Security and Safety, Vol. 3, 2024007 (2024)
https://doi.org/10.1051/sands/2024007
sands.edpsciences.org

Security and Privacy for Space-Air-Ground Integrated Networks

Research Article OPEN ACCESS

Software Engineering

Static program analysis for IoT risk mitigation in space-air-ground integrated networks

Haotian Deng[1], Tao Liu[2,3], Xiaochen Ma[1], Weijie Wang[1], Chuan Zhang[1,*], Huishu Wu[4,*], and Liehuang Zhu[1]

[1] *Beijing Institute of Technology, Beijing 100081, China*
[2] *China Academy of Information and Communications Technology, Beijing 100191, China*
[3] *Key Laboratory of Mobile Application Innovation and Governance Technology, Beijing 100191, China*
[4] *China University of Political Science and Law, Beijing 100088, China*

Received: 30 March 2024 / Revised: 28 April 2024 / Accepted: 29 April 2024 / Published online: 30 April 2024

Abstract The space-air-ground integrated networks (SAGINs) are pivotal for modern communication and surveillance, with a growing number of connected devices. The proliferation of IoT devices within these networks introduces new risks due to potential erroneous synergistic interactions that could compromise system integrity and security. This paper addresses the challenges in coordination, synchronization, and security within SAGINs by introducing a novel static program analysis (SPA) technique using zero-knowledge (ZK) proofs. This approach ensures the detection of risky interactions without compromising sensitive source code, thus safeguarding intellectual property and privacy. The proposed method overcomes the incompatibility between SPA and ZK systems by developing an imperative programming language for SAGINs and a specialized abstract domain for interaction threats. The system translates network control algorithms into arithmetic circuits suitable for ZK analysis, maintaining high accuracy in detecting risks. Evaluations of real-world scenarios demonstrate the system's efficacy in identifying risky interactions with minimal computational overhead. This research presents the first ZK-based SPA scheme for SAGINs, enhancing security and confidentiality in network analysis while adhering to privacy regulations.

Keywords Space-air-ground, static program analysis, abstract Interpretation, zero-knowledge proof

Citation Deng H, Liu T, Ma X, Wang W, Zhang C, Wu H and Zhu L. Static program analysis for IoT risk mitigation in space-air-ground integrated networks. Security and Safety 2024; **3**: 2024007. https://doi.org/10.1051/sands/2024007

1 Introduction

The space-air-ground integrated networks (SAGINs) are becoming increasingly significant in the modern era of communication and surveillance [1]. With the anticipated advancements in satellite technology and the growing need for comprehensive coverage, the number of connected devices and systems worldwide is expected to increase exponentially [2]. The emergence of platforms that manage SAGINs, such as advanced air traffic control systems, satellite communication interfaces, and ground-based surveillance technologies, supports complex interactions between different layers of the network, leading to enhanced efficiency and reliability [3].

* Corresponding authors (email: chuanz@bit.edu.cn (Chuan Zhang); wuhuishu0122@gmail.com (Huishu Wu))

This is an Open Access article distributed under the terms of the Creative Commons Attribution License (https://creativecommons.org/licenses/by/4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.
© The Author(s) 2024. Published by EDP Sciences and China Science Publishing & Media Ltd.

However, the increased complexity of these interactions may introduce new challenges in terms of coordination, synchronization, and security [4]. For instance, consider the coordination between satellite-based navigation systems and ground-based radar for accurate air traffic management [5]. A miscommunication or delay in data transmission could lead to critical operational errors, potentially causing collisions or other safety incidents. Similarly, the synchronization of data from multiple satellites for disaster management and emergency response requires precise timekeeping and seamless data exchange, which can be compromised by network latency or interference. Moreover, the security of SAGINs is paramount, as any vulnerabilities could be exploited by malicious actors to disrupt critical services or gain unauthorized access to sensitive information [6].

Static program analysis is a valuable technique for identifying potential issues in the coordination and synchronization of SAGINs, such as risky interactions between airborne, space-based, and ground systems [7]. To detect these risky interactions, a considerable amount of research has focused on revealing incompatibilities and potential conflicts in the interactions between different components of SAGINs, such as verifying the consistency of data exchange protocols and ensuring the robustness of communication links [8].

However, the traditional static program analysis approaches in SAGINs require developers to submit network configuration and control algorithms to third-party analyzers for analysis. This submission process raises concerns regarding the protection of intellectual property rights and compliance with global privacy regulations such as GDPR [9], DPA [10], CCPA [11], etc. There is a potential for violations when third parties analyze the source code or configuration files directly.

To address these concerns, we propose a novel static program analysis technology for detecting risky interactions in SAGINs without revealing sensitive source code. Our system utilizes the zero-knowledge (ZK) proof technique [12], which enables the prover to convince the verifier of the correctness of a statement without revealing any sensitive information. This allows developers to prove the analysis results to third-party analyzers while maintaining the confidentiality of their source code.

There are two major challenges in designing such a static program analysis scheme for SAGINs in ZK settings. The first challenge is the incompatibility between static program analysis algorithms and ZK systems, as most ZK systems operate on arithmetic circuits, whereas static program analysis for SAGINs typically involves more complex computations based on the RAM model. The second challenge is the lack of a specific abstract domain tailored for detecting risky interactions in SAGINs, which is crucial for the accuracy of static program analysis.

To overcome these challenges, we introduce an imperative programming language for SAGINs to facilitate the translation of network control algorithms into arithmetic circuits. Additionally, we develop a specialized abstract domain for interaction threats and design lattice operations and transfer functions based on the unique characteristics of interactions in SAGINs.

The main contributions are summarized as follows:

(1) We introduce the first static program analysis scheme in zero-knowledge (ZK) settings for SAGINs, enabling the detection of risky interactions without revealing source code.

(2) We present a novel imperative programming language for SAGINs and an abstract domain for interaction threats, which together improve the accuracy of static program analysis.

(3) Our evaluations of real-world SAGINs network scenarios demonstrate that our system can accurately detect risky interactions with an acceptable level of computational overhead.

In this paper, the remainder is organized as follows. Section 2 discusses the related works. We briefly introduce the preliminaries in Section 3. In Section 4 the system overview is presented. Section 5 provides the design details of the system. The experiments are given in Section 6. Finally, Section 7 is the conclusion.

2 Related works

Static program analysis techniques have been widely used for risky interactions analysis of IoT systems, Nguyen *et al.* [13] exploited static program analysis to convert the IoT program's source code as input for a model checker, which was used to expose interaction-level application flaws by detecting dangerous events that lead the system into an insecure state. It was also analyzed whether the detected violations were due to user misconfiguration or potentially malicious problems. However, this method only considers whether

individual applications violate security rules and does not consider the impact on the physical environment after the execution of device commands. In addition, Celik *et al.* [14] developed a static program analysis system Soteria for verifying that IoT programs and IoT environments follow the existing security, safety, and functional properties. Using the sensing-computing-executing program framework of existing IoT platforms to translate IoT programs' source code into intermediate representation code, Soteria analyzes the entire life cycle of a program, containing entry points, event handling methods, and method invocation graphs, so that Soteria can extract the state of the programs and it uses model checking methods to analyze whether these transitions follow existing secure operational procedures. Similarly, the system does not consider the interactions between multiple applications due to the shared physical environment. Ding *et al.* [15] extracted the physical interactions of IoT programs in a single platform through a static program analysis approach and designed an interaction risk calculation method. However, this work does not take into account the support of multiple IoT platforms and third-party platforms for IoT rules nowadays. IotCom [16] proposed a novel BRG abstraction technique that optimizes our analysis performance and significantly improves our performance relative to the state-of-the-art technology scalability. However, all previous static program analysis approaches in IoT systems require developers to submit IoT program source codes to third-party analyzers for analysis. Our work can static program analysis problems for detecting risky IoT interactions without revealing source code.

We present a novel static program analysis (SPA) technique that integrates zero-knowledge (ZK) proofs for IoT risk mitigation in Space-Air-Ground Integrated Networks (SAGINs). This approach is distinct from existing SPA techniques in several key ways, offering unique benefits while also presenting potential limitations.

2.1 Unique benefits of using ZK proofs in SPA

(1) Privacy Preservation: Traditional SPA techniques often require access to the source code for analysis, which can pose risks to intellectual property and privacy. By using ZK proofs, the proposed method allows developers to prove the absence of risky interactions without revealing the underlying source code. This zero-knowledge aspect is a significant advantage, ensuring that sensitive information remains confidential.
(2) Enhanced Security: ZK proofs enable a prover to demonstrate the truth of a statement without providing any additional information that could be exploited by malicious actors. This cryptographic method strengthens the security of the SPA process by ensuring that the analysis does not expose vulnerabilities that could be targeted.
(3) Compliance with Regulations: With data protection regulations such as GDPR and CCPA becoming increasingly stringent, the ability to conduct SPA without compromising source code is crucial. The ZK-based SPA scheme aligns with these regulations, allowing organizations to maintain compliance while still benefiting from the security assurances of SPA.
(4) Trust Establishment: The use of ZK proofs can establish trust between parties without the need for them to share sensitive information. This can be particularly beneficial in scenarios where third-party analysts or auditors are involved, as they can verify the code security without requiring access to it.

2.2 Potential limitations of using ZK proofs in SPA

(1) Complexity: The integration of ZK proofs into SPA introduces a level of complexity that may not be present with traditional SPA methods. Developers and analysts would need to understand the intricacies of both static program analysis and zero-knowledge proofs to effectively use the system.
(2) Performance Overhead: While the article mentions minimal computational overhead, the process of generating and verifying ZK proofs can be computationally intensive. This could potentially slow down the analysis process, especially in large-scale or resource-constrained environments.
(3) Scalability Concerns: The feasibility of applying ZK proofs at scale, particularly in networks with a vast number of IoT devices, maybe a concern. The scalability of the ZK-based SPA technique in such contexts need to be thoroughly tested and optimized.

In summary, the use of ZK proofs in SPA offers significant benefits in terms of privacy, security, regulatory compliance, and trust. However, these benefits come with potential limitations related to complexity, performance, and scalability.

3 Preliminaries

3.1 Abstract interpretation

Abstract interpretation [17] is one of the main formal methods of static program analysis. Abstract interpretation theory abstracts the program semantics, translates the concrete semantics of the program into the abstract domain, and analyzes the program properties under the abstract semantics in the abstract domain.

(1) Galois connection: Abstract interpretation theory is a mapping relation between concrete and abstract domains represented by a Galois connection. $\langle D, \sqsubseteq\rangle$ and $\langle D^{\#}, \sqsubseteq^{\#}\rangle$ are two given partial order sets, where $\langle D, \sqsubseteq\rangle$ is the set of partial order defined on the concrete domain, $\langle D^{\#}, \sqsubseteq^{\#}\rangle$ is the set of partial order defined on the abstract domain, the function α: $D \rightarrow D^{\#}$ and γ: $D^{\#} \rightarrow D$ are the pair of functions (α, γ), where the function α is the abstraction operator and call γ the concrete domain operator.

(2) Fixed point theory:$(D, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$, $(D^{\#}, \sqsubseteq^{\#}, \sqcup^{\#}, \sqcap^{\#}, \perp^{\#}, \top^{\#})$ be two complete lattices. The function pair (α, γ) is the Galois connection between them, and f and $f^{\#}$ are monotone functions on the two complete lattices, respectively. If $f^{\#}$ is a reliable abstraction of f, then $\alpha(\mathrm{LFP}(f)) \sqsubseteq^{\#} \mathrm{LFP}(f^{\#})$. Suppose there is no infinite increasing chain on the perfect lattice $(D^{\#}, \sqsubseteq^{\#}, \sqcup^{\#}, \sqcap^{\#}, \perp^{\#}, \top^{\#})$, and if $f^{\#}$ is a monotone function on $D^{\#}$ and $x_0^{\#}$ is a forward fixed point of $f^{\#}$, then the iterative process $x_{i+1}^{\#} = f^{\#}(x_i^{\#})$ will terminate at an infinite step. This iterative sequence is called the Kleene iterative sequence. Kleene iteration is a common method for solving the fixed point of a program in the abstract interpretation framework. The abstract interpretation framework ensures that the fixed point of the program under the abstract semantics is the upper approximation of the actual semantic fixed point, i.e., the reliability of the iterative process. Transform the static analysis into solving fixed points in the abstract domain.

3.2 Zero-knowledge proof

Zero-knowledge proofs were proposed by Goldwasser *et al.* [12], which is a two-party cryptographic protocol running between a prover and a verifier that can be used to perform knowledge proofs. Specifically, suppose that prover P has some secret information, and through a series of protocol processes, P can prove to prover V that he knows this information, and it will not disclose any relative information. Zero-knowledge proof has the following three properties:

(1) Completeness: Given a valid proof of a statement, the P can convince the V that the statement is true if both the P and the V run the protocol honestly.

(2) Soundness: V rejects with all if the result is not correctly computed.

(3) Zero-knowledge: P can prove the correctness of a statement to V without revealing any information other than the correctness.

The three properties of zero-knowledge proof enable it to have the capacities of trust establishment and privacy-preserving.

4 System overview

We present an overview of the system, a novel static program analysis technology for detecting IoT interaction threats without revealing source code in SAGINs. Our system consists of two types of users (prover, and verifier) and three major components.

4.1 System model

The system model, illustrated in Figure 1, adopts a three-layer architecture to facilitate seamless integration. The ground layer, represented by the green area, encompasses various smart vehicles, access points, and more. Positioned above it is the Air layer, depicted in blue, which consists of aeroplanes, drones, helicopters, and similar assets. Lastly, we have the space layer, depicted in grey, which comprises satellites. Each IoT device across these layers functions as a user within the SAGINs network. Figure 1 considers a practical scenario such as an emergency rescue operation. Specifically, the interaction between the car

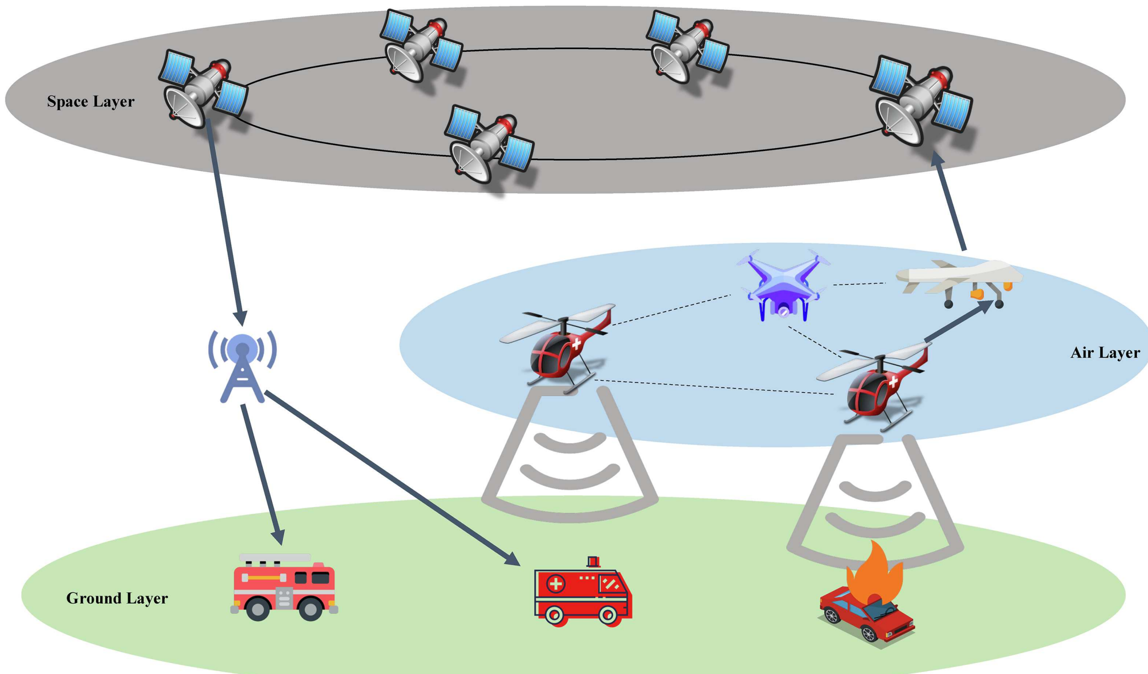

Figure 1: System model

on fire and the ambulance helicopter's IoT sensors may cause a fake alarm to fire fighting trucks by air layer and space layer. This risk is aggravated by the increased installed IoT programs and the unpredictability of physical spaces, thus it becomes more important to detect potentially risky interactions before installing programs. In this situation, the SAGIN plays a crucial role in the emergency response:

(1) Space layer: Satellites orbiting the Earth receive initial images from the Air Layer, providing the Ground Layer with real-time data on the extent of damage and potential hotspots that require immediate attention.

(2) Air layer: Helicopters and drones are deployed to the area where the early warning is required, relaying high-resolution images and live video feeds back to the Space Layer. Part of drones serve as communication relays, ensuring that the emergency response teams on the ground remain connected despite disrupted local communication infrastructure.

(3) Ground layer: Emergency response teams use the data received from the Space Layer to coordinate rescue efforts. Ground-based command centres analyze the information, prioritize rescue operations, and dispatch teams to the corresponding locations.

4.2 Security model

Here's a security model for the proposed system:

(1) The prover is semi-honest, meaning it will follow the protocol but may try to infer additional information from the verifier's responses.

(2) The verifier is honest, adhering strictly to the protocol and not attempting to gain unauthorized knowledge.

(3) The communication channel between the prover and verifier is secure and private, preventing eavesdropping by external adversaries.

(4) Adversaries may try to alter the proof or analysis results during transmission.

(5) External adversaries may attempt to intercept and analyze the communication between the prover and verifier.

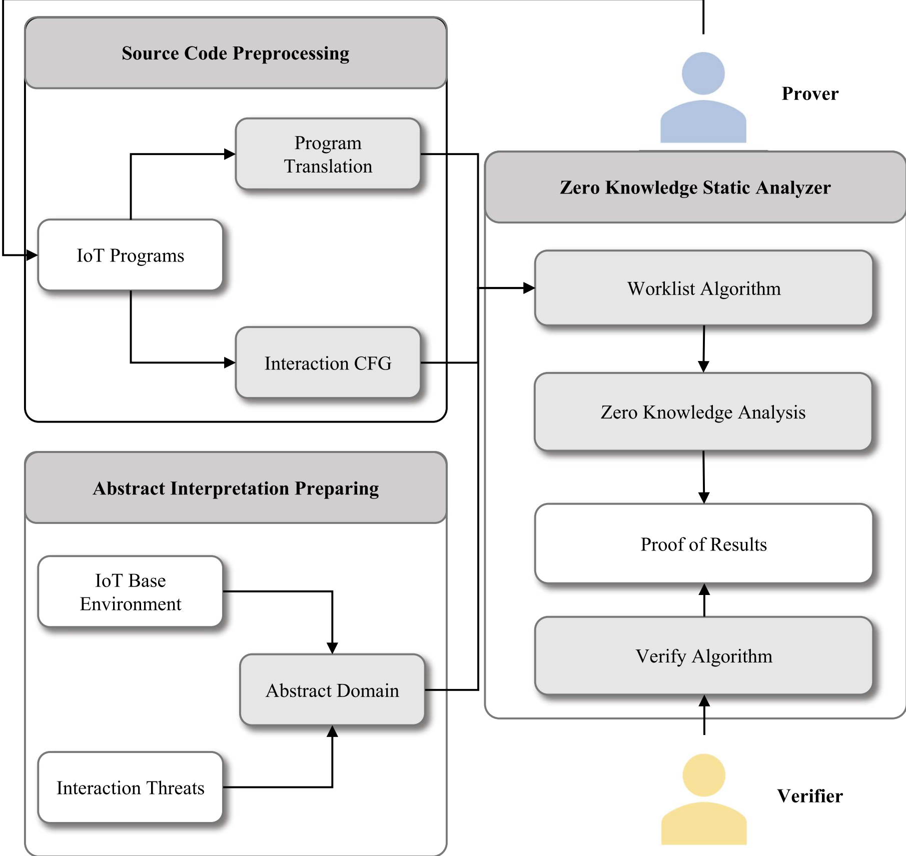

Figure 2: System workflow

4.3 System workflow

As shown in Figure 2, the system consists of source code preprocessing for identifying potential risky interactions, abstract interpretation preparation, and zero-knowledge static analysis components. Integrating these three components, we implemented a static program analysis approach to detect IoT interaction threats and ensure source code confidentiality.

Users can be categorized as provers and verifiers. The prover refers to the developer or owner of the programs, while the verifier can be an IoT platform or a hardware/software security company. The role of the prover is to construct proofs that demonstrate the correctness of analysis results without revealing the source code. The verifier's task is to validate these proofs and check the accuracy of the analysis results without accessing the source code. The workflow between the prover and verifier is as follows: Initially, the prover commits to the program and then proceeds with a zero-knowledge proof to demonstrate to the verifier the presence or absence of risky interactions in the IoT programs using abstract interpretation. It is assumed that the zero-knowledge static program analysis algorithms of the system are publicly available to users, and all users accept the validity of these algorithms. Using zero-knowledge abstract interpretation, the verifier can validate the correctness of the analysis results while maintaining the confidentiality of the prover's source code throughout the process.

(1) Source code preprocessing: We first convert IoT source codes into abstract commands tailored to avoid direct source code exposure. The purpose of this module is to prepare materials for static program analysis and includes two subcomponents: "Program Translation" and "Interaction Control Flow Graph (CFG)". Concretely, "Program Translation" refers to the process of converting different Internet of Things (IoT) programs into a unified programming language suitable for zero-knowledge proofs. This is because ZK proof relies on arithmetic circuits. Moreover, "Interaction CFG" excludes the control flow and data

of the general control flow graph, which have no impact on IoT interactions according to the properties of the IoT programs.

(2) Abstract interpretation preparing: We then translate the abstract model into arithmetic circuits for zero-knowledge proof. Specifically, we achieve this by compiling RAM programs into program-specific circuits. This module aims to propose an abstract domain for abstract interpretation based on the IoT-based environment and risky interactions in IoT systems since the abstract domain is problem-dependent. To the best of our knowledge, there isn't an abstract domain for IoT interaction threats that has been proposed yet.

(3) Zero knowledge static analyzer: We design a static program analysis scheme against interaction troubles in zero-knowledge settings. While most zero-knowledge systems operate on arithmetic circuits, static program analysis computations typically run on RAM model programs. Therefore, we need to overcome the technical obstacles to combining them. This module is designed to produce zero-knowledge proofs for static program analysis and includes three subcomponents, the worklist algorithm, zero knowledge analysis, and verify algorithm. The worklist algorithm takes the arithmetic circuits of programs, interaction CFG as input, and then operates arithmetic circuits over the abstract domain based on interaction CFG to generate the result. after that, Zero-knowledge analysis runs over the result to get analysis proof. Finally, The verifier obtains the result of the presence of IoT interaction threats by verifying proof.

5 Design details

In this section, we present the detailed design of our system. We first introduce our approaches for the source code preprocessing. Then, we discuss the design of abstract interpretation preparation. Finally, we describe our algorithm for zero-knowledge static analyzer.

5.1 Source code preprocessing

Since the arithmetic circuit and control flow graph of the program are needed in the zero-knowledge static analyzer, we need to pre-process the source code. The details are shown as follows.

5.1.1 Program translation

The computation of most zero-knowledge techniques over arithmetic or boolean circuits model, while abstract interpretation static program analysis schemes are in the RAM model. Therefore, We need to translate the RAM model program into the arithmetic model. However, IoT programs normally have loops, branches, and function call operations, which are not easy to translate to arithmetic circuits. Besides, programs on different IoT platforms are usually based on different programming languages (e.g., Groovy, Python, Javascript). Thus, we propose a new IoT imperative language based on Fang *et al.* [18] scheme that can be easily translated to arithmetic circuits as the bridge between traditional programming language with arithmetic circuits. The details are as follows:

(1) IoT imperative programming language. Most IoT programs follow the event-trigger paradigm and consists of three components: devices, events, and computation. A program is made up of multiple statements, each of which can be an assignment, a branch, or a loop. In our language, we use variables, constants, unary expressions, and binary expressions to represent expressions denoted by the symbol e. Variables can hold integer or boolean values, and we employ logical operators (and, or, not) as well as mathematical operators $(+, -, *, /)$ to express computations.

(2) Arithmetic converting. To convert the program into an arithmetic representation, we utilize table structures to represent the entire program. The following steps outline the conversion process: (1) Assign a unique "Stmt Code" to each type of statement. (2) The "Line No." field in the table stores the line number of the statement in the program. In the case of "if" and "while" statements, the line number refers to the line where the "if" or "while" condition is located. Additionally, we consider "else" and "end" as a single line, ensuring they also have a Line No. assigned. This approach helps in determining an upper bound for the number of outgoing edges during the construction of the control flow graph. (3) The variable "a" is only used in the assignment expression. (4) The "Variable ID" refers to the left part

of the assignment statements and the conditions in the "if" and "while" statements. By utilizing this table structure, we can effectively represent the program in an arithmetic form.

Similarly, for each expression a, we use "Expression Code" to identify the type of the expression. We store two possible "Variable ID", a "constant" and an "Op code". A "/" symbol means that the field is not applicable for this type of statement or expression and is left empty. Using these table structures, we can represent the whole program as a sequence of elements in the field F, and the prover can commit it using existing commitment schemes.

5.1.2 Interaction control flow graph

As abstract interpretation operates on flows, the detection of risky interactions in IoT programs requires a flow-sensitive approach. Hence, it becomes necessary to generate a control flow graph that represents the interactions. First, we generate a generic interaction control flow graph (CFG) based on existing work [19]. Second, We exclude flows that are not related to interaction on generic CFG, which reduces CFG size and reduces computational overhead. Finally, in order to associate with the arithmetic circuit. We revise the CFG following the rule: each node is a line of code labelled by its line number, and an edge denotes a low from node to node. Since we treat else and end as separate lines, in our simple programming language a node in this graph can have at most two outgoing edges in the case of loops and branches. Therefore, we represent the entire control low graph using a table of size. The row of the table stores the target line numbers of the two possible outgoing edges of the line.

5.2 Abstract interpretation preparing

After processing the source code, we get the input of the program p in the arithmetic circuit model and interaction CFG. Besides, the worklist algorithm also requires an abstract domain. The abstract domain is the core concept in abstract interpretation, static program analysis, and problem-dependent. The state set of a program is approximated by domain elements in the abstract domain, and the program's semantic actions (assignment, testing, control flow joining, looping, etc.) are modelled by domain operations in the abstract domain. Despite there being many different kinds of abstract domains, to the best of our knowledge, an abstract domain for analyzing the risky interaction in the IoT environment has not been proposed yet. A suitable abstract domain has a significant impact on the precision of the analysis problem. Therefore, we need to design the abstract domain of IoT interaction threats, including the octagon abstract domain, lattice operation, and transfer function.

5.2.1 The interaction octagon abstract domain

Based on the analysis of the IoT basic environment and interaction threats, we find that the commonly used symbol abstract domain and interval abstract domain are not suitable for analyzing IoT programs. This is due to the limited expressive capability of domain elements in these abstract domains, making it impossible to establish numerical relationships between multiple variables. Since IoT programs involve extensive variable relationships, and our analysis focuses on program interaction threats, a strong expressive capability is required. Therefore, we propose the interaction octagon abstract domain as a relational abstract domain.

Suppose that a finite set of program variables $V = \{v_1, v_2, \dots, v_n\}$, and the values of all variables belong to a numerical set $\mathcal{C}$, where $\mathcal{C}$ can be an integer set, a rational number set, or a real number set. A constraint shaped like $\{v_i - v_j \leqslant c(i \neq j), v_i + v_j \leqslant c(i \neq j), -v_i + v_j \leqslant c(i \neq j), -v_i - v_j \leqslant c(i \neq j), c \in \mathcal{C}\}$ is called an octagonal constraint.

5.2.2 Lattice operation

Lattice operations in the octagon abstract domain involve two key operations: meet (also known as the intersection) and join (also known as the union). These operations are used to combine and compare octagonal constraints within the abstract domain.

(1) Meet (Intersection $\cap$): The meet operation is the process of refining two abstract states into a more specific abstract state. In the octagon abstract domain, the meet operation is used to simulate the

branching points of program control flow, such as before the condition of an “if” statement. The goal of the meet operation is to find a more precise abstract state that represents the intersection of the two original states. For two octagonal constraint sets, the meet operation identifies all constraints that satisfy both states simultaneously. For example, if one state represents the constraint $x \geq 0$ and another state represents $x \leq 10$, the Meet operation may generate a new constraint indicating $0 \leq x \leq 10$.

(2) Join (Union $\cup$): The join operation is the process of merging two abstract states into a more general abstract state. In the octagon abstract domain, the Join operation is used to simulate the merging points of program control flow, such as after the two branches of an “if” statement or at the end of a loop. The goal of the Join operation is to find an octagonal constraint set that includes all possible values. These values should be consistent with at least one of the original two states. Specifically, for two octagonal constraint sets, the join operation considers all possible variable pairs and merges the constraints based on the relationships between these variables. For example, if one state represents the constraint $x \leq y+5$ and another state represents $y \leq x + 10$, the join operation may generate a new constraint indicating $x \leq y + 15$.

These lattice operations allow for the manipulation and refinement of octagonal constraints within the abstract domain, supporting the analysis and reasoning about variables and their relationships in the context of the interaction octagon abstract domain.

5.2.3 Transfer function

In static program analysis, a transfer function describes how an abstract state transitions from one program point to another after the execution of a single statement in the program. It maps the semantics of the program to elements in an abstract domain, enabling the analyzer to infer the potential states that the program may reach during its execution process. During the process of static analysis, the transfer function is used to update the abstract state to reflect all possible changes that may occur when the program reaches the next program point. This allows the analyzer to track the dynamic behaviour of the program during its execution and identify potential errors or risks.

Our approach involves two types of transfer functions.

(1) Assignment transfer function. When the program executes an assignment statement, the assignment transfer function is used to update the abstract state. It takes into account how the assignment operation changes the value of variables and updates the constraints in the abstract domain accordingly. For example, if the program executes the statement $x = y + 5$, the transfer function will update the constraints related to variable x to reflect its new value.

(2) Test transfer function. When the program executes a conditional statement (such as an if statement), the test transfer function is used to update the abstract state based on the result of the condition. It divides the abstract state into two parts: one corresponding to the state when the condition is true, and the other corresponding to the state when the condition is false. For example, if the program executes the statement $if(x > 0)$, the transfer function will generate two different abstract states for the true and false cases of $x > 0$ respectively.

5.3 Zero knowledge static analyzer

After the previous two modules, we are ready to start the zero-knowledge static program analysis, the details are as follows:

5.3.1 Worklist algorithm

We exploit the worklist algorithm from Fang *et al.* [18] as shown in Algorithm 1. The worklist algorithm takes as input a control flow graph of the program CFG and initializes a control flow queue W as $W = \{(l, l')|l' \in \text{CFG}(l)\}$. The abstract environment is initialized as $s_l(x) = \perp$ for all program variables x and for the bottom element of the abstract value lattice $\perp$. For a straight-line program, W has a simple definition mapping locations to their successor. For a program with a while loop, W will associate the line before the loop to both the first line of the loop and the first line after the loop, and associate the last line of the loop with the first line of the loop and the first line after the loop. For programs with

Algorithm 1 Worklist algorithm [18]

Input: A program p, interaction CFG, transfer function $T_{p,l}$, and lattice $L^{\#}$
Output: Abstract environment at each line $\{s_l\}_{l=1}^{n}$
1: Init $s_l(x) = \perp_{L^{\#}}$ for all l and x.
2: Init queue: $W = \{(l, l^{'}) | l^{'} \in CFG(l)\}$.
3: **while** $W \neq \varnothing$ **do**
4: $(l, l^{'}) = W.pop()$
5: **if** $T_{p,l}(s_l) \nsubseteq s_{l'}$ **then**
6: $s_{l'} = s_{l'} \sqcup T_{p,l}(s_l)$
7: **for all** $l^{''}$ follows $l^{'}$ **do**
8: $W.push(l^{'}, l^{''})$
9: **end for**
10: **end if**
11: **end while**
12: **return** $S = \{s_l\}_{l=1}^{n}$

functions, W encodes the control flow graph of the function call and return edges. The algorithm iterates over all control lows using a queue, updating line 6; this updates the analysis results at location l' to include new information, e.g., due to a new analysis result at location l. If the update leads to a new value for $s_{l'}$, then we must re-analyze all program points which are influenced by program point l'', so we add flows (l, l') to the worklist for lines 5, 7 and 8. When instantiated to the octagon abstract domain with infinite ascending chains for interaction analysis to join operation $\sqcup$ in line 6.

We utilize the worklist algorithm described by Fang *et al.* in their work [18], as shown in Algorithm 1. The worklist algorithm takes a control flow graph of the program CFG as input and initializes a control flow queue W with elements in the form (l, l') where l' belongs to the set CFG(l).

The abstract environment is initialized as $s_l(x) = \perp$ for all program variables x and for the bottom element $\perp$ of the abstract value lattice. In the case of a straight-line program, W is defined simply by mapping locations to their successors. For programs containing while loops, W associates the line before the loop with both the first line of the loop and the first line after the loop, and associates the last line of the loop with the first line of the loop and the first line after the loop. For programs with functions, W encodes the control flow graph of function calls and return edges.

The algorithm iterates over all control flows using a queue, updating line 6. This update incorporates new information, such as a new analysis result at location l, into the analysis results at location l'. If this update results in a new value for $s_{l'}$, it necessitates re-analyzing all program points influenced by program point l''. Therefore, flows (l, l') are added to the worklist for lines 5, 7, and 8.

When applied to the octagon abstract domain with infinite ascending chains for interaction analysis, the join operation $\sqcup$ is used in line 6.

5.3.2 Analysis & verify

(1) Zero knowledge analysis: After executing Algorithm 1, we obtain the set $S = \{s_l\}_{l=1}^{n}$. Subsequently, we apply the zero-knowledge proof scheme (e.g., SNARK [20]) to process S and obtain the analysis result. This process also generates the corresponding proof π.
(2) Verify algorithm: The verifier examines the program using the analysis parameters $(\mathrm{CFG}, T_{p,l}, \mathrm{Alg}, L^{\#}, S, \pi)$. Here, CFG represents the control flow graph, $T_{p,l}$ denotes the transfer function, Alg corresponds to the worklist algorithm, $L^{\#}$ represents the lattice, S represents the program in the abstract domain, and π is the zero-knowledge proof. The verification process yields a result $r = \mathrm{verify}(\mathrm{CFG}, T_{p,l}, \mathrm{Alg}, L^{\#}, S, \pi)$, where r can take values from the set 0 or 1. A value of "1" indicates the presence of risky interactions, while a value of "0" indicates the absence of risky interactions.

5.4 Security analysis

(1) Zero-Knowledge Proofs: This article introduces the use of Zero-Knowledge Proofs (ZKP) [12] where developers can prove to third-party verifiers that their program does not contain risky interactions. The

zero-knowledge property of ZKP ensures that the proof process does not leak the source code. This proof mechanism guarantees the confidentiality of the source code as verifiers can only confirm the program's safety without gaining knowledge of its implementation.

(2) Abstract Interpretation: This article adopts a novel imperative programming language and specialized abstract domain [17] to convert IoT control algorithms into arithmetic circuits suitable for ZKP analysis. This process involves abstracting the source code, further ensuring the confidentiality of the source code during the analysis process.

(3) Soundness: In particular, Soundness guarantees that if the prover claims that a program is secure, and the verifier accepts the proof, then the program is in fact secure. This means that even if there is a malicious prover trying to fool the system by claiming that an insecure program is secure, the Soundness attribute prevents such fraud from succeeding. In the paper, by introducing zero-knowledge proofs, developers can prove to third-party verifiers that their programs do not contain risky interactions without disclosing sensitive source code. The security of this procedure is based on the Soundness attribute of ZKP, which ensures that the verifier can only confirm the security of the program and cannot obtain any details about the implementation of the program. This not only protects intellectual property but also complies with privacy regulations such as GDPR and CCPA.

6 Evaluation

In this section, we perform an extensive evaluation of the proposed system.

6.1 Experiment Setup

(1) Dataset. Our experiments were conducted on a multiplatform dataset consisting of 12 IoT devices sourced from two different platforms. The first source is the SmartThings apps, where we collected 243 apps from the SmartThings public repository [21]. These apps are written in Groovy and utilize the SmartThings Classic API platform. The second source is the IFTTT applets [22], for which we utilized the dataset provided by Bastys *et al.* This dataset is in JSON format, with each object defining an IFTTT applet.
(2) Library. The system is implemented in C++ using the open-source compiler of libsnark to generate Rank-1 Constrains System (R1CS). Our system efficiently compiles static program analysis on a program into an R1CS instance. Besides, we have the flexibility to utilize any generic zero-knowledge proof scheme on R1CS. In this case, we have chosen the pairing-based SNARK [20]. This scheme offers a constant size proof and fast verifier time.
(3) Hardware. We performed the experiments on a MacBook Pro with a 2.3 GHz Quad-Core Intel Core i5 processor and 16GB RAM.

6.2 Experiment results and analysis

6.2.1 Accuracy

We select 12 groups of IoT programs from IoTMAL [14] to evaluate the effectiveness and accuracy of the system. Specifically, there are 7 groups of IoT programs that are individual programs, and 5 groups that are bundled programs. For bundled programs, each group contains 2 to 4 IoT programs that have a connection. There are the IFTTT individual program and BundleSmartLight bundled program without the interaction threat. Table 1 summarizes the results of our experiments for evaluating the accuracy of our system in detecting IoT interaction threats. Our system succeeds in detecting 10 group programs that have interaction threats, which denote ✓ in True Positive (TP) columns. Besides, we also found the BundleSmartLight bundled program without the interaction threat, which denotes ✓ in True Negative (TN) columns. However, our system has false positives for the IFTTT program, which denote ✓ in False Positive (FP) columns. By calculation, our system precision rate is 90.9%, the false rate is 8.3%, and the accuracy rate is 91.7%, which proves the effectiveness of our system, and the accuracy is enough for practical use.

Table 1. Interaction threats detection result

Test Programs	TP	TN	FP
Individual Programs			
1.BrightenMyPath	✓		
2.TurnItOnOffandOnEvery30Secs	✓		
3.Smartweather-station-controlle	✓		
4.Switch-activates-home-phrase-or-mode	✓		
5.Smartblock-chat-sender	✓		
6.Close-the-valve	✓		
7.IFTTT			✓
Bundled programs			
8.BundleSleepMode	✓		
9.BundleSmartWindow	✓		
10.BundleHomemode	✓		
11.BundleSmartLight		✓	
12.BundleFireAlarm	✓		
Precision			90.9%
False			8.3%
Accuracy			91.7%

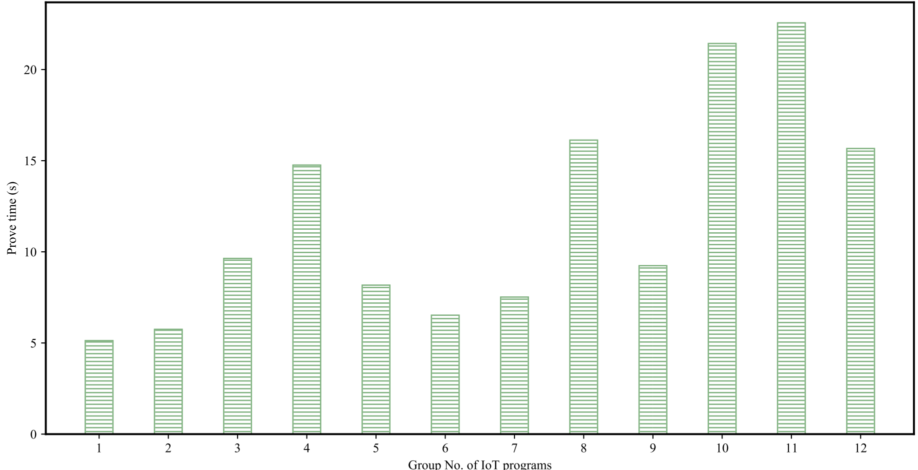

Figure 3. Prove times

6.2.2 Performance

In this part, we present the time we analyze 12 groups of programs, which were mentioned before. As shown in Figure 3. The maximum time overhead for generating proofs is 22.56 s, and the minimum time overhead is 5.13 s. Moreover, since we choose zksnark [20] as the backend, the size of the generated proof is constant at 128bytes, and the verification time is constant at about 20 ms. In Figure 4, The maximum number of lines of code for IoT programs is 215, and the minimum number of lines of code is 40. We found that the trend shown in Figure 4 is almost the same as that in Figure 3. Obviously, the number of lines of code and time overhead are positively correlated. It should be mentioned, that IoT programs usually have about 100 lines of code, so the performance of our system is practical.

7 Conclusion

This paper has addressed the critical challenges of coordination, synchronization, and security in SAGINs. We have proposed a novel static program analysis approach that leverages zero-knowledge proofs to detect risky interactions without revealing sensitive source code. This method not only protects intellectual property but also adheres to stringent privacy regulations. Our system, which includes a tailored imperative

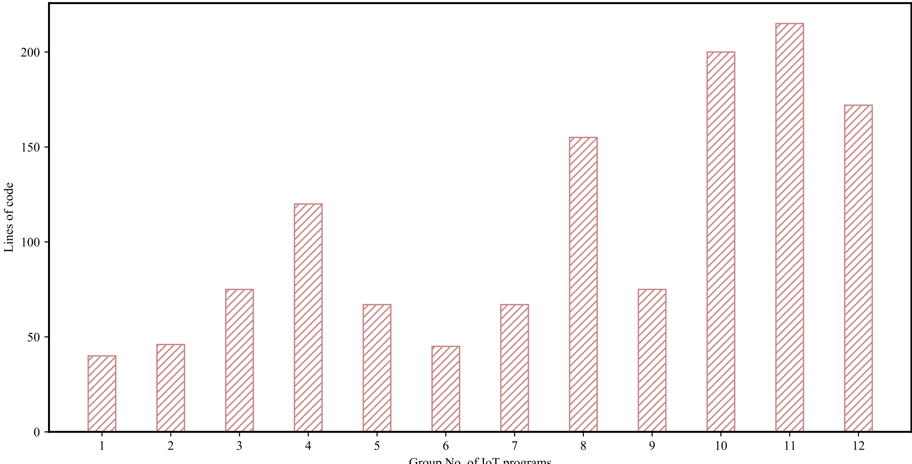

Figure 4. Lines of program

programming language and a specialized abstract domain, has been validated through extensive evaluations. It demonstrates high accuracy in identifying risky interactions with minimal overhead, thereby enhancing the reliability and security of SAGINs. Looking forward, the integration of advanced analytics and adaptive learning could further refine our approach, ensuring the robustness of SAGINs in an increasingly interconnected world.

Conflict of interest

The authors declare no conflict of interest.

Data Availability

No data are associated with this article.

Authors' Contributions

Haotian Deng identified the problem of IoT risk in Space-Air-Ground Integrated Networks research on the current status. Weijie Wang and Xiaochen Ma compiled the latest research status on Space-Air-Ground Integrated Networks and proposed research directions. Haotian Deng, Chuan Zhang, and Liehuang Zhu jointly designed this scheme, Liu Tao conducted the security analysis, and Huishu Wu performed the performance analysis.

Acknowledgements

We thank all anonymous reviewers for their helpful comments and suggestions.

Funding

This work is supported by the National Natural Science Foundation of China (Grant Nos. 62232002, 62202051), the National Key R&D Program of China (Grant Nos. 2021YFB2700500 and 2021YFB2700503), the China Postdoctoral Science Foundation (Grant Nos. 2021M700435, 2021TQ0042), the Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies (Grant No. 2022B1212010005), and the Key-Area Research and Development Program of Guangdong Province (Grant No. 2021B0101400003), the Open Project Funding of Key Laboratory of Mobile Application Innovation and Governance Technology, Ministry of Industry and Information Technology (Grant No. 2023IFS080601-K), the Yunnan Provincial Major Science and Technology Special Plan Projects (Grant No. 202302AD080003), the Beijing Institute of Technology Research Fund Program for Young Scholars, and the Young Elite Scientists Sponsorship Program by CAST (Grant No. 2023QNRC001).

References

[1] Wei L, Shuai J, Liu Yu, et al. Service customized space-air-ground integrated network for immersive media: Architecture, key technologies, and prospects. Chin Commun 2022; **19**: 1–13
[2] Wang P, Zhang J, Zhang X, et al. Convergence of satellite and terrestrial networks: A comprehensive survey. IEEE Access 2019; **8**: 5550–5588
[3] Ye J, Dang S, Shihada B, et al. Space-air-ground integrated networks: Outage performance analysis. IEEE Trans Wireless Commun 2020; **19**: 7897–7912

[4] Tang Y, Qian F, Gao H, et al. Synchronization in complex networks and its application–a survey of recent advances and challenges. Ann Rev Control 2014; **38**: 184–198

[5] Pongsakornsathien N, Bijjahalli S, Gardi A, et al. A performance-based airspace model for unmanned aircraft systems traffic management. Aerospace 2020; **7**: 154

[6] Weimann G. Cyberterrorism: The sum of all fears? Stud Confl Terr 2005; **28**: 129–149

[7] Sharif S, Zeadally S and Ejaz W. Space-aerial-ground-sea integrated networks: Resource optimization and challenges in 6g. J Network Comput Appl 2023; 103647

[8] Caamaño-Martín E, Laukamp H, Jantsch M, et al. Interaction between photovoltaic distributed generation and electricity networks. Prog Photovoltaics Res Appl 2008; **16**: 629–643

[9] European Union. General Data Protection Regulation (gdpr). https://gdpr-info.eu/, 2016

[10] The United Kingdom. Data protection act. https://www.gov.uk/data-protection, 2018

[11] State of California Department of Justice. California consumer privacy act. https://oag.ca.gov/privacy/ccpa, 2018

[12] Goldwasser S, Micali S and Rackoff C. The knowledge complexity of interactive proof-systems (extended abstract). In: 17th Annual ACM Symposium on Theory of Computing, Rhode Island, USA, ACM, 1985, 291–304

[13] Nguyen DT, Song C, Qian Z, et al. Iotsan: fortifying the safety of iot systems. In: CoNEXT 2018, Heraklion, Greece, ACM, 2018, 191–203

[14] Celik ZB, McDaniel P and Tan G. Soteria: Automated iot safety and security analysis. In: USENIX ATC 2018, Boston, MA, USA, USENIX Association, 2018, 147–158

[15] Ding WB and Hu HX. On the safety of iot device physical interaction control. In CCS 2018, Toronto, ON, Canada, ACM, 2018, 832–846

[16] Alhanahnah M, Stevens C and Bagheri H. Scalable analysis of interaction threats in iot systems. In: ISSTA 2020, Virtual Event, USA, ACM, 2020, 272–285

[17] Cousot P and Cousot R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, ACM, 1977, 238–252

[18] Fang ZY, Darais D, Near JP and Zhang YP. Zero knowledge static program analysis. In: CCS 2021, Virtual Event, Republic of Korea, ACM, 2021, 2951–2967

[19] Hsu KH, Chiang YH and Hsiao HC. Safechain: Securing trigger-action programming from attack chains. IEEE Trans Inf Forensics Secur 2019; **14**: 2607–2622

[20] Groth J. On the size of pairing-based non-interactive arguments. In: Marc Fischlin M and Coron J-S (eds.). Advances in Cryptology – EUROCRYPT 2016, Vienna, Austria. Lecture Notes in Computer Science, Springer, 2016, 9666, 305–326

[21] Statista. Number of internet of things (iot) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/, 2022.

[22] IFTTT. If this then that. https://ifttt.com/, 2024

Haotian Deng received his M.S. degree from Clemson University, United States, in 2021. He is currently working towards a Ph.D. degree in the School of Cyberspace Science and Technology, Beijing Institute of Technology, China. His research interests include blockchain, IoT security, and applied cryptography.

Liu Tao received her Ph.D. degree in Computer Software and Theory from Wuhan University, China, in 2010. She is currently a researcher working at the China Academy of Information and Communications Technology and the Key Laboratory of Mobile Application Innovation and Governance Technology, MIIT. Her research focuses on mobile intelligent terminal security, terminal security detection techniques, and personal information protection.

Xiaochen Ma received his B.S. degree from Beijing Institute of Technology, China, in 2020. He is currently a master student in the School of Computer Science and Technology, Beijing Institute of Technology. His research interests include cloud security and blockchain security.

Weijie Wang received his B.S. degree from Xidian University, China, in 2020. He is currently a master student in the School of Computer Science at the Beijing Institute of Technology. His research interests include federal learning, security, and privacy in blockchain.

Chuan Zhang received his Ph.D. degree in computer science from Beijing Institute of Technology, China, in 2021. He is currently an assistant professor at the School of Cyberspace Science and Technology, Beijing Institute of Technology. His research interests include secure data services in cloud computing, applied cryptography, machine learning, and blockchain.

Huishu Wu received his B.E. degree in information management from Hebei Normal University of Science Technology, China, in 2014. He received his master degree in management in China University of Political Science and Law, China, and master of law from University of Montreal, Canada, in 2017. He received his Ph.D. degree in law from University of Montreal in 2024. His research interests include data protection, audit system design, data science and algorithm optimization.

Liehuang Zhu received his Ph.D. degree in computer science from Beijing Institute of Technology, China, in 2004. He is currently a professor at the School of Cyberspace Science and Technology, Beijing Institute of Technology. His research interests include security protocol analysis and design, group key exchange protocols, wireless sensor networks, and cloud computing.