




Article

Distributed Ledger-Based Authentication and Authorization of IoT Devices in Federated Environments

Michał Jarosz ¹, Konrad Wrona ^{1,2} and Zbigniew Zieliński ^{1,*}

¹ Faculty of Cybernetics, Military University of Technology, 00-908 Warsaw, Poland; michal.jarosz@wat.edu.pl (M.J.)

² NATO Cyber Security Centre, 2597AK The Hague, The Netherlands

* Correspondence: zbigniew.zielinski@wat.edu.pl

Abstract: One of the main security challenges when federating separate Internet of Things (IoT) administrative domains is effective Identity and Access Management, which is required to establish trust and secure communication between federated IoT devices. The primary goal of the work is to develop a “lightweight” protocol to enable authentication and authorization of IoT devices in federated environments and ensure the secure communication of IoT devices. We propose a novel Lightweight Authentication and Authorization Framework for Federated IoT (LAAFFI) which takes advantage of the unique fingerprint of IoT devices based on their configuration and additional hardware modules, such as Physical Unclonable Function, to provide flexible authentication and authorization based on Distributed Ledger technology. Moreover, LAAFFI supports IoT devices with limited computing resources and devices not equipped with secure storage space. We implemented a prototype of LAAFFI and evaluated its performance in the Hyperledger Fabric-based IoT framework. Three main metrics were evaluated: latency, throughput (number of operations or transactions per second), and network resource utilization rate (transmission overhead introduced by the LAAFFI protocol). The performance tests conducted confirmed the high efficiency and suitability of the protocol for federated IoT environments. Also, all LAAFFI components are scalable as confirmed by tests. We formally evaluated LAAFFI security using Verifpal as a formal verification tool. Based on the models developed for Verifpal, we validated their security properties, such as message secrecy, authenticity, and freshness. Our results show that the proposed solution can improve the security of federated IoT environments while providing zero-day interoperability and high scalability. Compared to existing solutions, LAAFFI is more efficient due to the use of symmetric cryptography and algorithms adapted for operations involving IoT devices. LAAFFI supports multiple authorization mechanisms, and since it also offers authentication and accountability, it meets the requirements of Authentication, Authorization and Accounting (AAA). It uses Distributed Ledger (DL) and smart contracts to ensure that the request complies with the policies agreed between the organizations. LAAFFI offers authentication of devices belonging to a single organization and different organizations, with the assurance that the encryption key will be shared with another device only if the appropriate security policy is met. The proposed protocol is particularly useful for ensuring the security of federated IoT environments created ad hoc for special missions, e.g., operations conducted by NATO countries and disaster relief operations Humanitarian Assistance and Disaster Relief (HADR) involving military forces and civilian services, where immediate interoperability is required.

Keywords: internet of things; blockchains; authentication; distributed systems



Citation: Jarosz, M.; Wrona, K.; Zieliński, Z. Distributed Ledger-Based Authentication and Authorization of IoT Devices in Federated Environments. *Electronics* **2024**, *13*, 3932. <https://doi.org/10.3390/electronics13193932>

Academic Editor: Domenico Rosaci

Received: 8 September 2024

Revised: 29 September 2024

Accepted: 30 September 2024

Published: 4 October 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Many important applications of the Internet of Things (IoT), such as smart cities, smart health care or hybrid military operations, increasingly rely on the concept of federation. The purpose of forming a federation is to allow the various parties to cooperate and exchange information, such as the location of each other and the detected threats. An example

of a federation is the Federated Mission Networking (FMN) environment [1], which can support the Civil–Military Cooperation (CIMIC) of civilian emergency response services and military forces when providing HADR, e.g., after natural disasters. In FMN, each participant retains control over their capabilities and operations while accepting and meeting the requirements outlined in earlier agreed-upon arrangements, particularly in the joint security policy. The main idea behind FMN is to unite forces in a federated mission environment to improve information sharing and ensure effective command and decision making for complete control of operations. Establishing an effective Identity and Access Management (IAM) framework across the federation of separate administrative domains is necessary to ensure trust and secure communication between federated partners. However, it introduces several challenges from a security perspective.

Our study focuses mainly on critical IAM issues related to the authentication and authorization of federated IoT devices during HADR operations. To meet the requirements of such operations, we propose a novel Lightweight Authentication and Authorization Framework for Federated IoT (LAAFFI) that provides a flexible authentication protocol based on DL, leverages the unique configuration fingerprint of an IoT device, and can also accommodate IoT devices not equipped with secure storage space. Compared to existing solutions, the LAAFFI enables the authorization and establishment of a secure connection between any organization’s IoT device and a DL and between two federated IoT devices with reduced data overhead and number of messages exchanged. We see our contribution in this work as follows:

- Firstly, we have developed a novel effective authentication and authorization protocol of IoT devices in a federated environment using the IoT devices fingerprint and the Hyperledger Fabric as DL.
- Secondly, we formally evaluated the security of LAAFFI using Verifpal as formal verification tool. Based on the models developed for the Verifpal, we validated message secrecy, authenticity, and freshness of LAAFFI protocol.
- Thirdly, we implemented a prototype of LAAFFI and evaluated its performance metrics such as latency, throughput (number of operations or transactions per second), and transmission overhead introduced by the LAAFFI protocol.

The remainder of the paper is organized as follows. Section 2 describes an operational scenario and identifies the design requirements for a federated civil-military IoT environment. Section 3 introduces the LAAFFI protocol for the authentication and authorization of IoT devices based on DL and demonstrates how the framework meets the requirements introduced by federated IoT scenarios. Section 4 analyzes the entropy and security of three different sources of cryptographic material proposed for use in LAAFFI. Section 5 discusses the results of a formal security validation of the proposed protocol. Section 6 analyzes the resilience of LAAFFI to various security attacks. Section 7 introduces the implementation of the proof of concept of our framework, while Section 8 presents the results of the performance evaluation, and Section 9 discusses related works. Section 10 summarizes our main results and identifies possible directions for future work.

2. Operational Scenario and Requirements

A specific situation where an organization needs to use data from IoT devices belonging to other organizations is an exchange of information between organizations, such as military forces, municipal services, volunteer firefighters brigades, and the International Committee of the Red Cross (ICRC) providing humanitarian assistance in response to a natural disaster or an armed conflict. As the experience of the war in Ukraine shows, hostilities are often accompanied by attacks on civilian infrastructure and facilities far from the front line, usually located in highly urbanized areas. These attacks can be carried out, for example, by drones, missiles, or other means of aerial assault, and result in the destruction of residential buildings and civilian institutions, involving numerous human casualties. Preserving the lives of the victims can depend on their rapid localization and the provision of first aid. Furthermore, the war in Ukraine shows that the use of the civilian communications

infrastructure makes sense in such warfare scenarios since even in the case of the large-scale destruction of infrastructure, a wireless network is often still partially functional or can be restored rapidly. Therefore, at least parts of the Smart City infrastructure, particularly those that allow wireless communications, can be kept operational, providing access to various civilian IoT devices, such as traffic cameras and air pollution sensors. Personal mobile devices, such as smartphones or smartwatches, equipped with a special application that can be launched in an emergency, can also use the surviving infrastructure to raise the alarm when a life-threatening condition or injury is detected. To ensure the reliability of these information sources and the integrity of information transmission, devices operated by private users and civilian organizations should be registered and subjected to a domain authentication process by a trusted party, such as a mobile operator. To further increase the ability of the federated partners to obtain an accurate near real-time picture of the situation during a HADR operation, specialized surveillance assets, such as Unmanned Aerial Vehicles (UAVs), can be deployed by individual partners.

To obtain precise and timely situational awareness and the so-called common operational picture, there is often a need to transmit data obtained by sensors belonging to individual federation partners to all partners forming a federation. In some cases, to preserve connectivity, IoT devices, such as city surveillance cameras (CCTV), may need to connect to the network provided by another federation partner. With a federation formed, civilian and military participants can reliably exchange information about various threats or incidents that require emergency response. For example, specialized sound sensors can be attached to civilian infrastructure and installed at many points in the city area. They can transmit reconnaissance information on the type, location of detection, distance, and direction of an incoming object, such as a missile. This information should be immediately transmitted from a device attached to the civilian infrastructure to the military network, e.g., to assist in air defense and civilian protection.

If a residential building is damaged, the relevant services should immediately be notified of the extent of the damage and those injured. The detection of a series of explosions can cause military drones to be dispatched to the area and obtain a situational picture. Drones can obtain camera images and sensor data, for example, by probing the atmosphere for the presence of chemical agents.

The federated situational awareness system should ensure that the input data are trustworthy, i.e., they were obtained from authorized IoT devices, and they were not subjected to unauthorized modification. These operational requirements require the effective authentication and authorization of the IoT devices and the ability to establish a secure communication channel with the situational awareness system.

Several design requirements must be considered when developing a security framework for federated IoT environments [2,3]:

1. *Zero-day interoperability* means that the framework should ensure immediate interoperability due to the frequent dynamic formation of federations to carry out missions such as HADR. The limited computing resources of IoT devices require *lightweight* mechanisms, balancing performance and security. The communication overhead of the authentication protocols should be minimized in terms of the number of messages exchanged between the authentication parties and the size of the messages sent.
2. *Key management* is a complex process that must take into account the IoT limitations. This process applies to all cryptographic key handling operations, i.e., key generation, key exchange, key storage, revocation, and key usage.
3. *Decentralization* implies that no organization controls the entire system. The device or user belonging to one federated organization shall be able to authenticate to a server belonging to another federated organization and use its services without creating an identity or registering credentials with that organization. Decentralization is also a way to achieve high reliability.
4. *Separation* mandates that users and devices access resources only by a federation security policy, e.g., device authentication data must be protected and stored securely.

5. The authentication and authorization system should be resilient to failures and harmful activities to ensure the *high availability* of IoT information. This property can be realized by increasing service instances to handle more requests or maintaining business continuity when some instances are unreachable.
6. *Lightweight* protocols and technologies are required due to the limited performance of IoT devices.
7. *Scalability* requires the IoT authentication scheme to scale well even for a very large IoT network.
8. The framework must provide *accountability* for the authorization of communication flows and data delivery by individual IoT devices, e.g., the fact of an organization obtaining a key to communicate with devices from another organization should be recorded.

We propose a Distributed Ledger (DL)-based authentication and authorization framework for federated IoT environments to meet the requirements identified above. The choice of DL is motivated by several desirable features of DL that allow us to meet the identified requirements. First, some DLs, such as Hyperledger Fabric (HLF), can provide the *ability to compartmentalize stored data* for different purposes and organizations within the federation, including support for private data storage for specific organizations. Data can be compartmentalized by implementing *channels*, i.e., *subchains* [4], that create separate ledgers and isolate data destined for different members of the federation or different purposes, e.g., separating authentication and authorization information. Second, the *ability to store data outside the ledger* with access to data limited to specific federation participants supports the implementation of flexible security policies. Furthermore, it supports *smart contracts*, i.e., computer programs stored and executed by a DL, the outcomes of which are recorded in the DL [5]. In addition, the *modularity* of the main functional components of a DL, such as the database and the consensus mechanism, allows flexibility of the implementation choices required in a federated environment. Finally, *scalability and performance* must be suitable for a large-scale IoT federation. For federation participants to securely exchange information, e.g., about the various types of threats or incidents identified in Section 2, it is necessary to provide three types of operations involving DL:

1. Writing data to the DL, used to transmit data to other participants in the mission.
2. Reading data from DL, usually performed periodically to retrieve data transmitted by other participants.
3. Transferring data between IoT devices of different organizations.

In the cases of (1) and (2), DL mediates the exchange of data, and secure communication between the IoT device and the DL is needed; in the case of (3), direct communication between different devices is required, which occurs, for example, when it is necessary to transmit a stream of near real-time video data from the camera of a drone unit to the closest, in terms of coverage, IoT gateway of another organization providing access to these data. Another example in which direct communication is necessary between different IoT devices is the transfer of data from sensors deployed by one organization to the drones of another organization that perform operational reconnaissance.

3. Authentication and Authorization

Our Lightweight Authentication and Authorization Framework for Federated IoT (LAAFFI) leverages the unique fingerprint of the IoT device for its authentication and authorization and adheres to the principles defined in [6]. The protocol defines the registration procedure for the IoT device to the DL and communication procedures that allow the IoT device to securely communicate with the DL or with other IoT devices, including those belonging to different organizations within the federation. We assume that all federated partners participate in a DL containing the identities of the IoT devices and data required to authenticate and authorize them. Federated partners must agree on the rules for registering devices, managing permissions, and accessing data, formulated as smart

contracts, executable by all nodes of the DL. Each partner should operate multiple DL nodes to ensure sufficient reliability.

To protect against insider attacks, we assume the existence of a Security Information and Event Management (SIEM) that is responsible for collecting events from system components, such as DL nodes and gateways, and correlating them so that they can be used to detect insider threats. In particular, an insider with administrator privileges can gain access to the device authentication data and can impersonate an IoT device.

Similarly, suppose that an external attacker gains access to a device, e.g., by capturing an IoT device deployed in a contested environment. In that case, the device can be considered compromised and to be acting as an insider attacker with limited privileges. This device should be isolated from the system; for example, it should not be allowed to connect to DL. A SIEM can help detect compromised devices and rogue administrators by analyzing the actions taken by the device and identifying deviations from the usual behavior pattern.

3.1. Registration Phase

Device registration is a critical process for ensuring the system's security and, thus, should be performed only by authorized entities, preferably in advance of the federated operation. In the case of military devices, military IT personnel could be responsible for the registration: in the case of city cameras, municipal employees, and in the case of privately owned phones and smartwatches, the mobile operators. A DL provides a secure and reliable data store for device information. In the registration process, the site administrator adds to DL the relevant data about the device, its owner, and permissions. Data are added to DL by executing smart contracts that validate the added data.

During registration, the device sends a *parameter array* to the DL, which forms a *fingerprint* of the device. The chosen configuration data must consist of parameters with the appropriate entropy to form a *fingerprint* of the IoT device. Depending on the capabilities of the IoT device, the parameters recorded in the array may vary. We distinguish three cases. The first one is for low-end devices equipped with the operating system, *unique configuration data* such as hardware parameters, e.g., device serial number and memory card serial number, and software parameters, e.g., partition IDs, file system IDs, and keys stored on the device, can be used as a fingerprint. The device must store the commands required to obtain these configuration data when needed. Examples of commands to obtain such parameters on the RPI platform with the Raspbian operating system are presented in Table 1.

Table 1. Sample commands and maximum theoretical entropy of their results.

No	Command	Entropy in Bits
1	/opt/vc/bin/vcgencmd otp_dump grep "^ 29" cut -d : -f 2	32
2	cat /proc/device-tree/serial-number	32
3	udevadm info -a -n /dev/mmcblk0 grep serial cut -d = -f 3	32
4	udevadm info -a -n /dev/disk/by-label/boot grep ATTRS{cid} cut -d = -f 3	32
5	sudo blkid grep PTUUID awk '{print \$2}' awk -F "" '{print \$2}'	32
6	sudo blkid grep RECOVERY awk '{print \$4}' awk -F "" '{print \$2}'	128
7	sudo blkid grep SETTINGS awk '{print \$3}' awk -F "" '{print \$2}'	32
8	sudo blkid grep boot awk '{print \$4}' awk -F "" '{print \$2}'	32
9	sudo blkid grep root awk '{print \$3}' awk -F "" '{print \$2}'	128
10	sudo cat /etc/shadow grep pi awk -F '\$' '{print \$3}'	96
11	sudo dumpe2fs /dev/mmcblk0p5 grep Hash awk '{print \$4}'	128
12	sudo dumpe2fs /dev/mmcblk0p7 grep Hash awk '{print \$4}'	128
13	sudo dumpe2fs /dev/mmcblk0p7 grep "Filesystem created"	>23
14	cat /etc/ssh/host_dsa_key	>512
15	sudo cat /etc/dhcpd.secret	128

The second case refers to the situation when the IoT device is equipped with a *Physical Unclonable Function (PUF)*. Instead of storing commands to obtain unique configuration data, the device stores the signal input (challenges) for PUF. The responses to these challenges are preserved in the DL.

The third case is when the device can securely generate and store confidential data for longer periods (e.g., cryptographic keys); the device can generate random strings with the required entropy and store them in the DL. Each string is considered a separate parameter.

In the first and second scenarios above, the device does not store the persistent private authentication data in its storage—instead, it stores a program that will enable obtaining specific parameters, e.g., a system command to obtain a particular parameter or a challenge to PUF.

When using random data, these data must be stored as parameters securely on the device and in the DL authentication channel.

For the PUF challenge–response and unique configuration data, the hash of each parameter must be saved in DL as

$$P = \{H(p_1), H(p_2), \dots, H(p_n)\} \quad (1)$$

where $H(p_i) = \text{HMAC}(K, p_i)$ and K is one of the program parameters shown in Table 1. If the parameter array is disclosed, the device must be registered again. Using Hash-based Message Authentication Code (HMAC) instead of the hash function allows us to obtain other values of the software parameters if the parameter array is disclosed. The disclosure of the parameter array is difficult to detect, but analysis of the anomaly of requests generated by the device can help. We decide not to use Key Derivation Function (KDF) [7] due to the poor performance of KDF, which is a drawback for IoT devices.

We propose two approaches to securely upload and store the array in DL. The first approach is to use a temporary unique ID and a key that must be entered on the device stored on the DL in advance. They are used only once in the registration process and are deleted after use. The second approach is to perform the registration when the device is connected to the application gateway locally in a secure environment or through a secure authenticated link. The parameter array P_A that identifies the device A is passed through the application gateway AG_i of the organization to which the device belongs to node DL_i . The randomly generated 10-byte device identifier ID_A and the parameter array P_A are saved in DL_i . The encrypted identifier ID_A is sent back to the device. To encrypt the ID_A , the DL_i generates k random numbers $PN_A = \{n_1, n_2, \dots, n_k\}$ from the set $\{1, \dots, n\}$, which are used as indexes to the parameters stored in P_A . The values of P_A with indexes n_1, n_2, \dots, n_k are concatenated together, then the hash value from this concatenation is calculated, which is the key K_A . This key is used along with the generated *nonce* to encrypt ID_A and current timestamp. We use Authenticated Encryption with Associated Data (AEAD) to encrypt data. We do not specify what algorithm must be used. It can be, for example, AES-GCM, XChaCha20-Poly1305, Ascon, or Xoodyak. *nonce*, PN_A , and E_A are sent to the device A . Device based on PN_A can reconstruct the key and decrypt E_A . If the encrypted timestamp is correct, then the device accepts ID_A . The use of random identifiers prevents leakage of information about an IoT device, which can occur when using an identifier generated according to some of the proposed standards such as Watson IoT [8] or oneM2M [9]. For example, the WatsonID IoT identifier contains information about the type of device and the name of the organization to which it belongs. Figure 1 presents the device registration process.

3.2. Communication between IoT Device and Distributed Ledger

The communication procedure between the IoT device and the DL node is shown in Figure 2. When device A needs to communicate with DL, e.g., to write *data* to a channel, it must generate k random numbers $PN_A = \{n_1, n_2, \dots, n_k\}$ from the set $\{1, \dots, n\}$, which are used as indexes to the parameters stored in P_A . The values of the indexed parameters are concatenated together, and the hash of the concatenation produces a key K_A . This key and a freshly generated *nonce* _{A} are used to encrypt *data* along with the current timestamp t_A . The encrypted message E_A is sent, together with PN_A , to AG_i , which reconstructs the key K_A , decrypts E_A , and verifies the timestamp t_A . If the difference between t_A and the current time is less than the preconfigured threshold, an appropriate operation is performed on

data. Some possible operations include writing data to a ledger or other database, reading data from a ledger or other database, and verifying permissions. The *response* is sent to the device *A* secured in the same way as the message from the device. After receiving the *response*, device *A* reconstructs the key K_{DL} , decrypts the ciphertext E_{DL} , and verifies the timestamp t_{DL} . If it is correct, the *response* is accepted by device *A*.

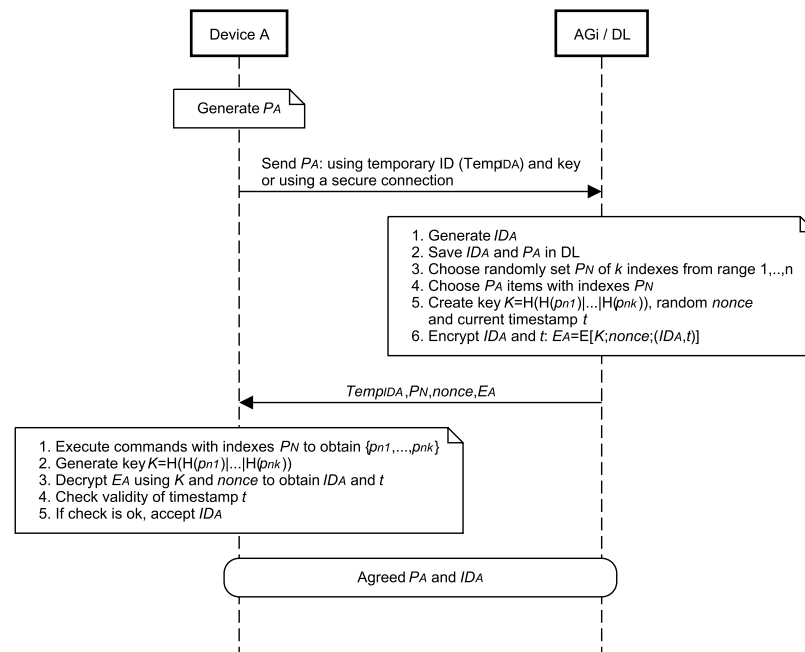


Figure 1. Communication between the IoT device and a ledger node during the registration phase.

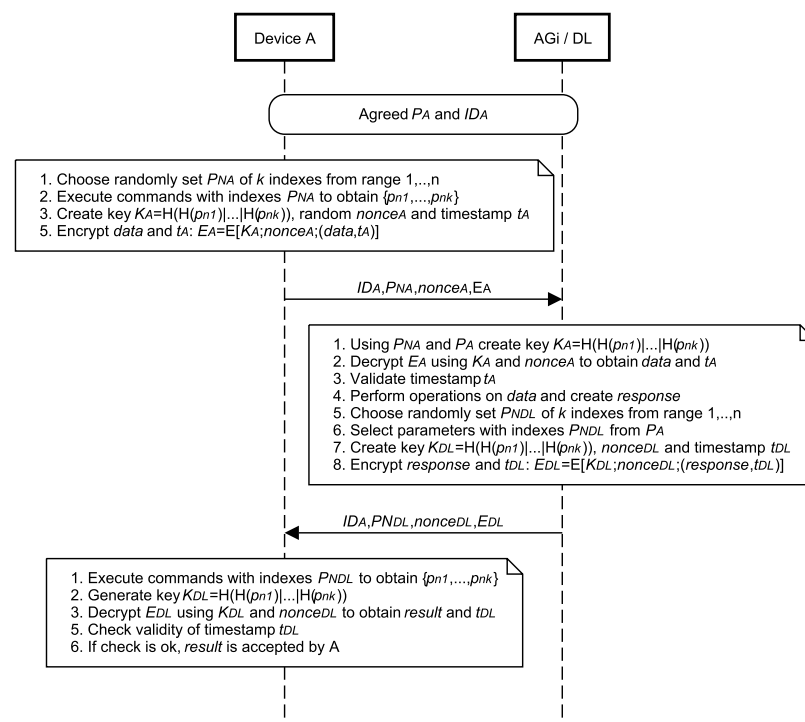


Figure 2. Communication between an IoT device and the distributed ledger.

3.3. Communication between IoT Devices

When IoT device *A* needs to communicate with device *B*, which belongs to different federated organizations, it is necessary to establish secure communication between the devices as shown in Figure 3. This can be achieved by encrypting the messages transmitted with a shared key, generated and delivered securely to the registered IoT devices by DL using the parameter tables of these devices. The shared key is issued only if the devices are allowed to communicate. Device *A* can obtain the ID_B of device *B* in many different ways, e.g., it can receive it from another device, it can have a record that it needs to communicate with that specific device, or device *B* can announce that it has a certain type of information needed by *A*. The procedure starts in the same way as in the case of establishing the communication of device *A* with DL, with *data* containing the identifier ID_B .

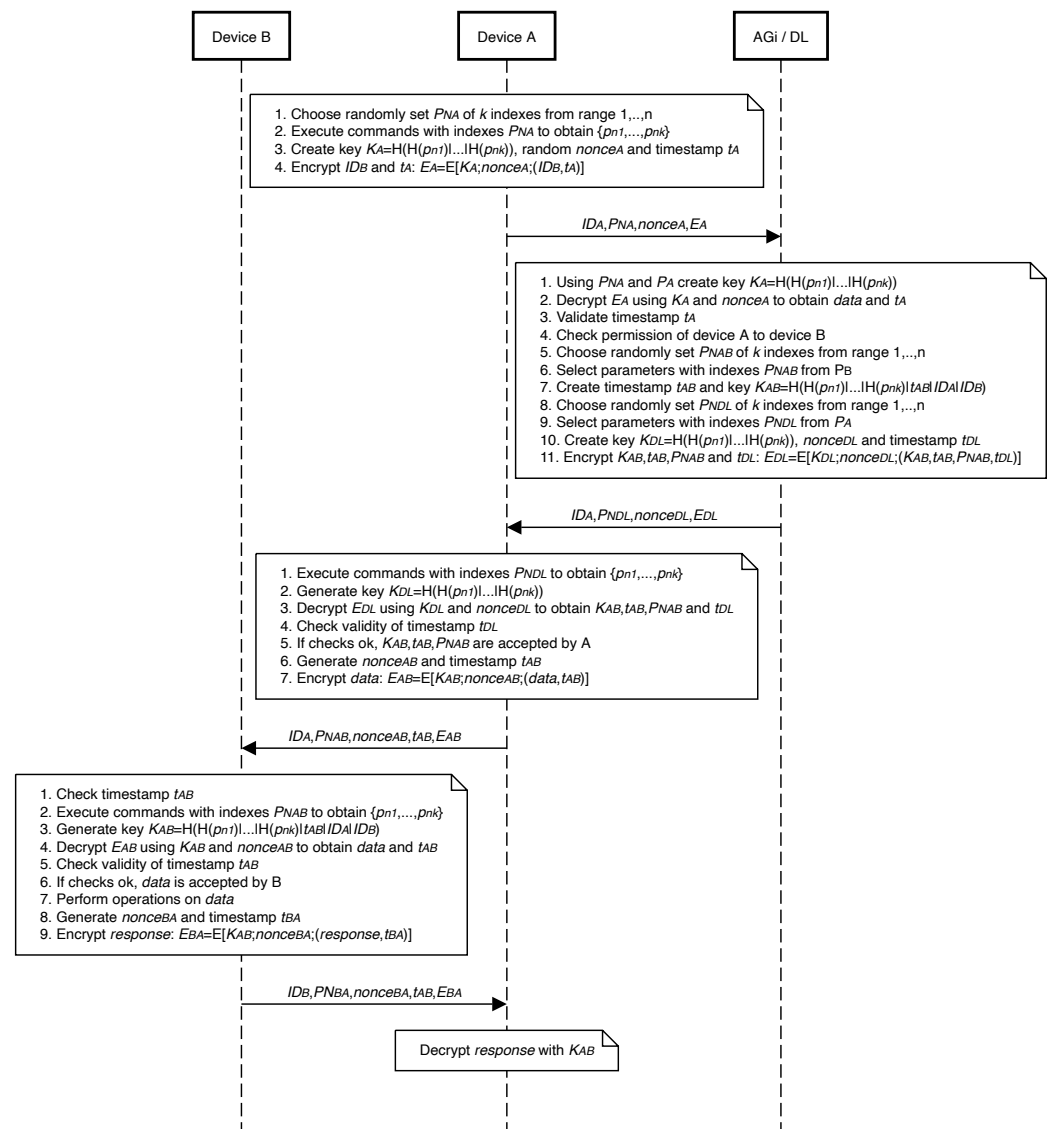


Figure 3. The procedure of communication between IoT devices.

3.4. Authorization of IoT Devices

Authorization is based on the Access Control Lists (ACLs) stored in the DL authorization channel, in which each device has an entry that contains a list of other devices with which it can interact, including allowed actions. Other authorization methods, for example, Role-based Access Control (RBAC), can also be implemented. The type of authorization mechanism must be agreed upon between the members of the federation before establishing

the DL network. The authorization channel is separate from the one used to store the information required for authentication. Permissions for a given IoT device can only be granted or removed by the security administrator of the organization to which the device belongs based on the identified needs of business processes and workflows supported by the devices. Each operation in the authorization channel is performed using the appropriate smart contract. If device A wants to check if it can interact with device B , it sends ID_B and an operation type to DL. Using smart contracts for authorization enables federation partners to formulate fine-grained security policies regarding the utilization of their assets and thus maintain control over information flows and access granted to other federation partners. Moreover, the DL provides a secure audit log for interaction between the devices, enabling security forensics and supporting monitoring and detection for possible faults or security anomalies. Any DL node can perform the check and send a response, which increases the resilience of the federated system to the disconnection of some of the nodes due to adversary activities or infrastructure failures. The communication is secured in the same way as described in Section 3.2.

4. Entropy Analysis

We propose three different sources of cryptographic material in LAAFFI, so it is necessary to examine their suitability by measuring entropy. An aspect of the security of the developed protocol that needs to be evaluated is the preservation of the confidentiality of the transmitted data. This is achieved when only the parties to the communication have access to the transmitted information. In the case where the device generates random strings of bits during the recording process, which are stored in its memory, obtaining randomness in the generation of these strings is possible thanks to the source of entropy provided by the operating system; in the case under study, it is the Linux operating system. Due to the difficulty of obtaining random values in the operating system, the generator used in Linux is pseudorandom. To generate pseudorandom values, environmental noise from the computer system is used, which is difficult for an attacker to reproduce. Sources of randomness include times between keystrokes, interrupts between certain system interrupts, etc., which are non-deterministic and thus difficult to examine by an external attacker who does not have access to the device. Bits from these sources are a whitening or de-biasing operation. This operation is designed to correct errors generated in the source of entropy. At the same time, the amount of entropy is estimated. Entropy estimation is performed using polynomial interpolation for the occurrence times of events of receiving random bits [10,11]. The bits from the sources of randomness are then transferred to the entropy pool. The entropy pool is a collection of random bits stored in the operating system's memory. Bits from the entropy pool are used as seeds in a cryptographically secure pseudorandom generator (Cryptographically Secure Pseudorandom Number Generator (CSPRNG)). As of version 5.6 of the Linux operating system kernel, `/dev/random` and `/dev/urandom` retrieve data from the CSPRNG with the difference that `/dev/random` will not provide data unless CSPRNG is initialized with a seed of sufficiently high entropy.

The process of generating pseudorandom values is shown in Figure 4. For IoT devices, many sources of entropy cannot be used because they do not exist, such as interrupts from the keyboard and hard drive. Therefore, IoT devices can be equipped with hardware random number generators. Some of them have a built-in hardware random number generator, such as Raspberry Pi 3.

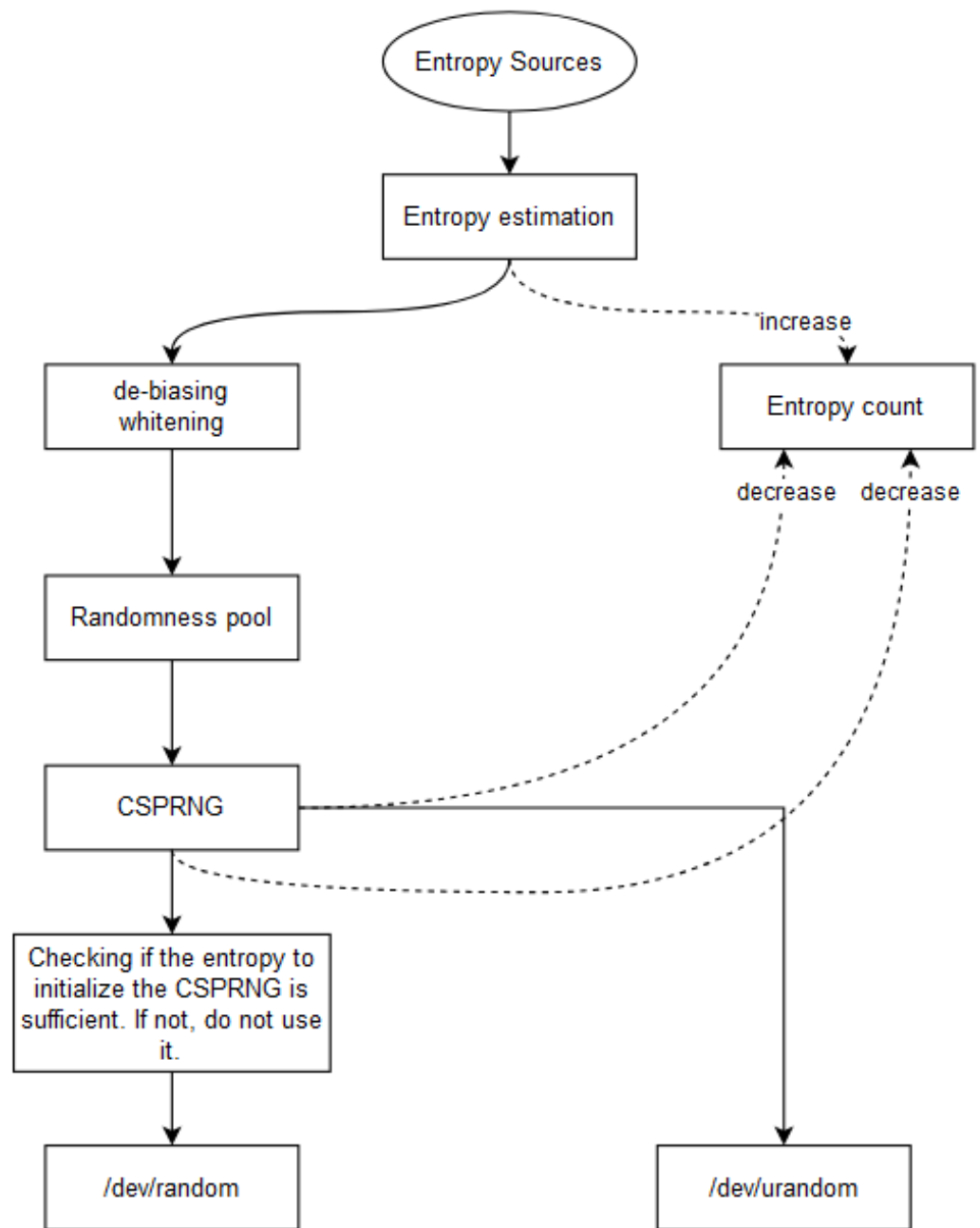


Figure 4. The method of generating pseudorandom numbers in the Linux operating system since version 5.6. Source: Own design based on [12].

In order to evaluate the suitability of the proposed solution, in a situation where PUFs are used, the entropy is evaluated on the basis of the data from the literature. To estimate the entropy of PUF circuits, we use the concept of min-entropy [13]. Min-entropy describes the unpredictability of an outcome defined solely by the probability of the most likely outcome. It is defined by the formula

$$H = \log_2 \frac{1}{p_{max}} \quad (2)$$

where H is the min-entropy, and p_{max} is the probability of correctly guessing the most likely outcome on the first attempt.

Due to the structure of some PUF circuits, it appears that estimating min-entropy can give erroneous results [14]. The estimation of entropy depends on the ordering of the

responses of the PUF circuit, and in some cases, the entropy is overestimated. Therefore, you may encounter the use of the context tree weighting method [15,16], or the tests described in NIST 800-22 are used [17]. In the case of the 800-22 tests, their use is highly questionable because they are prepared to test the randomness of pseudorandom number generators [15]. Other methods of estimating entropy are also proposed in [18] for certain types of PUFs.

In Table 2, we show the estimated entropy values for different types of PUF.

Table 2. Entropy values for different types of PUF circuits.

Type of PUF	Entropy [Bit]	Article
Direct characteristics used	6.6 bit from each sensor	[15]
SRAM	0.76 bit per cell 0.797–0.799 bit per cell	[15] [19]
DRAM	0.95 bit per cell	[20]

In addition to the type of PUF circuits, the entropy value for PUF circuits also depends on other factors, including temperature [21], circuits age [22], and the number of erroneous bits generated by the circuit. The more errors that occur in the PUF response, the lower the entropy that can be obtained [23]; therefore, it is necessary to pay attention to the internal distance between the responses of the PUF circuits. In general, an accurate estimation of the entropy of responses from PUF systems is very difficult, and the result itself is not certain. This is due to the nature of PUF circuits, for which the responses are dependent on physical processes. Despite this, the use of PUF circuits in the LAFFI protocol is possible. However, more of the system's response should be used to create an authentication key so that the size of the expected entropy is considered sufficient. The publication [15] also points out the issue of differences between different PUF layouts. If different PUF layouts have a low Hamming inter-distance value, then by knowing the answers of one system, you can try to guess the answers of another, twin PUF.

For the device parameters, we verify the entropy of the source by determining the min-entropy for each parameter. Since successive values of the following parameter bytes have equal probability of occurrence, we can use the Formula (1). Since the probability of each value for the parameter is identical, you can also use the formula

$$H = L * \log_2 b \quad (3)$$

where H is the min-entropy of the parameter, L is the number of bytes of the parameter (its length), and b is the number of possible characters.

The equal probability for software parameters is due to the fact that software parameters are generated pseudorandomly. They were created on the device during the installation and configuration of the operating system and software. In the case of the developed method, the device itself does not need to have a pseudorandom number generator to generate the software parameters since their values can be generated during the installation of the system (e.g., using an external pseudorandom number generator) outside the IoT device itself, and then the card with the installed and configured system can be placed in the target IoT device. In the case of hardware parameters, the generation of a serial number or other feature is performed at the manufacturing phase of the component. This is assumed to be performed in a pseudorandom fashion. Investigating the randomness of these parameters is controversial since a possible test would involve multiple character strings from different devices, rather than a single string from a single source, which conflicts with the assumptions developed by NIST [17]. These tests require a minimum of 1 million bits, and in the case of the serial number parameter, only 64 bits can be obtained from each device (a serial number has 8 bytes). It is therefore possible to identify three main reasons for abandoning the randomization tests. The first is the requirement to use a sequence of at least several thousand bits, which cannot be obtained from a single device.

One would have to concatenate from multiple devices, which means that the result of such tests will be highly questionable. The second is the cost of acquiring the devices. In order to conduct randomization tests, NIST [17] would be required to have access to a minimum of 15,625 devices (because 1,000,000 bits required/64 bits provided by 1 device), where the cost per device is about PLN 30. The third reason is that the study would have referred to the assessment of randomness for a single manufacturer's model or models. For the same reasons, the randomness of hardware parameters for the memory card is not examined. The article is limited to examining entropy for selected parameters that are considered promising for use in the key generation process. In the first step, parameter values are collected from 20 Raspberry Pi devices, 20 memory cards and 20 installed operating systems. Then, the entropy for each parameter is calculated using the *ent* tool [24]. The results are shown in Table 3.

Table 3. List of parameters with calculated entropy.

No.	Parameter	Entropy [Bit]	Number of Characters
1	Serial number of Raspberry Pi	32	8
2	Hardware ID Raspberry Pi	32	8
3	Memory card serial number	29	8
4	CID Value	33	9
5	PTUUID of partition	19	5
6	UUID of filesystem	128	32
7	UUID of filesystem for FATBOOT	32	8
8	Seed from /etc/shadow	93	16
9	Hash value form /etc/shadow	Depends on the password	88
10	Seed for directory hash function (ext4)	129	32
11	Operating system ID	129	32

Based on the results shown in Table 3, 274 bits of input entropy can be obtained from seven parameters with the lowest entropy, i.e., serial number; hardware ID; memory card serial number; CID value; PTUUID of partition; UUID of filesystem for FATBOOT; and seed from /etc/shadow. This value is sufficient to create a key with an entropy of 128 bits [13]. The entropy value is the result of adding up the entropy for the selected parameters. The entropy values can be summed with each other as long as the parameters are independent. You should be careful when choosing parameters to identify an IoT device, due to certain parameter dependencies introduced by manufacturers. For example, on the Raspberry Pi, the MAC address values and Bluetooth hardware address depend on the serial number of the device. The physical address of the wired interface (MAC address) is the value of B827EB and the lower 48 bits from the serial number, while the physical address of the wireless interface is B827EB and the lower 48 bits from the serial number XORed with 0x555555 mask. It is also important not to use the device's interfaces for communication when using the serial number as one of the parameters because this will lead to the disclosure of the device's serial number information. Some devices may also come from companies that do not pay attention to the uniqueness of the generated serial numbers and other parameters. This problem is encountered while performing tests for memory cards sourced from a popular Chinese online store. These cards have identical serial numbers and CID numbers. Therefore, the test is repeated for Samsung cards. Since the entropy for some parameters is several times higher than what is considered sufficient to create a hard-to-guess key therefore, some parameters can be split into at least two parts. An example is the value of the password hash function, which can have up to 512 bits of entropy. The number of entropy bits for this parameter depends on the algorithm used to create the password hash but also on the entropy of the password used. The operation of splitting the password hash can be performed by using the Pseudo Random Function (PRF), and then dividing the result into parts of an appropriate length analogous to the session key created in the protocol 802.11i [25]. The minimum number of parameters used to create a key depends on the entropy of each parameter. It is important that the sum of the entropy of the parameters used to create the key is twice the expected entropy of the cryptographic key [13]. In the solution proposed in the article, the final key value is

obtained by determining the value of the hash function for the concatenation of the HMAC function values of the individual parameters. The proof that HMAC is a PRF function is presented in the article [26]. We decide not to use the KDF function because its use does not increase entropy. It is also considered that increasing the difficulty of brute-force attacks is debatable for IoT devices. IoT devices have weaker processors, making the execution of KDF time-consuming; besides, the devices must be energy efficient, so the device operations themselves must not be energy intensive either. In securing communications, the focus is on the use of algorithms considered secure, as well as on the difficulty of “guessing” the input data from which the key is formed (increasing the key space). The combination of a secure encryption algorithm, the use of a key with sufficient entropy, and the proper management of access to parameters will translate into ensuring the confidentiality of communications between IoT devices. Prepared configuration parameters may depend on devices or their components that are built in the same time period and have similar serial numbers, as well as software that is installed at the same time on different devices. Therefore, it is necessary to study how the parameter values from different devices, components, and software differ. A similar problem occurs in PUF circuits, where, for these circuits, it is assumed to calculate two parameters determining the repeatability and uniqueness of the response. Because of the similar approach in the developed protocol, it is decided to determine the uniqueness of the parameter values in a manner similar to the evaluation performed for PUF circuits [27]. The study of repeatability of parameter values is considered unnecessary because the parameters are invariant over time, and also environmental factors are not able to change parameter values. In order to test uniqueness based on the collected values for each parameter, the Hamming distance is calculated, and the average μ_{inter} of the parameter values is determined. The parameter values are read from 20 Raspberry Pi 4 devices and Samsung memory cards, which are obtained from the same production run. The operating system on each memory card is installed in a few hours; the name and password for the user, the system name, are identical for each installation. To calculate the Hamming distance, each parameter value is converted into a bit string, and the Hamming distance is calculated for the bit strings. Value μ_{inter} in the best case should be close to 0.5. The value of the average μ_{inter} is calculated using the formula [28]

$$\mu_{inter} = \frac{1}{NL} * \sum_{i=1}^{M-1} \sum_{j=i+1}^M HD(x_i, x_j) \quad (4)$$

where N is the number of all pairs of values of the parameter under test for which the Hamming distance is calculated, L is the length of the parameter value, and M is the number of parameter values, while $\sum_{i=1}^{M-1} \sum_{j=i+1}^M HD(x_i, x_j)$ is the sum of the Hamming distance values for all pairs of values of the parameter under study. The number of all pairs can be calculated from the formula $N = \frac{M*(M-1)}{2}$. Values μ_{inter} are shown in Table 4.

The μ_{inter} values obtained during this study indicate which parameters are best to use for key creation. Note the low μ_{inter} value of the serial number of the memory card and its CID number, which is due to the fact that the cards came from a single set and there were cards with serial numbers following each other. This situation is not observed for serial numbers of Raspberry Pi devices. The μ_{inter} value for the password seed, among others, is close to 0.41, despite the fact that the value of this parameter is generated pseudorandomly. This is probably caused by the fact that the password seed is stored using a set of symbols, and thus not all the bits representing the symbol are used.

In conclusion, each of the three cases is suitable for use in LAAFFI. However, care must be taken that the selected parameters have sufficient entropy to guarantee the secrecy of the encrypted data. In the event of insufficient entropy, more parameters can be used, but the entropy of the selected parameters must be twice the entropy of the resulting key [13].

Table 4. List of parameters with calculated μ_{inter} .

No.	Parameter	μ_{inter}
1	Serial number of Raspberry Pi	0.339
2	Hardware ID of Raspberry Pi	0.328
3	Memory card serial number	0.229
4	CID Value	0.235
5	PTUUID of partition	0.319
6	UUID of filesystem	0.309
7	UUID of filesystem for FATBOOT	0.382
8	Seed from /etc/shadow	0.403
9	Hash value form /etc/shadow	0.410
10	Seed for directory hash function (ext4)	0.320
11	Operating system ID	0.323

5. Formal Security Validation

We formally verify the most important LAAFFI operations: (1) device registration, (2) communication of an IoT device with DL, and (3) communication between two IoT devices. In our validation, we assume that the DL used by LAAFFI is secure. We model and verify the security of each operation in Verifpal [29] to determine whether the three properties of secrecy, authenticity, and freshness are met. Our Verifpal models are available at [30]. In each case, we receive a positive result, which confirms the security of the verified operations of LAAFFI.

Verifpal is one of the tools for modeling modern communication protocols and verifying that security features are met. A characteristic feature of this tool, which is emphasized by the author, is the simple and intuitive assembly that allows for easier model creation for those unfamiliar with modeling issues of communication protocols. Verifpal is a tool that verifies symbolic models in an automatic way. Users can use only defined primitives (e.g., symmetric/asymmetric encryption, hash function, message signing, and many others) and cannot be extended. Verifpal also offers a set of security features that can be verified: the secrecy of transmitted data, authentication of parties to the communication, the possibility of a replay attack, and unlinkability. It has been used to test the security of many protocols, including Signal, TLS, ProtonMail [31]. The results of the analysis of these protocols conducted with Verifpal overlap with the analysis conducted by other tools.

The Verifpal model consists of four main elements: the type of attacker, the operations performed by the participants, the information sent by the participants, and the queries to be verified. Because Verifpal has limited properties verification abilities, the attacker's model itself is simplified. One of the main goals of the attacker in the developed protocol is to gain access to the transmitted data. Access to the data by an unauthorized party is a violation of the secrecy of the transmitted data. In order to read the transmitted information, the attacker must gain access to the key used to encrypt the data. An attacker, having the ability to access the transmitted data, can modify them and use them in other communications by impersonating the device. Another goal may be to attempt a replay attack by resending intercepted messages to disrupt the protocol. In this case, the attacker simply intercepts any message and retransmits it to other devices. Another goal of the attacker may be to impersonate another device. To achieve this goal, the attacker must obtain credentials (login and key). Each of these goals has been verified by the Verifpal model.

5.1. Registration Phase

In each of the models built for Verifpal, an attacker that acts in an active manner must be defined, that is, an attacker that has the ability to modify the data sent between communication participants.

According to the protocol description presented in Section 3, when registering a device A , it generates a parameter array $paramtable$ and a nonce value $nonceAH$, and has a stored one-time key OTP and an identifier ITD . In the real environment, a timestamp is also sent in addition to the parameter array. However, since Verifpal does not have the ability to operate on timestamps, it is decided that it is pointless to create and transmit a timestamp.

In addition, the parameter array is marked as a random value since the arrays and lists are not operated on in the symbolic model. The parameter array *paramtable* is encrypted using the algorithm *AEAD* using a temporary key *OTP* and a one-time value *nonceAH*. The temporary identifier *ITD*, *nonceAH*, and the ciphertext *E_DataAH* are sent to the Distributed Ledger node. A fragment of the model that implements these actions is shown in Listing 1.

Listing 1. A fragment of the model responsible for describing the actions performed by device A in the device registration process.

```
attacker[active]
principal DeviceA[
  generates paramtable
  generates nonceAH
  knows private OTP
  knows private TID
  E_DataAH = AEAD_ENC(OTP, paramtable, nonceAH)
]
DeviceA -> HLF: TID, nonceAH, E_DataAH
```

The Distributed Ledger node also has a temporary device identifier *TID* and a one-time key *OTP*. Using *nonceAH* and *OTP*, it decrypts the received ciphertext *E_DataAH*. The decrypted value received is the parameter array *D_DataAH*. The Distributed Ledger node generates a permanent device identifier *ID*. The parameter array and *ID* in the real scenario are written in the Distributed Ledger when registering the device. A key *key* is generated, which in the real system is derived from the set of parameters specified by *parameter_numbers*. Since it is impossible to represent this action in the Verifpal model, the simplifying assumption is made that the *key* key is known to both parties to the communication. The identifier *ID* is encrypted using the key *key* and the nonce value generated, *nonceHA*. The result of the encryption is the ciphertext *E_DataHA*. The nonce value *nonceHA* and the ciphertext *E_DataHA* are sent to device A. The modeled actions are shown in Listing 2.

Further descriptions of the models on the Verifpal software no longer include information about the need to verify the timestamps in the messages sent in the message and consider that the key is known to both parties of the communication. These simplifications are made because Verifpal does not have the ability to compare timestamps and operate on tables.

Listing 2. Fragment of the model responsible for describing the actions performed by the Distributed Ledger node during device registration A.

```
principal HLF[
  knows private OTP
  knows private TID
  D_DataAH = AEAD_DEC(OTP, E_DataAH, nonceAH)
  generates ID
  knows private key
  generates nonceHA
  E_DataHA = AEAD_ENC(key, ID, nonceHA)
]
HLF -> DeviceA: nonceHA, E_DataHA
```

Device A, upon receiving the response from the node, decrypts the received ciphertext *DataHA* using the key *key* that is known to the device and the nonce *nonceHA* sent in the message. The decrypted *DataID* is considered the permanent identifier of the device, and the device itself is considered registered. A fragment of the model that implements these actions is shown in Listing 3.

Listing 3. Fragment of the model responsible for describing the actions performed by device A after receiving a response in the registration process.

```
principal DeviceA[
  knows private key
  D_DataHA = AEAD_DEC(key, E_DataHA, nonceHA)
]
```

At the end of the device registration model, there is a list of queries that are verified within the model. The confidentiality of the transmitted parameter array (*paramtable*) and the identifier given to the device (*ID*) are verified. The authentication of the communication is also verified, as well as the ability to distinguish messages between protocol runs. All verified security features are met. The list of verified queries is shown in Listing 4.

Listing 4. List of queries verified by Verifpal during device registration process.

```
queries[
  confidentiality? paramtable
  confidentiality? ID
  authentication? DeviceA -> HLF: E_DataAH
  authentication? HLF -> DeviceA: E_DataHA
  freshness? E_DataAH
  freshness? E_DataHA
]
```

5.2. IoT Device Communication with Distributed Ledger

An active attacker is also defined in this model. Device A ‘knows’ which data *dataA* are to be sent to the Distributed Ledger. The generation of the nonce value *nonceAH* follows. Data *dataA* are encrypted with the key *keyAH* and nonce *nonceAH* using the algorithm *AEAD*. A nonce value of *nonceAH* is sent to the Distributed Ledger node, as well as the ciphertext *E_DataAH*. Listing 5 shows a fragment of the model showing the actions performed by the IoT device.

Listing 5. Fragment of the model responsible for describing the actions taken by device A sending data to the Distributed Ledger.

```
attacker[active]
principal DeviceA[
  knows private dataA
  generates nonceAH
  knows private keyAH
  E_DataAH = AEAD_ENC(keyAH, dataA, nonceAH)
]
DeviceA -> HLF: nonceAH, E_DataAH
```

After receiving the data, the node decrypts the ciphertext *E_DataAH* using the key *keyAH* (reconstructed from the values of certain parameters stored in the array during device registration) and the transmitted value *nonceAH*. To send the *reply* response, the node generates a new key *keyHA* and the nonce value *nonceHA*. The *reply* response is encrypted using the *AEAD* algorithm using the created key *keyHA* and nonce *nonceHA*. The ciphertext *E_DataHA*, and the nonce value *nonceHA* are sent to device A. The model that implements this part of the protocol is shown in Listing 6.

Device A knows the key *keyHA* and, using this key and the received value *nonceHA*, decrypts the ciphertext *E_DataHA*. In doing so, it obtains the answer from the Distributed Ledger node. Listing 7 shows a fragment of the model responsible for the actions performed by A.

Listing 6. Fragment of the model responsible for describing the actions performed by the Distributed Ledger after receiving data from device A.

```
principal HLF[
  knows private keyAH
  D_DataAH = AEAD_DEC(keyAH, E_DataAH, nonceAH)
  generates reply
  generates nonceHA
  knows private keyHA
  E_DataHA = AEAD_ENC(keyHA, reply, nonceHA)
]
HLF -> DeviceA: nonceHA, E_DataHA
```

Listing 7. Fragment of the model responsible for describing the actions performed by device A after receiving data from the Distributed Ledger.

```
principal DeviceA[
  knows private keyHA
  D_DataHA = AEAD_DEC(keyHA, E_DataHA, nonceHA)
]
```

This model also verifies the confidentiality of the transmitted data, the authentication of the communication parties, and the ability to distinguish messages sent in different protocol runs. The queries verified by Verifpal for this model are shown in the Listing 8. All verified security features are verified successfully.

Listing 8. List of queries verified by Verifpal in the process of device communication with Distributed Ledger.

```
queries[
  confidentiality? dataA
  confidentiality? reply
  authentication? DeviceA -> HLF: E_DataAH
  authentication? HLF -> DeviceA: E_DataHA
  freshness? E_DataAH
  freshness? E_DataHA
]
```

5.3. Communication between IoT Devices

Communication between devices requires the IoT device that wants to communicate with another IoT device to first obtain a key from the Distributed Ledger node. The attacker, as in the previous two models, can modify the messages transmitted. To obtain the communication key, device A encrypts the device identifier B using the algorithm AEAD using the key *keyAH* and the nonce *nonceAH*. The ciphertext of *E_DataAH* along with *nonceAH* is sent to the Distributed Ledger node. The model that implements this fragment is shown in Listing 9.

Listing 9. Fragment of the model responsible for describing the actions performed by device A sending a request to the Distributed Ledger to generate a key to communicate with another device.

```
attacker[active]
principal DeviceA[
  knows private dataAB
  knows public IDB
  generates nonceAH
  knows private keyAH
  E_DataAH = AEAD_ENC(keyAH, IDB, nonceAH)
]
DeviceA -> HLF: nonceAH, E_DataAH
```

The Distributed Ledger node, upon receiving the message, decrypts the ciphertext E_DataAH using the key $keyAH$ and the transmitted nonce $nonceAH$. Using the algorithm AEAD using the key $keyHA$ and the nonce $nonceHA$, the created key for inter-device communication $keyAB$ is encrypted. The ciphertext E_DataHA along with $nonceHA$ are sent to device A . Listing 10 shows an excerpt from the model describing the actions performed by the Distributed Ledger node.

Listing 10. A fragment of the model responsible for describing the actions performed by the Distributed Ledger upon receiving a request to generate a key for communication between devices A and B .

```
principal HLF[
  knows private keyAH
  D_DataAH = AEAD_DEC(keyAH, E_DataAH, nonceAH)
  knows private keyAB
  generates nonceHA
  knows private keyHA
  E_DataHA = AEAD_ENC(keyHA, keyAB, nonceHA)
]
HLF -> DeviceA: nonceHA, E_DataHA
```

Device A , upon receiving the message, decrypts it and thus obtains the $keyAB$ key to communicate with device B . The corresponding part of the model that implements these steps is shown in Listing 11.

Listing 11. Fragment of the model responsible for describing the actions performed by device A before sending data to device B .

```
principal DeviceA[
  knows private keyHA
  D_DataHA = AEAD_DEC(keyHA, E_DataHA, nonceHA)
]
```

In this model, as in the previous, the confidentiality of the transmitted data, the authentication of the parties to the communication, and the ability to distinguish messages sent in different runs of the protocol are verified. The verified features are shown in Listing 12.

Listing 12. Query list verified by Verifpal in the process of establishing communication between device A and device B .

```
queries[
  confidentiality? IDB
  confidentiality? keyAB
  authentication? DeviceA -> HLF: E_DataAH
  authentication? HLF -> DeviceA: E_DataHA
  freshness? E_DataAH
  freshness? E_DataHA
]
```

Verifpal finds one possible attack when analyzing the properties of this model. It is presented in Listing 13.

Listing 13. The result of the execution of the communication establishment model showing the first of the possible attacks.

```
Result : confidentiality? idb -
        idb (idb) is obtained by Attacker.
```

It refers to the failure to ensure the secrecy of the transmitted device B identifier to the Distributed Ledger node. This is because the identifier is publicly known. It is determined that this is not a problem that affects the security of the system. In a real system,

the identifier is sent with a timestamp that is verified; therefore, there is no possibility of a replay attack.

The second model includes communication between the *A* and *B* devices after the *A* device has received the *keyAB* key. The *dataAB* data are encrypted by the *A* device using the algorithm AEAD using the *keyAB* key and the generated nonce value *nonceAB* for each transmitted message. The received ciphertext *E_DataAB* is, together with *nonceAB*, sent to the *B* device. Listing 14 shows a fragment of the model that implements the establishment of communication between the *A* device and the *B* device.

Listing 14. An fragment of the model showing the operations performed by the A devices.

```
attacker [ active ]
principal DeviceA [
  knows private keyAB
  knows private dataAB
  generates nonceAB
  E_DataAB = AEAD_ENC(keyAB, dataAB, nonceAB)
]
DeviceA -> DeviceB: nonceAB, E_DataAB
```

In the real scenario, device *B* is able to create the *keyAB* key based on the number of parameters, its parameter array, and the transmitted timestamp. If this key, together with the transmitted nonce value, allows the decryption of the transmitted ciphertext, it means that device *A* has the right to communicate with device *B*, because otherwise the ledger node would not issue a valid key. In the model, we assume that the key *keyAB* is known to device *B*. Device *B* using this key and the nonce value *nonceAB* can decrypt *E_DataAB*. As a result of this operation, device *B* obtains the data *DataAB* on which it performs some operation. The response *reply* is encrypted with the algorithm AEAD using the key *keyAB* and the new nonce value *nonceBA*. The resulting ciphertext is *E_DataBA*, which is sent along with *nonceBA* to device *A*. Device *A*, using the *keyAB* it has and the value of the nonce received *nonceBA*, is able to decrypt *E_DataBA*, resulting in a *reply* response. Listings 15 and 16 show the modeled communication between *B* and *A*.

Listing 15. Fragment of the model showing the operations performed by device B after receiving a message from device A.

```
principal DeviceB [
  knows private keyAB
  D_DataAB = AEAD_DEC(keyAB, E_DataAB, nonceAB)
  generates reply
  generates nonceBA
  E_DataBA = AEAD_ENC(keyAB_2, reply, nonceBA)
]
```

Listing 16. Fragment of the model showing the operations performed by device A after receiving a response from device B.

```
DeviceB -> DeviceA: nonceBA, E_DataBA
principal DeviceA [
  D_DataBA = AEAD_DEC(keyAB, E_DataBA, nonceBA)
]
```

This model verifies the same properties as the previous models, that is, the confidentiality of communications, authentication of transmitted messages, and the ability to distinguish messages sent in different protocol runs. The list of verified queries is shown in Listing 17.

Listing 17. Query list verified by Verifpal in the process of communication between device A and device B.

```
queries [
  confidentiality? dataAB
  confidentiality? reply
  authentication? DeviceA -> DeviceB: E_DataAB
  authentication? DeviceB -> DeviceA: E_DataBA
  freshness? E_DataAB
  freshness? E_DataBA
]
```

As a result of verifying this model, Verifpal indicates the possibility that the authentication of messages transmitted between *A* and *B* devices is not fulfilled. Verifpal shows that an attack is possible, involving the swapping of transmitted messages, that is, a message sent from device *A* to device *B* is intercepted by an attacker and sent again to device *A*. In the second case, the situation is analogous, i.e., the attacker intercepts the message sent from device *B* to device *A* and sends it to device *B* in the next iteration. In both cases, the result of Verifpal's analysis of the models confirms that the authentication of the parties would not be provided; however, in the actual system, both messages will not be executed by the devices. This is because the devices will receive data that will not be in the expected format, i.e., device *A* will receive a message built as a request and expect a response, while device *B* expects a request and will receive a response. Both attacks are shown in Listings 18 and 19.

Listing 18. The result of executing a device communication model showing the first of possible attacks.

```
Result : authentication? Devicea -> Deviceb: e_dataab---When:
  nonceab -> nonceba <- mutated by Attacker (originally nonceab)
  e_dataab -> AEAD_ENC(keyab, reply, nonceba) <- mutated by Attacker (originally AEAD_ENC(
    keyab, dataab, nonceab))
  d_dataab -> reply <- obtained by Attacker
  d_dataaba -> reply <- obtained by Attacker
  e_dataab (AEAD_ENC(keyab, reply, nonceba)), sent by Attacker and not by Devicea, is
    successfully used in AEAD_DEC(keyab, AEAD_ENC(keyab, reply, nonceba), nonceba) within
    Deviceb's state.
```

Listing 19. The result of executing a device communication model showing the second of possible attacks.

```
Result : authentication? Deviceb -> Devicea: e_dataaba---When:
  d_dataaba -> dataab <- obtained by Attacker
  nonceba -> nonceab <- mutated by Attacker (originally nonceba)
  e_dataaba -> AEAD_ENC(keyab, dataab, nonceab) <- mutated by Attacker (originally AEAD_ENC(
    keyab, reply, nonceba))
  d_dataaba -> dataab <- obtained by Attacker
  e_dataaba (AEAD_ENC(keyab, dataab, nonceab)), sent by Attacker and not by Deviceb, is
    successfully used in AEAD_DEC(keyab, AEAD_ENC(keyab, dataab, nonceab), nonceab) within
    Devicea's state.
```

Another issue with this model is the possibility of sending the same message repeatedly, i.e., launching a replay attack. In the real scenario, it is impossible to perform this attack because each message contains a creation timestamp that is verified. The IoT device or Distributed Ledger node when decrypting the message compares this timestamp with the current time. The disadvantage of this solution is having a correctly synchronized time on each IoT device. The time on the IoT devices should be synchronized with the organization's time server in a way that ensures time server authentication and communication confidentiality. In the prepared implementation, the time after which a message is rejected

because it is considered out of date is 5 s. The timestamp is also compared to the timestamp received in the previous message. If it is identical or smaller, the message is rejected. The timestamp is sent along with the data in encrypted form using the AEAD algorithm.

6. Analysis of Attack Resilience

IoT networks are exposed to many risks due to the valuable data that are collected and processed. These data can be related to people’s health, protected data from an organization, or data critical to public safety. Conducting various types of attacks on IoT systems is made easier due to the large number of devices with limited resources, which requires the use of appropriate security solutions. Devices often come from different manufacturers and do not use specific standards, making them difficult to manage, thus impacting security. The goal of an attacker in IoT networks is most often to steal data, interrupt transmission, and use IoT devices to perform other attacks including DDoS, falsifying data processed in IoT networks. To achieve the mentioned goals, the attacker can carry out various attacks. The attacks that can be carried out on federated networks IoT are the same as those that can be carried out on non-federated networks. However, when analyzing attacks on federated networks, it is important to consider an attacker that belongs to a different organization than the devices or services being targeted. An attacker not belonging to an organization that owns a device or service is likely to have more rights than a non-federated entity but less than a member of an organization that owns devices and services. Table 5 presents a list of possible attacks against the developed protocol LAAFFI. The criterion for selecting attacks is the possibility of violating the properties of information security. The list of attacks is based on an analysis of the existing literature [32–36]. Some of the attacks listed in Table 5 are generalized categories that encompass different attack techniques. An example of such an attack is the Denial of Service (DoS) attack, which can include flood attacks, fragmentation attacks, and reflection amplification attacks.

LAAFFI works in all three layers of the IoT model: perception, network, and application. It is exposed to security attacks performed at each of these layers. Below, we start our discussion with an analysis of general attacks, applicable to all layers, and then continue with a discussion focused on specific layers of the IoT stack. We also provide a dedicated discussion of attacks applicable to Hyperledger Fabric as one of the key third-party components of our framework.

Table 5. Types of attacks that can be attempted against LAAFFI.

Perception Layer	Network Layer	Application Layer
DoS/DDoS Spoofing Cryptanalysis	DoS/DDoS Spoofing Cryptanalysis	DoS/DDoS Spoofing Cryptanalysis
Malicious code injection Physical damage Replacing a device or its components Adding a malicious device	Eavesdropping Replay attack Packet injection Session hijacking	Malicious code injection Data loss Unauthorized access to data Abuse of authority
Device cloning Sybil attack Side-channel attack	Man-in-the-middle attack	Permissions modification by unauthorized users

In the following, we do not report the results of the penetration testing of specific components of LAAFFI. This is due to the fact that our work is focused on the development of a research prototype, which would require some implementation adjustments for operational deployment, and the result of penetration tests are relevant only for a specific implementation instance. However, several penetration studies for the specific components used in LAAFFI, such as Hyperledger Fabric or IPFS, can be found in the literature, e.g., [37].

6.1. General Attacks

An attack that is applicable in various forms to all layers and is difficult to protect against is the (Distributed) Denial-of-Service (DoS/DDoS) attack. The specific type of DoS/DDoS that is applicable to the perception layer is a resource deprivation attack [38]. The aim of the attack is to exploit an available resource; most often, this resource is the network bandwidth, CPU performance, and less often disk space. IoT devices are susceptible to this type of attack due to the hardware characteristics. In order to successfully perform a DoS/DDoS attack, a sufficiently large number of messages need to be sent to the device. The number of messages depends on the hardware configuration of the IoT device. In the case of the network layer [38,39], the DoS/DDoS attack can consist of jamming communication between devices. The attacker transmits a signal using the same frequencies as the devices that communicate with each other. As with the DoS/DDoS attack at the perception layer, the way to protect communication parties from this type of attack is very limited. In the application layer, LAAFFI is vulnerable to DoS/DDoS attacks due to the fact that if too many operations are performed that require adding data to the ledger, e.g., device registration and changing permissions, the Hyperledger Fabric resources can be exhausted. In other cases, there is no limit to the scaling of the ledger nodes. Adding more nodes allows more operations to be processed per second, thus forcing the attacker to increase the number of operations to carry out this attack successfully. Protecting against this type of attack is extremely difficult, but an application gateway may have the ability to reject messages that are duplicates or have an incorrect format. This rejection process can reduce the number of messages that must be handled by the Distributed Ledger nodes. In addition, you can try to protect your organization from a DoS/DDoS attack by taking advantage of the protection services against such attacks offered by telecom operators, or the Web Application Firewall (WAF).

Spoofing attacks [40] encompass a large set of attacks aimed at impersonating, in the case of LAAFFI, a device, an application gateway, or a Distributed Ledger node. Since in LAAFFI, each message is authenticated, it is not possible to perform attacks of this type until the attacker has access to the parameters from which the key can be created. Mutual authentication of the application gateway with the Distributed Ledger node is performed using Transport Layer Security (TLS) based on the certificates of both parties. Therefore, it is important to secure the private key of each application gateway and Distributed Ledger node so that an attacker does not gain the ability to impersonate the application gateway or Distributed Ledger node.

Cryptanalysis [41] encompasses a collection of attacks aimed at obtaining the plaintext of a transmitted message. Examples of this attack include brute-force attack, dictionary attack, and statistics attack. In order to make it difficult to obtain the plain text of the message sent between the IoT device and the application gateway or another IoT device, it is recommended to use encryption and HMAC algorithms that are considered secure. The parameters used to create the key must also have the right amount of entropy so that an attacker cannot guess the key used to secure the message.

Another class of general attacks exploits vulnerabilities in software languages [42] or in the software itself [43]. You can protect yourself from these attacks by updating the software that you use. Attacks can also target a specific software solution and exploit flaws in that software. In the case of a developed implementation of the LAAFFI protocol, an attacker may try to exploit flaws in the Hyperledger Fabric, which we discuss later. The defense against malicious code injection [34,44] in the case of LAAFFI requires ensuring that the user application has a properly implemented mechanism for validating the received data. To prevent this type of attack, it is necessary to implement data validation in the application gateway code and chaincode. Each member of the federation is responsible for maintaining and updating the application gateway. Accordingly, each of them must implement such a mechanism. It should also be possible for each organization's application gateway to be security tested by the other organizations. In the case of chaincodes, the consent of all members is required for a chaincode to be implemented. Therefore, security testing of the

new version of the chaincode should be carried out before accepting the new version of the chaincode.

6.2. Perception Layer

In the *perception layer*, which includes IoT devices, possible attacks on LAAFFI include, in order of increasing technical complexity, the following:

- **Physical damage:** The attack consists of damaging the IoT device and thereby preventing or limiting the ability of the device to provide services. The developed protocol makes it possible to quickly deploy a larger number of IoT devices, which will result in reducing the impact of damage to individual IoT devices.
- **Replacing a device or its components:** This is an attack involving replacing a device or device components. The key generated in the LAAFFI protocol to encrypt and authenticate messages is generated based on the responses obtained from PUF circuits or based on parameters representing selected hardware and software features of the IoT device. In both cases, the attack is detectable based on checking the number of rejected messages against the number of message decryption failures created by the substituted device. The detection of this attack is not possible when the key is created from parameters stored on a memory card.
- **Adding a malicious device [34]:** In the case of this attack, the attacker adds his device to the federated IoT environment. In the case of the developed protocol, without registering the device, there is no possibility of communication with other components of the system. On the other hand, in a situation where the attacker has succeeded in registering his malicious device, the device can only communicate and perform operations with the devices and services to which it will be granted permissions.
- **Device cloning [44]:** An attack on an IoT device involves cloning the device. The cloned device behaves like a real node but engages in malicious activities and conducts wormhole, blackhole attacks because it has access to sensitive data. To carry out a device cloning attack, it is required to obtain the credentials of another IoT device. In the case of LAAFFI, the attack is possible if the attacker takes control of another device with permissions to obtain all the parameter values needed to create the keys. If the LAAFFI uses the hardware–software parameters of the device or the PUF, cloning the memory card is insufficient to carry out this attack.
- **Sybil attack [45]:** The attack involves creating multiple identities for a single IoT node. For the developed protocol, it is possible to register a IoT device more than once. The device is registered by one organization, so it is important that each organization carefully supervises the registration process. During the registration process, it is recommended to verify that the device has not been registered before.
- **Side channel attack [46]:** This is an attack in which the characteristics of the device [47] are observed to obtain information about the algorithm, key, or unencrypted data used. The attacker analyzes the duration of operations, power usage, or electromagnetic emissions [48]. It should be noted that the possibility of executing this attack depends on the design of the IoT device and the technologies used. For this reason, the study of the possibility of launching a side-channel attack is abandoned. It should be assumed that if the attacker has access to the IoT device, it should be considered that the device has been compromised and should be considered untrusted.

6.3. Network Layer

At the *network layer*, the attacker can try to perform the following attacks on LAAFFI, listed in order of their increasing technical complexity:

- **Eavesdropping [49]:** In the case of LAAFFI, the transmitted data is encrypted. The only data that are transmitted in plaintext are the device ID, the parameter numbers used, and the nonce value. These data do not provide a way to create the key used to secure the transmitted data.

- **Replay attack [50]:** This is an attack that is based on intercepting a message and sending it later. In the case of the developed solution, it is impossible to perform this attack due to the fact that each message contains a creation timestamp, which is verified. The IoT device or Distributed Ledger node when decrypting the message compares this timestamp with the current time. The disadvantage of this solution is having a correctly synchronized time on each IoT device. The time on the IoT devices should be synchronized with the organization's time server in a way that ensures authentication of the time server and confidentiality of the communication. In the prepared implementation, the time after which a message is rejected due to the fact that it is considered out of date is 5 s. The timestamp is also compared with the timestamp received in the previous message. If it is identical or smaller, then the message is rejected. The timestamp is sent along with the data in encrypted form using the AEAD algorithm.
- **Packet injection [44]:** The attack involves injecting packets to disrupt communication. In the case of LAAFFI, the attack can consist of duplicating transmitted packets or creating invalid packets. In both cases, they will be ignored by the recipient.
- **Session hijacking [44]:** The attack involves taking over a session token created either between a device and an application gateway or between two devices. LAAFFI does not use sessions, so it is not possible to take over a session. However, it may happen that the attacker takes over the key for communication between two devices created by the Distributed Ledger node. In such a situation, the attacker has the ability to decrypt the communication between these devices and can impersonate any device in this communication. To prevent this, it is important to protect the key used for this communication.
- **Man-in-the-middle attack [51]:** In this attack, the attacker eavesdrops and modifies the message sent between the parties. The message in the developed protocol consists of elements whose modification is easy to detect. The identifier, parameter numbers, and nonce value are necessary to create the key and decrypt the message. Changing any element will prevent the creation of a valid key and decryption of the transmitted ciphertext. Modification of the ciphertext will prevent correct decryption, which can be easily detected through the use of an encryption algorithm that uses an AEAD.

6.4. Application Layer

In the *application layer*, which includes the Distributed Ledger and the application gateway, an attacker can attempt the following attacks, listed below in order of their increasing technical complexity:

- **Data loss:** LAAFFI uses a Distributed Ledger to store data, so each ledger node has a copy of the data. The loss of a single node does not make the data unavailable. Moreover, data stored in the ledger cannot be deleted or overwritten.
- **Unauthorized access to data:** To prevent unauthorized access to the data, LAAFFI relies on the use of a private Distributed Ledger with authorized nodes. Hyperledger Fabric allows for the creation of private channels and therefore restricts access to the data stored in the Distributed Ledger with the need to respect the conditions stored in the chaincodes. To prevent unauthorized access stored in the Distributed Ledger, every user, Distributed Ledger node, and application gateway is authenticated using Public Key Infrastructure (PKI). The certificates are issued only to authorized individuals. It is also important to properly manage users' access rights so that they only have access to the required data types and functions. For example, only an entity with a certificate and the appropriate key pair has the ability to create a Distributed Ledger node and an application gateway.
- **Abuse of authority—**In the case of LAAFFI, the possibility of a successful abuse attack is minimized through the use of chaincodes, which are designed to verify every operation performed on data stored in the Distributed Ledger. For this reason, each

organization has, among other things, the ability to modify the permissions of its IoT devices, but this cannot be performed by unauthorized entities.

- Modification of permissions by unauthorized users: In the LAFFFI protocol, the modification of permissions is only possible through chaincodes, and thus any modification requires the conditions written in the chaincode to be met. If the chaincode is written correctly, only the device owner or authorized entity can modify the device’s IoT permissions.

6.5. Hyperledger Fabric

The access control mechanism in Hyperledger Fabric strictly relies on trust in the organizations that create the ledger in the channel. If a channel member acts maliciously, they can execute, for example, a Wormhole attack [52], or expose the information stored in the channel. Against this attack, we can protect ourselves by storing encrypted data. Alternatively, private data can be stored outside of the Distributed Ledger, e.g., with only the hash value of the data being stored in the ledger. Another significant security challenge is the protection of the Membership Service Provider (MSP) from the leakage of the private key of the CA and administrators. If an attacker takes control of an organization’s MSP or obtains administrator permissions, they will be able to perform any operation on behalf of that organization, including creating new nodes, reporting changes to chaincodes and the Hyperledger Fabric system configuration. Compromising one organization can lead to compromising the entire Distributed Ledger. Another problem is possible errors in chaincodes. Hyperledger Fabric’s chaincode runs in a sandbox, which makes it difficult to exploit security vulnerabilities due to the fact that the process is isolated and the container cannot be accessed from the network. However, once control of the node on which the chaincode is running is taken, the attacker has an open path to analyze and exploit the chaincode vulnerabilities. Vulnerabilities can have various effects; they can lead to, among other things, the leakage of information to unauthorized entities. To avoid errors when writing chaincodes, chaincodes should be thoroughly tested before implementation. For this, static and dynamic analysis [53,54] or fuzzers [55] can be used. During the operational phase, based on event analysis, alerts can be generated if attempts are detected to attack Distributed Ledger nodes [56].

7. Implementation

Taking into account the desirable characteristics of DL identified in Section 3, we compare the most widely used DL as shown in Table 6. We choose HLF as the most suitable type of DL. Compared to some Distributed Ledger implementations, all implementations listed in Table 6 have the ability to execute a smart contract. This feature is crucial for running the LAFFFI because each transaction must meet the requirements defined in this contract. HLF supports creating channels and compartmentalizing stored data, including private data storage, that is, one can define which nodes store the data while only the hash of the data is stored in the ledger. HLF is also modular; that is, one can change the type of database and the consensus mechanism.

Table 6. Comparison of distributed ledgers.

Feature	Hyperledger Fabric	IOTA	Quorum	Ethereum
Limited access to confidential data	Yes			No
Private data				
Modularity				
Separate ledgers				
Smart contracts				
Scalability and throughput	High			Low

Modularity and support for separate ledgers are specific features of the Hyperledger Fabric, enabling secure compartmentalization. The privacy of the data stored is crucial for LAAFFI, as unauthorized access to the data stored in the ledger allows an unauthorized entity to perform any operation as a trusted IoT device. In addition to using a private data store, Hyperledger Fabric also offers an extensive access control mechanism based on Access Control Lists and roles. Due to the fact that LAAFFI requires a large number of read operations per second, the Distributed Ledger must have high throughput and scalability. Scalability allows the number of operations processed to be increased by adding more Distributed Ledger nodes. Hyperledger Fabric, Quorum, and IOTA offer high performance, while Ethereum does not, due to the type of Distributed Ledger and the consensus mechanism used. Finally, the latency time to receive data in a read operation for Hyperledger Fabric is identical to that of a normal database, due to the fact that Hyperledger Fabric uses a separate database to store the latest values. This is also a feature that other Distributed Ledger implementations do not have by default.

Figure 5 illustrates the interactions between the three main components of LAAFFI: (1) IoT device, including the client application and the Authentication and Authorization (AA) service to support communication with HLF nodes and other IoT devices; (2) application gateway, which is an interface between the devices and the HLF nodes; and (3) HLF node running on a normal computer or in the cloud and interconnected to the SIEM and the performance monitoring system.

Communication between the IoT device and the application gateway, depicted as interface A in Figure 5, takes advantage of Constrained Application Protocol (CoAP) [57] and Concise Binary Object Representation (CBOR) [58], as they were specifically designed for constrained devices. Although CBOR Object Signing and Encryption (COSE) [59] provides a security layer for CBOR, we do not use it because it requires encryption of the entire message, and, in our case, only the part with the data is encrypted. Since HLF supports secure communication only over TLS, the application gateway mediates communication between IoT devices using LAAFFI (interface A) and HLF nodes using TLS (interface B). The application gateway can also serve as a load balancer to ensure that queries are evenly distributed among all DL nodes. It can also filter malicious traffic before it reaches the DL nodes, thus shielding them from handling rogue requests. Each organization should operate at least one application gateway; if the organization wants to keep high availability, then there need to be at least two application gateways. Communication with other services, e.g., SIEM, performance monitoring tools, or cloud services, can occur through the application gateway (interface C); this ensures that cloud services do not have to support the LAAFFI framework we propose, and communication between cloud services and the application gateway is performed in a standard way, e.g., using TLS.

We use three HLF channels, depicted on the right side of Figure 5. *Registration channel* stores data used during the device registration process. *Authentication channel* stores the arrays of parameters required for device authentication; chaincode stored in this channel is used to encrypt and decrypt messages and create keys for communication with other devices. *Authorization channel* stores all access control policies; the chaincode in this channel checks whether the devices can communicate with each other and which operations one device can perform on another device. When one of the organizations is considered malicious or compromised, we recommend using the HLF "Private data" function to prevent malicious activities of other organizations. The credentials of IoT devices are stored only at the nodes of the organization that owns those IoT devices. If one organization is compromised, it will not have the credentials of other organizations' devices and thus cannot eavesdrop on communications or impersonate other organizations' devices.

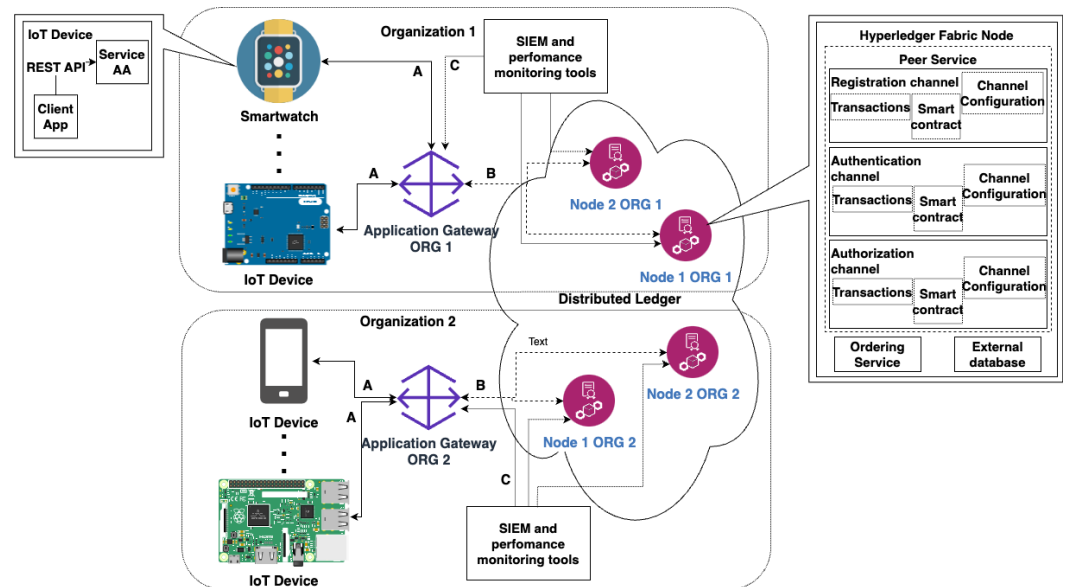


Figure 5. A federated IoT network with two organizations with two DL nodes each.

8. Performance Evaluation

When conducting performance evaluations of distributed computing systems, such as our IoT framework based on Hyperledger Fabric, three main metrics must be considered: latency, throughput, and network resource utilization rate (the transmission overhead introduced by the protocol). Latency measures the time it takes for an operation to be performed by the IoT device and the distributed register. Throughput, on the other hand, measures the speed at which work is performed, specifically the number of operations or transactions performed per second. Network resource utilization, while also related to throughput, is expressed in the number of bytes transferred per second, providing a different perspective on the system's performance.

Evaluating the performance of a framework system using the LAFFI protocol is a complex task. The overall performance evaluation of the system's operation can be significantly influenced by a variety of factors, including input parameters and processing scenarios. These factors can range from system configuration and lower layer protocols used to the hardware configuration of the application gateway and distributed registry nodes, making the evaluation process intricate and multifaceted:

1. node load (e.g., number of ordered transactions/unit of time);
2. Configuration of the consensus process;
3. Protocols used;
4. The type of network used (e.g., Wi-Fi);
5. Hardware configuration of the application gateway, number of application gateways;
6. Performance of other tasks (e.g., system update and backup);
7. Number of application gateways, number of DL nodes;
8. Number of organizations forming a federation.

The main objective of the research is to confirm the usability of the developed solution (LAFFI protocol and system framework architecture) in specific applications. In order to do this, it is necessary to consider basic scenarios of system use for which performance tests should be conducted. The main usage scenarios are the following:

1. To carry out device registration—the most important indicators are the number of device registration operations per second, the latency of the registration process, the number of redundant bytes transferred between the deviceIoT and the application gateway;
2. The establishment of a secure channel for communication with the distributed registry and other services—the most relevant indicators include the number of encryption

and decryption operations per second depending on the amount of data, the number of privilege verification operations per second, and the latency of the distributed registry nodes to perform operations;

3. The establishment of a secure communication channel between devices belonging to different domains—the most relevant indicators are the number of operations to generate a key for communication between IoT devices, the number of privilege verification operations per second, and the latency of the distributed register nodes to perform operations.

To determine whether the proposed solution is scalable, it would be necessary to examine how metrics related to the performance of DL nodes change depending on the hardware configuration of the nodes, the number of nodes in the organization, and the number of organizations that form the federation.

We evaluate the impact of LAAFFI on the performance of IoT devices, DL, and the network. A service implemented in Golang on Raspberry Pi (RPi) provides all the functions required on the IoT device, i.e., device registration, encryption and decryption of messages, checking permissions, communication with other IoT devices, and a Representational State Transfer (REST) Application Programming Interface (API) interface to communicate with the application gateway. Developers of applications for RPi only need to implement a REST API client to communicate with our service. In the case of DL, we test the performance of the operations required for authentication and authorization. In the case of the network, we examine the transmission overhead introduced by LAAFFI.

We test the performance of LAAFFI on RPi devices by launching a service that receives requests from the client application. Each request is mediated by LAAFFI and is sent in CBOR format using CoAP over a communication channel with the highest bandwidth to the application gateway. For RPi 4 and 3B+, it is Ethernet, while for RPi zero, WH is Wi-Fi. The application gateway returns the same request to the service running on RPi. The service decrypts the message and passes it to the client's application. One cycle is from the moment the client application sends the request to the service until the response is received from the service. We perform the test four times for various amounts of data sent to the service. In each test, we check how long the device needs to perform 50,000 cycles and, based on these data, calculate the number of cycles per second. The results of this evaluation are presented in Table 7.

Table 7. Performance on various Raspberry Pi platforms.

Length (Bytes)	RPi 4	Operations/s RPi 3B+	RPi 0 WH
20	752.60	461.90	76.90
100	726.00	445.74	76.11
1024	373.31	231.82	53.37
15,360	49.46	30.56	6.74
30,720	26.21	16.10	3.42
51,200	16.40	9.79	1.68
102,400	8.30	4.99	-

We carry out performance tests of HLF. We investigate how the number of selected operations per second depends on the number of organizations and their nodes. The selected operations include encrypting and decrypting data, obtaining a key for inter-device communication, and registering IoT devices. We also check how the configuration of the HLF and the hardware configuration of the nodes affect performance. For this purpose, we build an environment consisting of a node with Hyperledger Caliper installed and several HLF nodes. A detailed summary of the basic configuration parameters of the DL nodes and the HLF parameters is presented in Table 8. Hyperledger Caliper [60] is a performance testing tool for HLF, Hyperledger Besu, and Ethereum. The Hyperledger Caliper node generates appropriate requests to the HLF nodes.

Table 8. Distributed Ledger configuration parameters.

Element	Parameter	Value
Node	AWS Instance	c5.xlarge or c5.2xlarge
	CPU	Intel Xeon Platinum 8000, 4 or 8 cores
	Memory (GB)	8 or 16
	Disk (Gib)	50 EBS
	Network (Gbps)	Up to 10
	OS	Ubuntu 20.04.1
	HLF version	1.4.8
	Golang Version	1.15.2
Hyperledger Fabric	Docker Version	1.5–2
	Organizations	From 2 to 4
	Nodes in organization	From 2 to 4
	Endorsement Policy	One node from each organization
	BatchTimeout	2 s
	MaxMessageCount	1000
	AbsoluteMaxBytes	99 MB
	PreferredMaxBytes	5 MB
Orderer consensus	EtdRaft	
Number of orderer nodes	The same as peers	

All tests are performed on the Amazon Web Services (AWS) Elastic Compute Cloud (EC2) platform. Table 9 consists of the results of the performance tests for three types of operations: *basic* operations include the encryption and decryption of messages exchanged between IoT devices and DL; *read* operations are operations that require access to the data stored in DL; and *write* operations require modification of the data stored in DL. Basic and read operations do not require a consensus and are executed by individual DL nodes. Since write operations change the state of HLF, they require consensus. The number of operations per second increases linearly with the performance of the nodes. The results are shown in Table 9. In the case of operations that require reaching a consensus, i.e., transactions that add data about the device and its permissions to the ledger, the results represent the number of operations performed by all organizations per second. The DL nodes need to reach a consensus; the number of operations per second decreases with the number of organizations. As the approval policy requires the acceptance of transactions by any one node in each organization, the number of operations per second increases with the number of nodes in the organization.

The tested latency is acceptable since new data are added only during device registration and authorization. In the tests performed, we focus on verifying the performance of the LAAFFI protocol, so we do not test the writing of data from the devices to the HLF and the exchange of data between devices via the DL. The waiting time to query the data stored in the ledger is comparable to a database query.

Network overhead: We also evaluate the amount of data sent from the client to the server and compare LAAFFI with Datagram Transport Layer Security (DTLS). We perform three tests with different protocols: (1) CoAP + DTLS with client and server certificates for authentication; (2) CoAP + DTLS with preshared key between the client and server; (3) CoAP + CBOR + LAAFFI; and (4) CoAP + CBOR + AES with the shared key. Figure 6 shows the overhead data transferred during our tests. These results indicate that our framework induces less overhead than DTLS.

Table 10 presents the amount of data and the number of packets transferred by comparing our solution (LAAFFI) with DTLS in terms of overhead.

In our experiments, we also collect data on latency. It takes less than 3 s to add a block of up to 800 transactions to HLF.

Table 9. Performance of HLF (C1: 4 cores, 8 GB RAM, C2: 8 cores, 16 GB RAM).

Operation	Msg Length	Operations/s								Type
		2 Orgs				3 Orgs	4 Orgs			
		2 Peers		3 Peers		2 Peers	2 Peers			
	C1	C2	C1	C2	C1	C2	C1			
Encryption	100 B	1649.55	2574.35	2548.85	3297.95	3122.25	5136.15	1430.40	1299.80	Basic
	1 kB	1574.80	2549.40	2304.75	3248.20	3098.75	5190.10	1549.70	1299.30	
	15 kB	1199.80	2396.45	1655.15	2898.30	2310.95	4618.50	1160.25	1024.60	
	30 kB	999.80	2198.95	1574.45	2706.65	2148.35	4237.70	924.60	824.90	
	50 kB	837.15	1863.70	1274.40	2497.55	1699.10	3471.05	774.85	699.90	
100 kB	628.40	1341.35	901.05	1881.25	1154.90	1990.20	577.70	524.90		
Decryption	100 B	1669.70	2102.95	2498.65	3112.50	3148.70	4304.10	1449.60	1299.85	
	1 kB	1574.80	2549.80	2348.75	2946.30	3097.80	4343.55	1403.90	1299.85	
	15 kB	1199.85	2379.60	1847.65	2798.20	2490.85	4143.95	1149.75	999.90	
	30 kB	1074.70	2263.70	1556.45	2598.80	2006.80	4092.70	974.85	874.90	
	50 kB	899.85	1805.50	1362.00	2398.20	1768.85	3641.80	774.90	767.05	
100 kB	649.80	1432.15	934.90	2171.60	1171.65	2778.00	594.90	524.90		
Device-to-device key		1654.60	1974.25	2390.95	3223.30	3297.35	4343.20	1399.75	1299.70	Read
Check permissions		1639.65	2447.00	2498.30	3224.00	3278.85	5184.75	1474.55	1434.60	
Add permission		539.40	784.30	622.55	914.55	646.10	918.05	498.65	443.10	Write
Device registration		500.60	729.45	564.00	793.15	589.20	863.80	464.80	412.70	

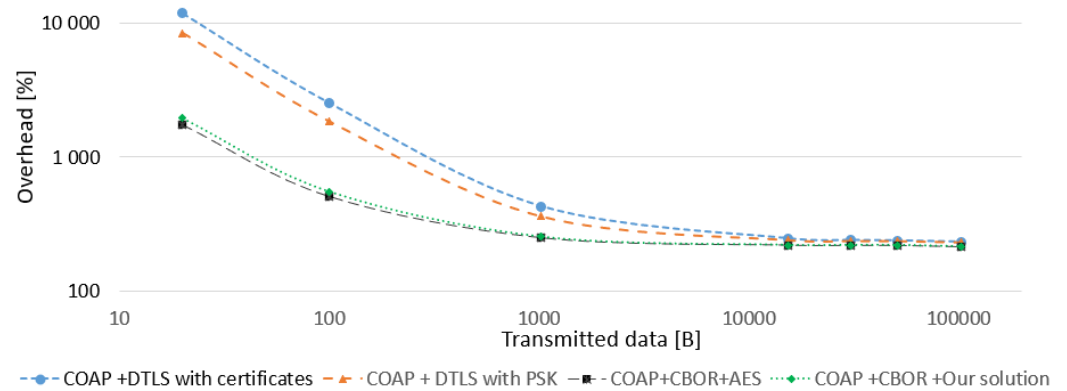


Figure 6. Overhead data transferred during our tests.

Table 10. The number of bytes transferred using various protocols.

No. of Bytes	CoAP + DTLS with Certificate		CoAP + DTLS with PSK		LAAFFI		CoAP + CBOR + AES with PSK	
	Uploaded Data Size	No. of Packets	Uploaded Data Size	No. of Packets	Uploaded Data Size	No. of Packets	Uploaded Data Size	No. of Packets
20	2388		1685		392		349	
100	2550	14	1845	14	552	3	509	3
1024	4399		3695		2594	6	2560	6
15,360	37,888	58	36,864	56	33,792	48	33,850	48
30,720	73,728	102	72,704	101	68,608	93	67,419	93
51,200	121,856	161	120,832	161	114,688	153	112,179	153
102,400	238,592	311	236,544	305	224,256	294	219,389	294

9. Related Work

The research results on the use of Distributed Ledger (DL) for authentication in federated IoT environments are dominated by solutions using DL in support of authentication and authorization based on asymmetric cryptography. At the same time, in our work, we focus on the use of symmetric cryptography, which is much more suitable for constrained IoT devices and provides resilience to quantum attacks.

In LAAFFI, we use DL mainly to enforce authorization, access control, and accountability. This differs from previous work on the use of DLs as a shared database that stores certificates that confirm the identity of IoT [61–64], or [65,66] and as a trust bridge between different Certification Authority (CA) types and IoT devices [67]. The authors have different approaches to storing information in DLs. In the case of [64], they store hashes of device certificates; in ref. [63], there are device certificates; and the authors of ref. [67] store the CA public keys. In the certificateless approaches [65,66], the authors store domain information (e.g., master key), identity information, and public keys for each device in the domains. In the solutions using Identity-based Cryptography (IBC), the authors propose storing domain information, device identities, and, in the case of [62], also a blacklist containing the rejected identities.

An approach that is conceptually more similar to ours but focused on the use of IBC is presented in [61], where DL not only is used as a data store but also executes smart contracts to verify the request and is responsible for creating the keys. In [68], a hybrid approach is proposed, in which authentication between organizations uses asymmetric cryptography, while the use of symmetric cryptography improves organizational performance. In contrast to our solution, instead of a DL, it uses an identity provider with information about all devices belonging to different domains. One of the disadvantages of this approach is that, although an organization does not have access to the private keys of devices, it can access other information about all devices. In addition, each device must be registered with multiple identity providers. Solutions using only symmetric cryptography are less common. None of them uses a DL and relies on the RADIUS service [69] or their own [70] service.

A comparative analysis of LAAFFI and related work is shown in Table 11.

In comparison, LAAFFI is more efficient due to the use of symmetric cryptography and algorithms adapted for operations involving IoT devices. Sending data between IoT devices belonging to different organizations requires only three messages. LAAFFI supports multiple authorization mechanisms, and since it also offers authentication and accountability, it meets the requirements of AAA. It uses DL not only as a database to store authentication data, but, most importantly, it uses smart contracts to ensure that the request complies with the policies agreed between the organizations. Using a DL also ensures accountability for all operations. LAAFFI offers the authentication of devices belonging to a single organization and different organizations, with the assurance that the encryption key will be shared with another device only if the appropriate security policy is met. All LAAFFI components are scalable as confirmed by tests.

The high availability offered by LAAFFI makes it resilient to failure or the disablement of individual components, which is particularly important in tactical and contested environments.

Table 11. Comparison of authentication and authorization methods. CRAM—Challenge–Response Authentication Mechanism, DB—Database, EAP—Extensible Authentication Protocol, IBC—Identity-based Cryptography, Inf—Informal analysis, KM—Key Management, MAC—Message Authentication Code, NA—no data available.

Authentication	Asymmetric								Hybrid	Symmetric			
Crypto	PKI			IBC			Certificateless		MAC + PKI	EAP	CRAM	AES	
Source	[67]	[63]	[64]	[62]	[71]	[61]	[65]	[66]	[68]	[69]	[69]	LAAFFI	
Use case	DB								NA			DB KM AAA	
Authorization	No		NA									Yes	Yes
Scalability	Yes	NA	Yes	Yes		NA							
Lightweight Separation	No								NA				
Decentralization	Yes												
High availability													
Accountability													
Zero-day	NA												
Security Analysis	No	Inf						Tamarin	Scyther	No	AVISPA	Tamarin + Verifpal	

10. Conclusions

We propose a novel Lightweight Authentication and Authorization Framework for Federated IoT (LAAFFI) based on Hyperledger Fabric (HLF). The novelty of the proposed solution consists of registering in the HLF a unique fingerprint of IoT devices according to its available resources and designing a protocol to establish trust and secure communication between IoT devices with the HLF and between IoT devices, even belonging to different organizations that form a federation. Our solution supports devices with various capabilities, including devices with additional hardware resources, such as Physical Unclonable Function (PUF) or Trusted Processing Module (TPM), and simple IoT devices with minimal computing resources, such as Arduino. Our framework is also fully decentralized, and the data stored in the Distributed Ledger (DL) are replicated between nodes, increasing the reliability of the entire solution. All LAAFFI interactions with DL nodes are recorded in HLF, providing accountability in a federated environment. The private data feature of HLF allows keeping the data confidential. However, in this case, only the organization that owns the IoT devices can issue a communication key because only that organization can access the parameter table stored as private data.

We have formally evaluated the security of LAAFFI using Verifpal. We used models developed using this tool to verify the basic security properties of LAAFFI, such as message secrecy, authentication, and freshness. Furthermore, we have implemented a prototype of LAAFFI and evaluated its performance with respect to the volume of transmitted data and the number of operations. The results show that LAAFFI is highly scalable and efficient. It ensures secure data transfer using less data than the commonly used DTLS protocol.

The directions of further work are related to reducing or overcoming the limitations that the developed protocol currently has. Among the most significant are the lack of group communication for IoT devices, the architecture used by Hyperledger Fabric not being quantum safe, and the requirement that IoT devices be equipped with an operating system. Our work opens up additional research questions regarding applications of Distributed Ledger in tactical environments that we plan to address in our future work. For example, the performance of LAAFFI in edge computing scenarios, where individual DL nodes could be connected through tactical radio links, offering constrained wireless communication channels, must be evaluated. Moreover, although LAAFFI, due to its use of only symmetric cryptography, offers resilience to quantum attacks, current implementations of HLF are not

quantum resilient. The performance impact of the transition towards quantum-resilient cryptography needs to be assessed.

Author Contributions: Conceptualization, M.J., K.W. and Z.Z.; methodology, K.W. and Z.Z.; software, M.J.; validation, M.J.; formal analysis, M.J.; investigation, M.J., K.W. and Z.Z.; resources, K.W.; data curation, M.J.; writing—original draft preparation, M.J., K.W. and Z.Z.; writing—review and editing, M.J., K.W. and Z.Z.; visualization, M.J. and K.W.; supervision, K.W. and Z.Z.; project administration, Z.Z.; funding acquisition, K.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The Veripal models used in our work are available at https://gitlab.com/MichalJarosz/laaffi_formalmodels (accessed on 7 September 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

Acronyms

AA	Authentication and Authorization
AAA	Authentication, Authorization and Accounting
ACL	Access Control List
AEAD	Authenticated Encryption with Associated Data
API	Application Programming Interface
AWS	Amazon Web Services
CA	Certification Authority
CBOR	Concise Binary Object Representation
CIMIC	Civil–Military Cooperation
CoAP	Constrained Application Protocol
COSE	CBOR Object Signing and Encryption
CSPRNG	Cryptographically Secure Pseudorandom Number Generator
DL	Distributed Ledger
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
EC2	Elastic Compute Cloud
FMN	Federated Mission Networking
HADR	Humanitarian Assistance and Disaster Relief
HLF	Hyperledger Fabric
HMAC	Hash-based Message Authentication Code
IAM	Identity and Access Management
IBC	Identity-based Cryptography
ICRC	International Committee of the Red Cross
IoT	Internet of Things
KDF	Key Derivation Function
LAAFFI	Lightweight Authentication and Authorization Framework for Federated IoT
MSP	Membership Service Provider
PKI	Public Key Infrastructure
PRF	Pseudo Random Function
PUF	Physical Unclonable Function
RBAC	Role-based Access Control
REST	Representational State Transfer

RPi	Raspberry Pi
SIEM	Security Information and Event Management
TLS	Transport Layer Security
TPM	Trusted Processing Module
UAV	Unmanned Aerial Vehicle
WAF	Web Application Firewall

References

1. Brannsten, M.R.; Johnsen, F.T.; Bloebaum, T.H.; Lund, K. Toward federated mission networking in the tactical domain. *IEEE Commun. Mag.* **2015**, *53*, 52–58. [CrossRef]
2. Manso, M.; Furtak, J.; Guerra, B.; Michaelis, J.; Ota, D.; Suri, N.; Wrona, K. Connecting the Battlespace: C2 and IoT Technical Interoperability in Tactical Federated Environments. In Proceedings of the MILCOM 2022–2022 IEEE Military Communications Conference (MILCOM), Rockville, MD, USA, 28 November–2 December 2022; pp. 1045–1052. [CrossRef]
3. Pal, S.; Hitchens, M.; Rabehaja, T.; Mukhopadhyay, S. Security Requirements for the Internet of Things: A Systematic Approach. *Sensors* **2020**, *20*, 5897. [CrossRef] [PubMed]
4. Plian. Multichain Framework. 2022. Available online: <https://pliangroup.gitbook.io/plian/about-the-blockchain/multichain> (accessed on 15 September 2022).
5. ETSI. Permissioned Distributed Ledgers (PDL) Smart Contracts—System Architecture and Functional Specification. Technical report, ETSI. 2021. Available online: https://www.etsi.org/deliver/etsi_gr/PDL/001_099/004/01.01.01_60/gr_PDL004v010101p.pdf (accessed on 15 September 2022).
6. Sullivan, N.; Wood, C.A. Guidelines for Writing Cryptography Specifications. Internet-Draft Draft-Sullivan-Cryptography-Specification-00, IETF Network Working Group. 2023. Available online: <https://datatracker.ietf.org/doc/draft-sullivan-cryptography-specification/00/> (accessed on 6 November 2023).
7. Chen, L. Recommendation for Key Derivation Using Pseudorandom Functions. SP 800-108r1, NIST. 2022. <https://doi.org/10.6028/NIST.SP.800-108r1> (accessed on 29 September 2024).
8. IBM. IBM Cloud Docs: Internet of Things Platform. Available online: <https://cloud.ibm.com/docs/IoT/index.html> (accessed on 15 September 2022).
9. Koo, J.; Oh, S.R.; Kim, Y.G. Device Identification Interoperability in Heterogeneous IoT Platforms. *Sensors* **2019**, *19*, 1433. [CrossRef]
10. Goichon, F.; Lauradoux, C.; Salagnac, G.; Vuillemin, T. Entropy Transfers in the Linux Random Number Generator. Research Report RR-8060, INRIA. 2012. Available online: <https://inria.hal.science/hal-00738638/document> (accessed on 29 September 2024).
11. Pousse, B. Short Communication: An Interpretation of the Linux Entropy Estimator. Cryptology ePrint Archive, Paper 2012/487. 2012. Available online: <https://eprint.iacr.org/2012/487> (accessed on 29 September 2024).
12. Huhn, T. Myths about /dev/urandom. 2021. Available online: <https://www.2uo.de/myths-about-urandom/#from-linux-48-onward> (accessed on 20 September 2022).
13. Turan, M.S.; Barker, E.B.; Kelsey, J.M.; McKay, K.A.; Baish, M.L.; Boyle, M. Recommendation for the Entropy Sources Used for Random Bit Generation. SP 800-90B, NIST. 2018. <https://doi.org/10.6028/NIST.SP.800-90B> (accessed on 29 September 2024).
14. Shiozaki, M.; Hori, Y.; Fujino, T. Entropy Estimation of Physically Unclonable Functions. *IACR Cryptol. EPrint Arch.* **2020**, *2020*, 1284. Available online: <https://ia.cr/2020/1284> (accessed on 15 September 2022).
15. Maes, R.; Verbauwhede, I. Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions. In *Towards Hardware-Intrinsic Security: Foundations and Practice*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 3–37. [CrossRef]
16. Pehl, M.; Tretschok, T.; Becker, D.; Immler, V. Spatial Context Tree Weighting for Physical Unclonable Functions. In Proceedings of the 2020 European Conference on Circuit Theory and Design (ECCTD), Sofia, Bulgaria, 7–10 September 2020; pp. 1–4. [CrossRef]
17. Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Barker, E.; Leigh, S.; Levenson, M.; Vangel, M.; Banks, D.; Heckert, N.; et al. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. SP 800-22 Rev.1a, NIST. 2010. Available online: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf> (accessed on 29 September 2024).
18. van den Berg, R. Entropy analysis of physical unclonable functions. Master’s Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2012. Available online: <https://research.tue.nl/files/47045775/738925-1.pdf> (accessed on 29 September 2024).
19. Colopy, R.; Chopra, J. SRAM Characteristics as Physical Unclonable Functions. Worcester Polytechnic Institute, Project Number: MQP-BS2-0803. 2009. Available online: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=8c601d1daa6f321aefa57bdb506d7a9ceec062d35> (accessed on 29 September 2024).
20. Abulibdeh, E.; Younes, L.; Mohammad, B.; Humood, K.; Saleh, H.; Al-Qutayri, M. DRAM-Based PUF Utilizing the Variation of Adjacent Cells. *IEEE Trans. Inf. Forensics Secur.* **2024**, *19*, 2909–2918. [CrossRef]

21. Keller, C.; Gürkaynak, F.; Kaeslin, H.; Felber, N. Dynamic memory-based physically unclonable function for the generation of unique identifiers and true random numbers. In Proceedings of the 2014 IEEE International Symposium on Circuits and Systems (ISCAS), Melbourne, Australia, 1–5 June 2014; pp. 2740–2743. [CrossRef]
22. Tehranipoor, F.; Karimian, N.; Yan, W.; Chandy, J.A. DRAM-Based Intrinsic Physically Unclonable Functions for System-Level Security and Authentication. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2017**, *25*, 1085–1097. [CrossRef]
23. Will, N. Development of a Measurement Setup for Large Scale SRAM PUF Evaluation. Ruhr-Universität Bochum. 2017. Available online: https://informatik.rub.de/wp-content/uploads/2021/11/BA_Will.pdf (accessed on 29 September 2024).
24. Linux Certif Documentation of ENT—Pseudorandom Number Sequence Test. Available online: <http://www.linuxcertif.com/man/1/ent/> (accessed on 29 September 2024).
25. Chaplin, C.; Qi, E.; Ptasinski, H.; Walker, J.; Li, S. 802.11i Overview. Technical Report 802.11i, IEEE. 2005. Available online: https://ieee802.org/16/liaison/docs/80211-05_0123r1.pdf (accessed on 29 September 2024).
26. Bellare, M. New Proofs for NMAC and HMAC: Security without Collision Resistance. *J. Cryptol.* **2015**, *28*, 844–878. [CrossRef]
27. Maes, R. Physically Unclonable Functions: Constructions, Properties and Applications. Ph.D. Thesis, Katholieke Universiteit Leuven—Faculty of Engineering, Leuven, Belgium, 2012. Available online: <https://www.esat.kuleuven.be/cosic/publications/thesis-211.pdf> (accessed on 29 September 2024).
28. Feiten, L.; Sauer, M.; Becker, B. On Metrics to Quantify the Inter-Device Uniqueness of PUFs. Cryptology ePrint Archive, Paper 2016/320. 2016. Available online: <https://eprint.iacr.org/2016/320> (accessed on 15 September 2022).
29. Kobeissi, N. Verifpal User Manual. Manual, Symbolic Software. 2021. Available online: <https://verifpal.com/res/pdf/manual.pdf> (accessed on 15 September 2022).
30. Jarosz, M. Verifpal Model of LAAFFI. Manual, Symbolic Software. 2021. Available online: https://gitlab.com/MichalJarosz/laaffi_formalmodels (accessed on 15 September 2022).
31. Kobeissi, N.; Nicolas, G.; Tiwari, M. Verifpal: Cryptographic Protocol Analysis for the Real World. Cryptology ePrint Archive, Paper 2019/971 2019. Available online: <https://eprint.iacr.org/2019/971> (accessed on 15 September 2022).
32. Hassija, V.; Chamola, V.; Saxena, V.; Jain, D.; Goyal, P.; Sikdar, B. A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures. *IEEE Access* **2019**, *7*, 82721–82743. [CrossRef]
33. Mouaatamid, O.E.; Lahmer, M.; Belkasm, M. Internet of Things Security: Layered classification of attacks and possible Countermeasures. *Electron. J. Inform. Technol.* **2016**. Available online: <https://api.semanticscholar.org/CorpusID:54878156> (accessed on 15 September 2022).
34. Saqib, M.; Moon, A.H. A Systematic Security Assessment and Review of Internet of Things in the Context of Authentication. *Comput. Secur.* **2023**, *125*, 103053. [CrossRef]
35. Sengupta, J.; Ruj, S.; Das Bit, S. A Comprehensive Survey on Attacks, Security Issues and Blockchain Solutions for IoT and IIoT. *J. Netw. Comput. Appl.* **2020**, *149*, 102481. [CrossRef]
36. Stoyanova, M.; Nikoloudakis, Y.; Panagiotakis, S.; Pallis, E.; Markakis, E.K. A Survey on the Internet of Things (IoT) Forensics: Challenges, Approaches, and Open Issues. *IEEE Commun. Surv. Tutorials* **2020**, *22*, 1191–1221. [CrossRef]
37. Shaw, G. *Penetration Testing Technical Report: Hyperledger Fabric v1.4 and v2.0*; Technical Report; Nettitude: Singapore, 2019.
38. Vishwakarma, R.; Jain, A.K. A survey of DDoS attacking techniques and defence mechanisms in the IoT network. *Telecommun. Syst.* **2020**, *73*, 3–25. [CrossRef]
39. Munshi, A.; Alqarni, N.A.; Abdullah Almalki, N. DDOS Attack on IOT Devices. In Proceedings of the 2020 3rd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 19–21 March 2020; pp. 1–5. [CrossRef]
40. Khan, F.; Al-Atawi, A.A.; Alomari, A.; Alsirhani, A.; Alshahrani, M.M.; Khan, J.; Lee, Y. Development of a Model for Spoofing Attacks in Internet of Things. *Mathematics* **2022**, *10*, 3686. [CrossRef]
41. Dwivedi, A.D.; Srivastava, G. Security analysis of lightweight IoT encryption algorithms: SIMON and SIMECK. *Internet Things* **2023**, *21*, 100677. [CrossRef]
42. CVEdetails.com. Golang: Security Vulnerabilities, CVEs. Available online: https://www.cvedetails.com/vulnerability-list/vendor_id-14185/Golang.html (accessed on 29 September 2024).
43. CVEdetails.com. Hyperledger Fabric: Security Vulnerabilities, CVEs Published in 2022. Available online: https://www.cvedetails.com/vulnerability-list/vendor_id-18415/product_id-117539/year-2022/Hyperledger-Fabric.html (accessed on 29 September 2024).
44. Sasi, T.; Lashkari, A.H.; Lu, R.; Xiong, P.; Iqbal, S. A comprehensive survey on IoT attacks: Taxonomy, detection mechanisms and challenges. *J. Inf. Intell.* **2023**. [CrossRef]
45. Zhang, K.; Liang, X.; Lu, R.; Shen, X. Sybil Attacks and Their Defenses in the Internet of Things. *IEEE Internet Things J.* **2014**, *1*, 372–383. [CrossRef]
46. Devi, M.; Majumder, A. Side-Channel Attack in Internet of Things: A Survey. In *Proceedings of the Applications of Internet of Things*; Mandal, J.K., Mukhopadhyay, S., Roy, A., Eds.; Springer: Singapore, 2021; pp. 213–222. [CrossRef]
47. Grassi, P.; Garcia, M.; Fenton, J. Digital Identity Guidelines. SP 800-63-3, NIST. 2017. Available online: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-3.pdf> (accessed on 29 September 2024).

48. Kala, S.; Nalesh, S. Chapter 24—Security and challenges in IoT-enabled systems. In *System Assurances*; Johri, P., Anand, A., Vain, J., Singh, J., Quasim, M., Eds.; Emerging Methodologies and Applications in Modelling; Academic Press: Cambridge, MA, USA, 2022; pp. 437–445. [\[CrossRef\]](#)
49. Liao, C.H.; Shuai, H.H.; Wang, L.C. Eavesdropping prevention for heterogeneous Internet of Things systems. In Proceedings of the 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 12–15 January 2018; pp. 1–2. [\[CrossRef\]](#)
50. Feng, Y.; Wang, W.; Weng, Y.; Zhang, H. A Replay-Attack Resistant Authentication Scheme for the Internet of Things. In Proceedings of the 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Guangzhou, China, 21–24 July 2017; Volume 1, pp. 541–547. [\[CrossRef\]](#)
51. Conti, M.; Dragoni, N.; Lesyk, V. A Survey of Man In The Middle Attacks. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 2027–2051. [\[CrossRef\]](#)
52. Andola, N.; Raghav; Gogoi, M.; Venkatesan, S.; Verma, S. Vulnerabilities on Hyperledger Fabric. *Pervasive Mob. Comput.* **2019**, *59*, 101050. [\[CrossRef\]](#)
53. Li, P.; Li, S.; Ding, M.; Yu, J.; Zhang, H.; Zhou, X.; Li, J. A Vulnerability Detection Framework for Hyperledger Fabric Smart Contracts Based on Dynamic and Static Analysis. In Proceedings of the EASE'22: 26th International Conference on Evaluation and Assessment in Software Engineering, New York, NY, USA, Gothenburg, Sweden, 13–15 June 2022; pp. 366–374. [\[CrossRef\]](#)
54. Shah, N.K.; Nandurkar, S.; Bilapate, M.; Maalik, M.A.; Harne, N.; Shaik, K.; Kumar, A. Smart Contract Vulnerability Detection Techniques for Hyperledger Fabric. In Proceedings of the 2023 IEEE 8th International Conference for Convergence in Technology (I2CT), Lonavla, India, 7–9 April 2023; pp. 1–7. [\[CrossRef\]](#)
55. Ding, M.; Li, P.; Li, S.; Zhang, H. HFContractFuzzer: Fuzzing Hyperledger Fabric Smart Contracts for Vulnerability Detection. In Proceedings of the Evaluation and Assessment in Software Engineering, Trondheim, Norway, 21–23 June 2021. [\[CrossRef\]](#)
56. Putz, B.; Böhm, F.; Pernul, G. HyperSec: Visual Analytics for blockchain security monitoring. *arXiv* **2021**, arXiv:2103.14414. [\[CrossRef\]](#)
57. Shelby, Z.; Hartke, K.; Bormann, C. The Constrained Application Protocol (CoAP). RFC 7252, IETF. 2014. Available online: <https://www.rfc-editor.org/info/rfc7252> (accessed on 29 September 2024).
58. Bormann, C.; Hoffman, P. Concise Binary Object Representation (CBOR). RFC 7049, IETF. 2013. Available online: <https://www.rfc-editor.org/info/rfc7049> (accessed on 29 September 2024).
59. Schaad, J. CBOR Object Signing and Encryption (COSE). RFC 8152, IETF. 2017. Available online: <https://www.rfc-editor.org/info/rfc8152> (accessed on 29 September 2024).
60. Hyperledger. Hyperledger Caliper Documentation, 2022. Available online: <https://hyperledger.github.io/caliper/v0.6.0/getting-started/> (accessed on 15 September 2022).
61. Cheng, G.; Chen, Y.; Deng, S.; Gao, H.; Yin, J. A Blockchain-Based Mutual Authentication Scheme for Collaborative Edge Computing. *IEEE Trans. Comput. Soc. Syst.* **2022**, *9*, 146–158. [\[CrossRef\]](#)
62. Tong, F.; Chen, X.; Wang, K.; Zhang, Y. CCAP: A Complete Cross-Domain Authentication Based on Blockchain for Internet of Things. *IEEE Trans. Inf. Forensics Secur.* **2022**, *17*, 3789–3800. [\[CrossRef\]](#)
63. Xiao, X.; Guo, F.; Hecker, A. A Lightweight Cross-Domain Proximity-Based Authentication Method for IoT Based on IOTA. In Proceedings of the 2020 IEEE Globecom Workshops (GC Wkshps), Taipei, Taiwan, 7–11 December 2020; pp. 1–6. [\[CrossRef\]](#)
64. Zhang, Y.; Luo, Y.; Chen, X.; Tong, F.; Xu, Y.; Tao, J.; Cheng, G. A Lightweight Authentication Scheme Based on Consortium Blockchain for Cross-Domain IoT. *Sec. Comm. Netw.* **2022**, *2022*, 9686049. [\[CrossRef\]](#)
65. Li, W.; Zhang, S.; Chen, Z.; Sen, L. Cross-Domain Authentication Scheme for IoT Devices Based on Blockchain. In Proceedings of the 2022 IEEE 13th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 21–23 October 2022; pp. 67–73. [\[CrossRef\]](#)
66. Wang, X.; Gu, C.; Wei, F.; Lu, S.; Li, Z. A Certificateless-Based Authentication and Key Agreement Scheme for IIoT Cross-Domain. *Secur. Commun. Netw.* **2022**, *2022*, 3693748. [\[CrossRef\]](#)
67. Yang, Y.; Wei, L.; Wu, J.; Long, C.; Li, B. A Blockchain-Based Multidomain Authentication Scheme for Conditional Privacy Preserving in Vehicular Ad-Hoc Network. *IEEE Internet Things J.* **2022**, *9*, 8078–8090. [\[CrossRef\]](#)
68. Santos, M.L.; Carneiro, J.C.; Franco, A.M.; Teixeira, F.A.; Henriques, M.A.; Oliveira, L.B. FLAT: Federated lightweight authentication for the Internet of Things. *Ad. Hoc. Netw.* **2020**, *107*, 102253. [\[CrossRef\]](#)
69. Skarmeta, A.; Garcia Carrillo, D.; Olivereau, A. End-Node Security. In *Internet of Things Security and Data Protection*; Ziegler, S., Ed.; Springer International Publishing: Cham, Switzerland, 2019; pp. 45–69. [\[CrossRef\]](#)
70. Alshahrani, M.; Traore, I. Secure mutual authentication and automated access control for IoT smart home using cumulative Keyed-hash chain. *J. Inf. Sec. App.* **2019**, *45*, 156–175. [\[CrossRef\]](#)
71. Shen, M.; Liu, H.; Zhu, L.; Xu, K.; Yu, H.; Du, X.; Guizani, M. Blockchain-Assisted Secure Device Authentication for Cross-Domain Industrial IoT. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 942–954. [\[CrossRef\]](#)

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.