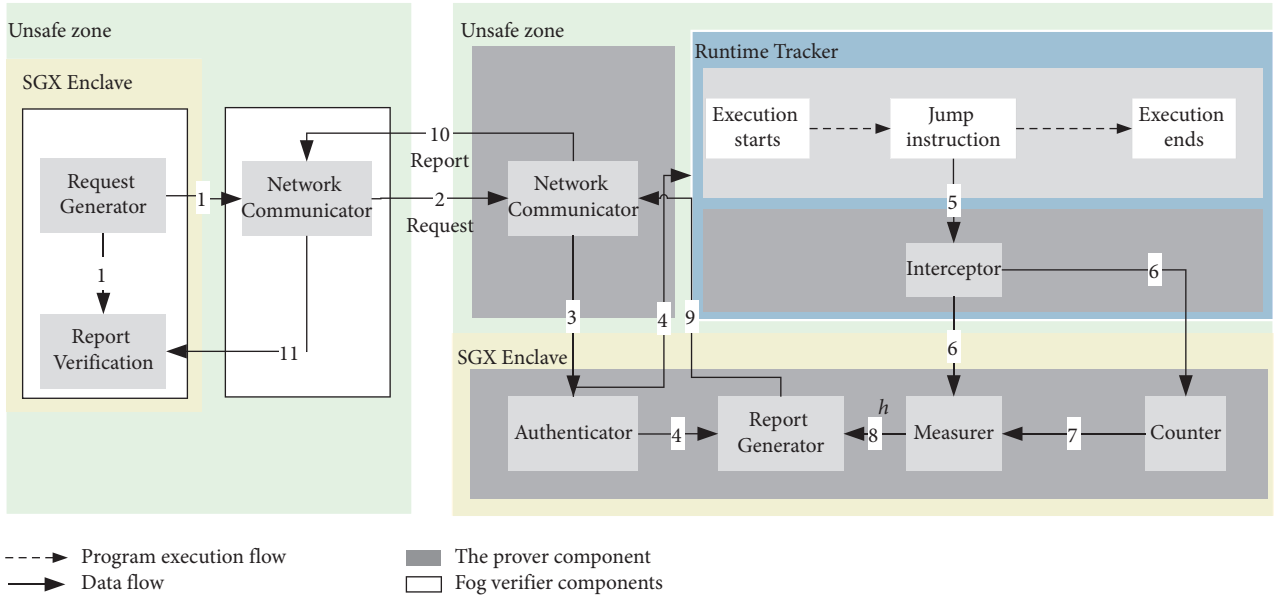

Figure 4: SGX-based control flow attestation architecture.

Table 2: Description of the notation.

Notations	Description
p, q	Large prime numbers
GF(p)	A finite field over prime p
$E_p(a, b)$	An elliptic curve over GF(p)
G	An additive cyclic group of points on the elliptic curves with prime order q
P	A generator of G
$k{\cdot}P$	Multiplication of elliptic curves, $k{\cdot}P = P + P + \cdots + P$ (k times), where $k \in Z_q^*$ and $P \in E_p(a, b)$
Z_q^*	A finite additive group of nonzero integers modulo q
Z_p	A finite additive group of integers modulo p
||,⊕	Data concatenation and bitwise XOR operations
$H1$, $H2$, $H3$	Secure hash function
ID_v, ID_p	Unique identities of the fog verifier and prover
PK_i	The public key of i
SK_i	The private key of i
Comp (M, N)	Compare if M and N are the same

(9) Report generator: using the prover's private key, the fog verifier's public key, and the session key calculated by the authenticator, an attestation report is generated and sent to the verifier through the network communicator.

5.3. A Lightweight Bidirectional Control Flow Attestation Protocol Based on Elliptic Curve

5.3.1. Protocol Scheme Design. This section proposes a lightweight bidirectional control flow attestation protocol scheme based on elliptic curves. The scheme is divided into three stages: initialization stage, registration stage, and attestation stage. The initialization stage is executed only once by the KGC during the system establishment process. The symbols used in this paper are shown in Table 2.

(1) Initialization stage: in this stage, the KGC performs the following operations to initialize a set of system parameters.

(1) Set a system security parameter k and generate two large security prime numbers p and q.
(2) Choose an elliptic curve $y^2 \equiv x^3 + ax + b \pmod p$, denoted by $E_P(a,b)$, where $a, b \in Z_p$ and $4a^3 + 27b^2 \bmod p \neq 0$.
(3) Choose an additive cyclic group G of order q, and choose a generator P for G, which contains the elliptic curve points defined by $E_P(a,b)$.
(4) Define the following secure hash functions: $H1$: $\{0,1\}^* \longrightarrow Z_q^*$, $H2$: $\{0,1\}^* \times G \longrightarrow Z_q^*$, and $H3$: $\{0,1\}^* \longrightarrow \{0,1\}^*$.
(5) Public system parameter $(q, E_P(a,b), H1, H2, H3)$.

(2) Registration stage: this stage completes the registration of fog verifiers and provers, as shown in Figure 5. It is executed only once in the lifetime of each fog verifier and prover. The fog verifier registration process will do the following. The prover registration process is similar to that of the fog verifier, which will not be described here.

(1) The fog verifier uses the hardware information on the device (such as CPU serial number, motherboard serial number, hard disk serial number, physical MAC, etc.) to calculate its unique ID_V, and transmits it to the key generation center through a secure and trusted channel.

(2) After receiving the unique identity ID_V of the fog verifier, the key generation center randomly selects an integer R, connects ID_V and R, and uses the formula $SK_V = H1(ID_V \| R)$ to calculate the private key of the fog verifier $SK_V, SK_V \in Z_q^*$. Then, the public key PK_V of the fog verifier is calculated by $PK_V = SK_V \bullet P$. Finally, the key generation center transmits PK_V and SK_V to the fog verifier using a secure and trusted channel.

(3) After the fog verifier receives the keys PK_V and SK_V generated by the key generation center, the registration process ends.

(3) Attestation stage: this stage completes the attestation of the program control flow on the prover by the fog verifier, as shown in Figure 6. Among them, the fog verifier attests the operation of the program on the prover through a series of operations and judgments, that is, judges whether the running path of the program on the prover is safe and credible. The attestation protocol stipulates that both the fog verifier and the prover can access the binary of the program, and through the traditional static attestation protocol, the program being executed on the prover is guaranteed to be unmodified and complete. The specific process of the attestation stage is as follows:

(1) The fog verifier first calculates the shared key DHK used for identity authentication according to $DHK = PK_p \bullet SK_v$, and then randomly generates a number n, $n \in Z_q^*$. N is calculated by elliptic curve multiplication $N = n \bullet P$, and session key S is calculated using $S = N \bullet SK_v$. Use $h1 = H3(in \| T \| id)$ to hash the program id, program input in, and current timestamp T1 to get $h1$. Then calculate the XOR of $h1$, n and DHK, $C1 = h1 \oplus n \oplus DHK$, and then perform hash operation $C2 = H2(C1, S)$ on C1 to get C2. Finally, the input in of the program, timestamp T1, program id, and identity of the authentication information C1 and C2 are spliced together to form an attestation request Req, which is sent to the prover.

(2) After the prover receives the attestation request Req from the fog verifier, it first checks the freshness of the timestamp. If the difference between timestamps exceeds a certain threshold, the session abruptly ends. Then use the secure hash function $H3$ to calculate $h1 = H3(in \| T \| id)$, and then calculate the shared key DHK' used for identity authentication through $DHK' = PK_v \bullet SK_p$. Through $n' = C1 \oplus h1' \oplus DHK'$, n' is calculated, which is used to calculate the session key. Use $S' = n \bullet PK_v$ to calculate the session key S', and then perform a hash operation $C2' = H2(C1, S')$ on C1 to obtain C2'. Then $Comp(C2, C2')$ compares whether C2 and C2′ are the same; if not, the session is terminated. So far, the prover has completed the identity authentication of the fog verifier. Then the prover executes the target program aid according to the program id and the program input in and obtains the address information $[Src_0, Dest_0], \cdots, [Src_n, Dest_n]$ and the set F of the execution times of the jump instruction. Then, according to formula (1), the secure hash function $H3$ is used to calculate the cumulative hash value h through the address information $[Src_0, Dest_0], \cdots, [Src_n, Dest_n]$ of the jump instruction and the set F of execution times of the jump instruction. Then use $hs = h \oplus S'$ to encrypt the final measurement value h to calculate hs, and then perform hash operation $Cr = H2(T2 \| h, DHK')$ on the timestamp T2 and the final measurement value h to obtain Cr. Finally, the timestamp T2, the encrypted measurement value hs' and the identity authentication information Cr are spliced together to form an attestation report Rep, which is sent to the fog verifier.

(3) After the fog verifier receives the attestation report, it first checks the freshness of the timestamp. If the difference between timestamps exceeds a certain threshold, the session abruptly ends. Then use the session key S to decrypt the final measurement value h' according to $h\imath = hs \oplus S$, then use the secure hash function $H2$ to calculate $Cr' = H2(T2 \| h', DHK)$ to get Cr', then $Comp(Cr, Cr')$ compares Cr and Cr' to see if they are the same, and if they are different, the session is terminated. So far, the fog verifier has completed the identity authentication of the prover. Then the fog verifier queries the blockchain for the expected measurement value $h_{expected}$ corresponding to the input in of the program to be verified, and finally $Comp(h_{expected}, h')$ compares whether $h_{expected}$ and h' are consistent; if they are consistent, it means the target program aid is not subject to control flow hijacking attacks.

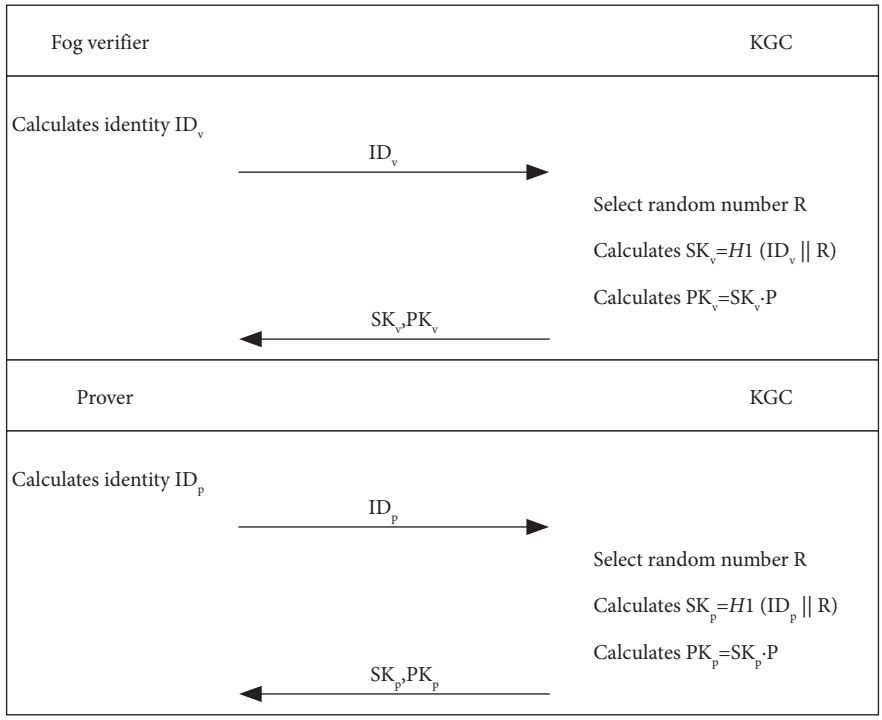

Figure 5: Registration process.

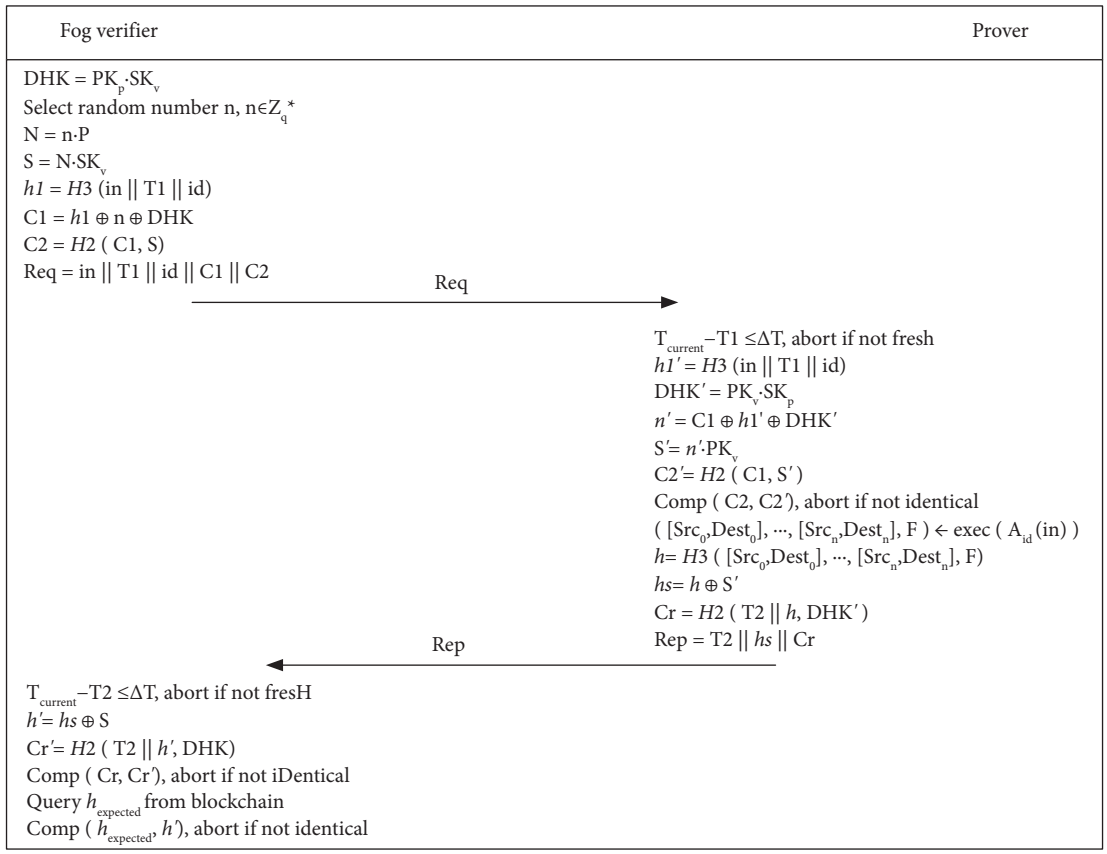

Figure 6: Attestation process.

Table 3: Parameters of the Hyperledger Fabric.

Consensus algorithm	Batch timeout (s)	Maximum message count	Block size (kB)	Number of order nodes	Number of peer nodes
RAFT	2	10	512	3	7

Table 4: Query time of the query mechanism under different peer nodes.

Query number of peer nodes	1	3	5	7
Average query time (ms)	11.894	23.976	37.895	51.682

Table 5: Runtime performance comparison of the two schemes.

Program	BDFCFA		Control flow events for MGC-FA			Control flow events for GACFA
	Control flow events	Runtime overhead (ms)	$p = 0$	$p = 0.3$	$p = 1$	
adpcm-test.c	509	317.630	2×106	1.1×105	1×105	151
fft1k.c	329	18.392	3.6×105	2.7×105	3.3×104	49
fir.c	338	69.569	7800	2400	734	61
lms.c	353	24.363	1.2×105	4.4×104	8912	73
ludcmp.c	367	42.759	571	410	14	17
qurt.c	248	17.900	248	202	14	15
minver.c	310	16.204	310	300	6	25
fft1.c	381	31.720	884	776	136	58
sqrt.c	296	13.189	296	4	4	10

5.3.2. Protocol Security Analysis. The lightweight bidirectional control flow attestation protocol based on the elliptic curve in this paper has the following security features:

(1) Two-way authentication: the protocol proposed in this paper realizes two-way authentication between the fog verifier and the prover, and the identities of both parties are authenticated before the control flow attestation. Both the fog verifier and the prover use their own private key and the other party's public key when calculating the shared key *DHK*. It is almost impossible for an attacker to calculate the shared key *DHK* only knowing their public key. The prover verifies the fog verifier by verifying whether C2 and C2' are the same. The fog verifier verifies the prover by verifying whether Cr and Cr' are the same. The calculation of C2 and Cr will use *DHK*. Therefore, as long as the public key infrastructure is secure, only legitimate fog verifiers and provers can perform control flow attestation.

(2) Anti-impersonation attack: the private key SK_V of the fog verifier is strictly kept secret. Although the public key PK_V and the generator P are open to the public, it is extremely difficult to calculate the private key SK_V from the public key PK_V and the generator *P*, which belongs to solving the elliptic-curve discrete logarithm problem. It is also extremely difficult for an attacker to directly calculate the private key SK_V through the ID_v of the fog verifier. Because a random number *R* is added when calculating the private key SK_V, it is almost impossible to calculate the private key SK_V without knowing the random number *R*. The prover is similar to the fog verifier, which will not be described here. Therefore, in the protocol proposed in this paper, attackers cannot pretend to be fog verifiers and provers and can prevent impersonation attacks.

(3) Anti-man-in-the-middle attack: when A and B communicate, the attacking host *C* becomes a forwarder in the middle, and the information between them is forwarded by *C*. *C* can not only eavesdrop on the communication of A and B but also tamper with the information. Then pass it on to the other party. We assume that the middleman C intercepts the attestation request Req = in ||T1 || id ||C1 ||C2 and forwards it after maliciously tampering with the attestation request. If the middleman tampers with the data in in, T1 or id, then the prover will not be able to get the expected value by calculating $h1' = H3(\text{in}\|T\|id)$, resulting in failure to compare C2 and C2'. If the middleman tampered with the data of C1, the prover will not be able to calculate the correct n' through n' = C1⊕h1'⊕DHK', resulting in the termination of the session. If the middleman tampered with the data of C2, it will directly lead to the failure of comparing C2 and C2'. Similarly, if the middleman C intercepts the attestation report Rep = T2 || hs || Cr, and forwards it to the fog verifier after maliciously tampering with the attestation report. Assuming that the middleman has tampered with the data of T2, the fog verifier will make an error in calculating $Cr' = H2(T2\|h', \text{DHK})$, resulting in a failure to compare Cr and Cr'. If the middleman tampered with the data of *hs*, the fog verifier will not be able to decrypt the final measurement value h' with h' = hs⊕S, resulting in the termination of the session. If the middleman tampered with the Cr data, it will directly lead to the failure of comparing Cr and

Cr'. Therefore, for the protocol proposed in this paper, the attacker cannot achieve the purpose of the attack through the man-in-the-middle attack.

(4) Anti-replay attack: the replay attack is that the attacker sends an authentication information that the destination host has received to deceive the other party. In the protocol proposed in this paper, both the attestation request and the attestation report contain the timestamp T, which is not only sent in clear text but is also hidden in C1 and C2 in the attestation request and Cr in the attestation report. Therefore, if an attacker replays the attestation request from the fog verifier or the prover's attestation report, the fog verifier and the prover can identify it by checking the freshness of T. If the attacker replaces a new timestamp T' in the attestation request or attestation report, the identity authentication of the prover or fog verifier will also fail. Because the prover makes an error when calculating C2, which causes the authentication to fail. The same applies to fog verifiers. The protocol proposed in this paper ensures that the attestation request or report of each transmission is different and there is no leakage of any valuable information, so the attacker cannot deceive the other party by replaying the intercepted message.

(5) Known session key security: the fog verifier uses the random number n, $n \in Z_q^*$ to calculate N, and then uses the private key SK_v and N to calculate the session key S. Since n for each calculation of the session key is randomly selected, the attacker cannot obtain other session keys through the known session key. It is not even possible to obtain any valid information from the known session key because the session key needs to use the private key SK_v of the fog verifier in addition to N, and it is almost impossible to obtain any valid information only knowing S.

6. Evaluation

6.1. Query Mechanism Performance Evaluation. We use Hyperledger Fabric version 1.4.12 to build a blockchain network for query mechanism performance evaluation. The configuration of the blockchain network built by Hyperledger Fabric is shown in Table 3.

In this network, we use fog verifiers to randomly query 1, 3, 5, and 7 peer nodes 45 times, respectively, and get the average query time, as shown in Table 4. It can be seen that although the increase in the number of query peer nodes will increase the query time, in fact, compared with the entire attestation process, this overhead is not large. During the attestation process, we ask fog verifiers to randomly query 3 peer nodes in the blockchain network. The average attestation overhead of obtaining fog verifiers after 45 remote attestations is 29.50 ms. The overhead of attestation includes the cost of the fog verifier to generate the attestation request and the cost of the fog verifier to verify the attestation report, which includes the query time. This overhead is acceptable for fog verifiers. In summary, our query mechanism does not incur too much overhead, while mitigating the harm of database tampering by attackers.

6.2. Runtime Overhead Evaluation. Since our BDFCFA scheme is aimed at embedded devices, to evaluate the performance of BDFCFA, we use the SNU real-time benchmark [32] to test BDFCFA. There are many C files in this benchmark, such as adpcm-test.c and fft1.c for embedded platforms. In addition, since the runtime overhead is mainly determined by the number of control flow events, we mainly compare the number of control flow events for each scheme when evaluating the runtime overhead.

The experimental results of the runtime overhead are shown in Table 5. We compare the MGC-FA [7] and GACFA [8] with our BDFCFA scheme. The experimental data on the runtime overhead of MGC-FA come from the literature [7]. The scheme with $p = 0.3$ in MGC-FA is the scheme proposed in the literature [7]. The scheme when $p = 0$ is the same as the control flow checking scheme in the literature [33], and the scheme when $p = 1$ is the same as the control flow checking scheme in the literature [8]. Runtime overhead is the average value obtained after the experiment is repeated many times. As can be seen from the table, the number of BDFCFA control flow events is less than or equal to the number of control flow events when the probability threshold $p = 0$ of the MGC-FA scheme. Among them, the number of control flow events of the three programs adpcm-test.c, fft1k.c, and lms.c is very different and is not of the same order of magnitude. Although the probability threshold for the MGC-FA scheme is $p = 0.3$ or 1, there are some programs with fewer control flow events than our BDFCFA. However, when the probability threshold is $p = 0.3$ or 1 of the MGC-FA schemes, they reduce the number of control flow events by sacrificing the control flow security, thereby reducing the runtime overhead. Because the number of control flow events directly affects the timeto check and measure control flow events, and the time overhead of checking and measuring control flow events accounts for the vast majority of the runtime overhead. Similarly, GACFA makes its control flow events smaller than BDFCFA on the premise of sacrificing control flow security. In summary, our BDFCFA scheme effectively reduces runtime overhead without sacrificing control flow security, making the whole scheme more suitable for resource-constrained embedded devices.

6.3. Attestation Protocol Performance Evaluation. Since our attestation protocol is used for control flow attestation, we compare the performance of the challenge-response based unidirectional control flow attestation protocol used in [4–8] with our elliptic curve-based bidirectional control flow attestation protocol. We use two attestation protocols separately in the BDFCFA scheme for comparison. Among them, the elliptic curve used by the bidirectional control flow attestation protocol based on the elliptic curve is 256 bits, and the secure hash function is also 256 bits. The challenge-response based unidirectional control flow attestation

Table 6: Comparison of communication cost theory.

Protocol	Communication costs	Communication bandwidth consumption (byte)
The elliptic curve-based bidirectional control flow attestation protocol	Req + Rep	144
The challenge-response-based unidirectional control flow attestation protocol	$C + R$	120

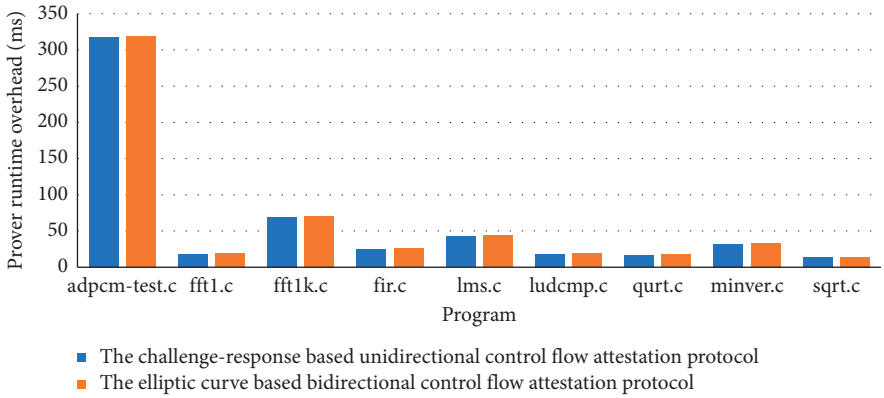

Figure 7: Comparison graph of runtime costs for both protocol provers.

protocol uses 256-bit elliptic curve based ECSDA and 256-bit hash function.

6.3.1. Communication Cost Performance Analysis. First, we theoretically analyze the communication cost of the two protocols. In the challenge-response-based unidirectional control flow attestation protocol, the verifier first needs to send a challenge C containing the program id, the program input in, and the random number N to the prover. The prover needs to return an attestation report R containing the expected measurement value h, the challenge C, and the digital signature S obtained by signing the expected measurement value and the challenge. In the bidirectional control flow attestation protocol based on the elliptic curve proposed in this paper, the fog verifier first needs to send an attestation request Req containing the input *in* of the program, the timestamp T1, the id of the program, and the identity authentication information C1 and C2. The prover needs to return an attestation report containing the timestamp T2, the encryption measurement value *hs*, and the identity authentication information Cr. The theoretical communication costs of the two protocols are shown in Table 6.

We assume that the size of program id, program input in, timestamp, and random number N are all 4 bytes, so the size of challenge C is 12 bytes. Since the bidirectional control flow attestation protocol based on an elliptic curve uses a 256-bit elliptic curve, the sizes of C1, C2, and Cr are all 32 bytes, so the size of the attestation request Req is 76 bytes and the size of the attestation report Rep is 68 bytes. Therefore, the communication bandwidth consumption of the bidirectional control flow attestation protocol based on the elliptic curve is 144 bytes. The challenge-response-based unidirectional control flow attestation protocol uses 256-bit elliptic curve-based ECSDA and a 256 bit hash function, so the size of the attestation report R is 108 bytes. The communication bandwidth consumption of the challenge-response-based unidirectional control flow attestation protocol is 120 bytes. It can be seen from the data that the communication bandwidth consumption of the two protocols is almost the same.

6.3.2. Protocol Operational Efficiency Analysis. We use the SNU real-time benchmark [32] to compare the runtime overhead of the two protocols on the prover. By performing multiple experiments on nine programs selected in the SNU real-time benchmark and averaging them, we obtain a comparison chart of the runtime overhead of the two protocol provers, as shown in Figure 7. It is obvious from the figure that the prover's runtime overhead difference between the bidirectional control flow attestation protocol based on the elliptic curve and the unidirectional control flow attestation protocol based on the challenge response is very small, only one to two milliseconds. Therefore, the runtime overhead of the two protocols for the prover is almost the same.

We also compared the attestation time of the verifiers of the two protocols. The attestation time here refers to the time when the verifier generates the attestation request plus the time when the verifier verifies the attestation report, and does not include the time caused by the communication between the verifier and the prover. Any program in the SNU real-time benchmark will only generate an attestation request of the same size for the verifier and will only receive an attestation report of the same size. Therefore, we used the two protocols to perform 45 repeated experiments on the adpcm-test.c program in the SNU real-time benchmark test

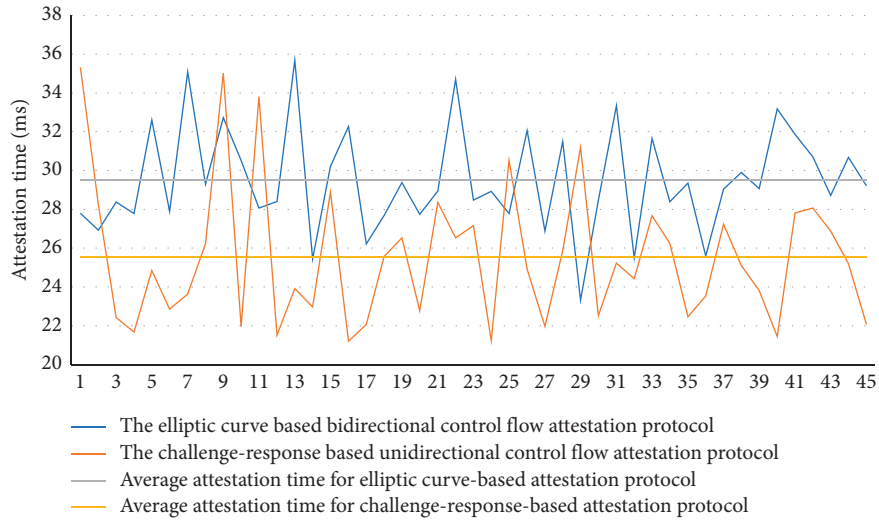

Figure 8: Comparison of the attestation time between the two protocol verifiers.

Table 7: Security comparison.

Security features	SF_1	SF_2	SF_3	SF_4	SF_5	SF_6	SF_7	SF_8	SF_9	SF_{10}
C-FLAT [4]	×	√	√	×	×	√	√	√	√	√
LO-FAT [5]	×	×	√	×	×	√	√	√	√	√
ATRIUM [6]	×	×	√	×	×	√	√	√	√	√
MGC-FA [7] $p=0$	×	√	√	×	×	√	√	√	√	√
MGC-FA [7] $p=1$	×	√	√	×	×	√	×	×	×	√
Liu et al. [9]	√	×	√	×	√	√	×	×	×	√
BDFCFA	√	√	√	√	√	√	√	√	√	√

√: the scheme supports features, ×: the scheme does not support features. SF1: Resist impersonation attacks, SF2: resist man-in-the-middle attacks, SF3: resist replay attacks, SF4: two-way authentication, SF5: known session key security, SF6: detect ROP attacks, SF7: detect JOP attacks, SF8: detect branch variable attacks, SF9: detect loop variable attacks, and SF10: detect function pointer attacks.

and obtained the comparison chart of the attestation time of the verifiers of the two protocols, as shown in Figure 8. It can be seen from the figure that the verifier attestation time of the unidirectional control flow attestation protocol based on the challenge response is shorter than that of the bidirectional control flow attestation protocol based on the elliptic curve. Among them, the average attestation time of the verifier of the bidirectional control flow attestation protocol based on the elliptic curve is 29.50 ms, and the average attestation time of the verifier of the unidirectional control flow attestation protocol based on challenge response is 25.53 ms. The difference between the two is close to 4 ms, and the difference is not big.

By analyzing the communication cost and operation efficiency of the two protocols, the communication cost and operation efficiency of our proposed bidirectional control flow attestation protocol based on the elliptic curve are very small compared to the unidirectional control flow attestation protocol based on the challenge response. However, our proposed protocol is more secure and can greatly ensure the communication security between the verifier and the prover. Therefore, our protocol is more suitable for program remote control flow attestation in the field of software security.

6.4. Security Performance Evaluation. We compare the security of our scheme BDFCFA with some control flow remote attestation schemes in recent years, as shown in Table 7. Since the security of the GACFA [8] scheme is related to specific procedures, there is no security comparison here. Both C-FLAT [4] and MGC-FA [7] can detect all control flow hijacking attacks proposed in the table when the probability threshold $p=0$. However, it is not resistant to impersonation attacks, there is no two-way authentication, and there is no way to maintain the security of the known session key. However, LO-FAT [5] and ATRIUM [6] cannot resist man-in-the-middle attacks except for the security features that C-FLAT does not have. Because the attestation protocols of these two schemes only digitally sign the random number in the attestation request when generating the attestation report, the attacker can completely intercept the attestation request and tamper with the id or input of the program to achieve the purpose of the attack. When MGC-FA has probability threshold $p=1$, in addition to not having the security characteristics of probability threshold $p=0$, it cannot detect the JOP attack, branch variable attack, and loop variable attack in control flow hijacking attacks. The reason for this is because MGC-FA only measures function pointers and function return addresses in the program when

the probability threshold $p = 1$. Liu et al. [9] used PUF as a lightweight root of trust for the prover, which can resist impersonation attacks and achieve known session key security. However, since it only digitally signs the log, it is also not resistant to man-in-the-middle attacks. Moreover, it does not perform two-way authentication and only records the function pointer and function return address in the program in the log, so only the ROP attack and function pointer attack in the control flow hijacking attack can be detected. Our scheme BDFCFA not only can detect all control flow hijacking attacks proposed in the table but also resist network attacks such as impersonation attacks, replay attacks, and man-in-the-middle attacks, and at the same time realizes two-way authentication, ensuring the security of known session keys, and each session key is different.

7. Conclusion

This paper proposes a blockchain-assisted distributed fog computing control flow attestation (BDFCFA), which can not only adapt to today's explosive growth of embedded devices, reduce the communication overhead between the verifier and the prover, and improve the real-time performance of control flow authentication but also mitigate the attack of the centralized database being tampered with. At the same time, we use SGX to protect the integrity and confidentiality of the verifier and prover data during the certification process. In addition, the query mechanism adopted by the fog verifier when querying the measurement data of the program from the blockchain network can spend less time overhead, thereby mitigating the harm of the attacker tampering with the database. Compared to MGC-FA [7], the simplified control flow representation model used in our scheme can effectively represent the control flow of the program under the premise of ensuring the security of the control flow, thus reducing the runtime overhead of the prover in the attestation process. Through comparative experiments with the challenge-response-based unidirectional control flow attestation protocol, it can be inferred that our proposed bidirectional control flow attestation protocol based on the elliptic curve can greatly protect the communication security between the verifier and the prover and does not generate excessive performance overhead and communication costs. This protocol is more suitable for program remote control flow attestation than the challenge-response-based unidirectional control flow attestation protocol used in the program control flow scheme by previous researchers. Finally, by comparing the security of BDFCFA with some remote control flow attestation schemes in recent years, it can be seen that the BDFCFA scheme has the highest security and can better protect the security of program control flow attestation. In summary, BDFCFA can adapt to today's explosive growth of embedded devices, improve the real-time performance of control flow attestation, alleviate the harm of attackers tampering with the database, reduce the runtime overhead of the prover during the attestation process, and greatly protect the security of the communication between the verifier and prover, and does not produce excessive performance overhead and communication cost.

In the future, we will study remote control flow attestation based on the combination of dynamic and static measurements. Because of the current runtime control flow measurement, only the control flow data when the program is dynamically running are measured. However, if the attacker tampers with the binary code of the program on the premise of ensuring the original control flow, the existing control flow remote attestation will not be able to detect this behavior.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare no conflicts of interest in this work.

Acknowledgments

This work was supported in part by the Major Scientific and Technological Projects in Yunnan Province under Grant 202002AB080001-8, the Yunnan Key Laboratory of Blockchain Application Technology under Grant 202105AG070005 and Projects YNB202109 and YNB202115, the Scientific Research Fund Project of Yunnan Provincial Department of Education under Grant 2022Y160, the National Natural Science Foundation of China under Grant 61971208, the Yunnan Reserve Talents of Young and Middle-Aged Academic and Technical Leaders (Shen Tao) under Grant 2019HB005, and the Yunnan Young Top Talents of Ten Thousands Plan (Shen Tao, Zhu Yan, Yunren Social Development) under Grant 2018 73.

References

[1] M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity," in *Proceedings of the ACM Conference on Computer & Communications Security*, p. 340, Alexandria VA USA, November 2005.

[2] H. Shacham, "The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)," in *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 552–561, Alexandria, VA, USA, November 2007.

[3] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang, "Jump-oriented programming: a new class of code-reuse attack," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2011*, pp. 30–40, ACM, Hong Kong, China, March 2011.

[4] T. Abera, N. Asokan, L. Davi et al., "C-FLAT: control-flow attestation for embedded systems software," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 743–754, Vienna, Austria, October 2016.

[5] G. Dessouky, S. Zeitouni, T. Nyman et al., "LO-FAT: low-overhead control flow attestation in hardware,"vol. 24, pp. 1–24, in *Proceedings of the 54th Annual Design Automation Conference 2017, DAC 2017*, vol. 24, pp. 1–24, ACM, Austin, TX, USA, June 2017.

[6] S. Zeitouni, G. Dessouky, O. Arias et al., "ATRIUM: runtime attestation resilient under memory attacks," in *Proceedings of*

the 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 384–391, Irvine, CA, USA, November 2017.

[7] J. Hu, D. Huo, M. Wang, Y. Wang, Y. Zhang, and Y. Li, "A probability prediction based mutable control-flow attestation scheme on embedded platforms," in *Proceedings of the 2019 18th IEEE International Conference on Trust, Security and Privacy Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 530–537, Rotorua, New Zealand, August 2019.

[8] J. Zhan, Y. Li, Y. Liu, H. Li, S. Zhang, and L. Lin, "NSGA-II-Based granularity-adaptive control-flow attestation," *Security and Communication Networks*, vol. 2021, Article ID 2914192, 16 pages, 2021.

[9] J. Liu, Q. Yu, W. Liu, S. Zhao, D. Feng, and W. Luo, "Log-based control flow attestation for embedded devices," in *Cyberspace Safety and Security. CSS 2019*, J. Vaidya, X. Zhang, and J. Li, Eds., vol. 11982, Cham, Springer, 2019 Lecture Notes in Computer Science.

[10] H. -N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for Internet of Things: a survey," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8076–8094, 2019.

[11] Q. Wang, X. Zhu, Y. Ni, L. Gu, and H. Zhu, "Blockchain for the IoT and industrial IoT: A review," *Internet of Things*, vol. 10, pp. 1–9, 2020.

[12] V. Buterin, "On public and private blockchains," 2015, https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/.

[13] Y. Yao, X. Chang, J. Mišić, V. B. Mišić, and L. Li, "BLA: blockchain-assisted lightweight Anonymous authentication for distributed vehicular fog services," *IEEE Internet of Things Journal*, vol. 6, pp. 3775–3784, Article ID 2892009, 2019.

[14] N. Gao, R. Huo, S. Wang, T. Huang, and Y. Liu, "Sharding-hashgraph: a high performance blockchain-based framework for industrial Internet of Things with hashgraph mechanism," *IEEE Internet of Things Journal*, 2021.

[15] S. Jangirala, A. K. Das, and A. V. Vasilakos, "Designing secure lightweight blockchain-enabled RFID-based authentication protocol for supply chains in 5G mobile edge computing environment," *IEEE transactions on industrial informatics*, vol. 16, no. 11, pp. 7081–7093, 2020.

[16] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols (extended abstract)," in *Secure Information Networks*, B. Preneel, Ed., vol. 23, Boston, MA, Springer, 1999 IFIP — the International Federation for Information Processing.

[17] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Advances in Cryptology – CRYPTO 2017. CRYPTO 2017*, J. Katz and H. Shacham, Eds., vol. 10401, Cham, Springer, 2017 Lecture Notes in Computer Science.

[18] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the third symposium on Operating systems design and implementation (OSDI '99)*, pp. 173–186, USENIX Association, USA, 1999.

[19] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proceedings of the 2014 USENIX Annual Technical Conference*, Philadelphia, PA, USA, June 2014.

[20] M. Chiang and T. Zhang, "Fog and IoT: an overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, pp. 854–864, 2016.

[21] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proceedings of the workshop on hardware and architectural support for security and privacy (HASP)*, pp. 1–6, Tel-Aviv, Israel, June 2013.

[22] Y. Gao, H. Lin, Y. Chen, and Y. Liu, "Blockchain and SGX-enabled edge-computing-empowered secure IoMT data analysis," *IEEE Internet of Things Journal*, vol. 8, pp. 15785–15795, 2021.

[23] F. Schuster, M. Costa, C. Fournet et al., "VC3: trustworthy data analytics in the cloud using SGX," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pp. 38–54, San Jose, CA, USA, May 2015.

[24] J. Wang, Z. Hong, Y. Zhang, and Y. Jin, "Enabling security-enhanced attestation with Intel SGX for remote terminal and IoT," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 88–96, 2018.

[25] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology — CRYPTO '85 Proceedings. CRYPTO 1985*, H. C. Williams, Ed., vol. 218, Berlin, Heidelberg, Springer, 1986 Lecture Notes in Computer Science.

[26] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[27] S. D. Galbraith and P. Gaudry, "Recent progress on the elliptic curve discrete logarithm problem," *Designs, Codes and Cryptography*, vol. 78, no. 1, pp. 51–72, 2016.

[28] K. Mahmood, S. A. Chaudhry, H. Naqvi, S. Kumari, X. Li, and A. K. Sangaiah, "An elliptic curve cryptography based lightweight authentication scheme for smart grid communication," *Future Generation Computer Systems*, vol. 81, pp. 557–565, 2018, Pages 557-565, ISSN 0167-739X,.

[29] D. Sadhukhan, S. Ray, G. P. Biswas, M. K. Khan, and M. Dasgupta, "A lightweight remote user authentication scheme for IoT communication using elliptic curve cryptography," *The Journal of Supercomputing*, vol. 77, no. 2, pp. 1114–1151, 2021.

[30] S. Rostampour, M. Safkhani, Y. Bendavid, and N. Bagheri, "ECCbAP: a secure ECC-based authentication protocol for IoT edge devices," *Pervasive and Mobile Computing*, vol. 67, Article ID 101194, 2020.

[31] H. Hu, Z. Leong Chua, S. Adrian, P. Saxena, and Z. Liang, "Automatic generation of data-oriented exploits," in *Proceedings of the 24th USENIX Conference on Security Symposium*, pp. 177–192, Washington, D.C. USA, August 2015.

[32] SATABS, "SNU real-time benchmarks," 2022, http://www.cprover.org/goto-cc/examples/snu.html.

[33] S. Das, W. Zhang, and Y. Liu, "A fine-grained control flow integrity approach against runtime memory attacks for embedded systems," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 24, pp. 3193–3207, Article ID 2548561, 2016.