Secure and Lightweight Authentication Schemes for Internet of Things (IoT)

by

Mohammed M. Alshahrani
B.Sc. of Computer Science, King Khalid University, Saudi Arabia, 2009
MEng in Internetworking, Dalhousie University, Canada, 2013

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

Secure and Lightweight Authentication Schemes for Internet of Things (IoT)

by

Mohammed M. Alshahrani
B.Sc. of Computer Science, King Khalid University, Saudi Arabia, 2009
MEng in Internetworking, Dalhousie University, Canada, 2013

Supervisory Committee

Prof. Issa Traoré, Supervisor
(Department of Electrical and Computer Engineering)

Prof. Fayez Gebali, Department Member
(Department of Electrical and Computer Engineering)

Prof. Hausi A. Müller, Outside Member
(Department of Computer Science)

Supervisory Committee

Prof. Issa Traoré, Supervisor
(Department of Electrical and Computer Engineering)

Prof. Fayez Gebali, Department Member
(Department of Electrical and Computer Engineering)

Prof. Hausi A. Müller, Outside Member
(Department of Computer Science)

ABSTRACT

IoT platforms face huge challenges in deploying robust authentication mechanisms due to the fact that edge devices and resource-constrained devices may not have enough compute and storage capabilities to deploy and run existing mechanisms, which involve in general complex computations. Moreover, establishing end-to-end device authentication in the Internet of Things (IoT) networks is challenging because of the heterogeneous nature of IoT devices. One of the well-known challenges confronting the IoT infrastructure is related to authentication. Many IoT devices rely on weak authentication schemes, which has led in the last few years to several successful and widely publicized hacking incidents [26]. According to the ISO/IEC 27002 standard, authentication is the process of determining whether something is, in fact, what it is declared to be [12]. Authentication is considered the main gate to protect IoT networks from various security threats; determining who the entity is (authentication) is of high importance to establish a secure session between IoT devices. This dissertation identifies gaps in the literature and presents new authentication schemes and security mechanisms to improve IoT security and privacy against common attacks such as replay and impersonation. This research enhances IoT security and privacy by introducing a new lightweight mutual authentication and key exchange protocol for IoT based on dynamic identity and cumulative chained-hash. Nodes can anonymously and mutually authenticate and establish a session with the controller node using dynamic identities and temporary

symmetric keys in an unlinkable and untraceable manner. Moreover, the enforcement of security policies between nodes is guaranteed by setting up virtual domain segregation and restricting node capabilities of sending and receiving data to or from other nodes. The Cumulative chained-hash technique is introduced as a way to ensure the identity of the sender (through challenge-response). Additionally, we introduce a new anonymous device-to-device mutual authentication and key exchange protocol based on the ZigBee technique. The proposed protocol relies on symmetric encryption and counter and enables IoT devices to authenticate in the network and agree on a shared secret session key when communicating with each other via a trusted intermediary (home controller). To achieve forward secrecy, the session keys are changed frequently after every communication session. The proposed scheme achieves secure, anonymous authentication with the unlinkability and untraceability of IoT device transactions.

The security of the protocols is evaluated and simulated using three different methods: informal analysis, formal analysis using the Burrows–Abadi–Needham logic (BAN), and model-checking using the automated validation of Internet security protocols and applications (AVISPA) toolkit. The overhead and efficiency of the proposed schemes are analyzed and compared with other related schemes. The results showed that our protocols are in general more efficient.

Table of Contents

List of Abbreviations

AES	Advanced Encryption Standard
AVISPA	Automated Validation of Internet Security Protocols and Applications
BAN	Burrows–Abadi–Needham
CBC-MAC	Cipher Block Chaining Message Authentication Code
CFAR	Constant False Alarm Rate
CLAtSe	Constraint-Logic-based Attack Searcher
CTR	Counter
CCM	Counter with CBC-MAC
DTLS	Datagram Transport Layer Security
DoS	Denial-of-Service
D2D	Device-to-Device
ECC	Elliptic Curve Cryptography
EAP	Extensible Authentication Protocol
GBA	Group-based Authentication
HLPSL	High-Level Protocol Specification Language
ID	Identification
IF	Intermediate Format
IoT	Internet of Things

IP	Internet Protocol
HLPSL	Keyed-hash Message Authentication Code
M2M	Machine-to-Machine
MTCD	Machine-type Communication Device
MAC	Message Authentication Code
MFA	Multi-Factor Authentication
OFMC	On-the-fly Model-Checker
OTP	One-Time Password
PDA	Personal Digital Assistant
PRNG	Pseudorandom Number Generator
RFID	Radio-Frequency Identification
SATMC	SAT-based Model-Checker
SPAN	Security Protocol Animator
TA4SP	Tree Automata based on Automatic Approximations for the Analysis of Security Protocols
TRNG	True Random Number Generator
VSIoTN	Virtual Segregation of IoT Networ
XOR	Exclusive OR

List of Tables

List of Figures

ACKNOWLEGMENTS

"In the name of Allah, Most Gracious, Most Merciful"

Alhamdulillah, all praises to Allah for the strengths and his blessing in completing this dissertation. No word can express my sincere thanks to my supervisor Prof. Issa Traoré, for his supervision and constant support. Without his continuous support, guidance, and encouragement, this thesis might have not been successfully completed. My acknowledgement is also due to Prof. Fayez Gebali and Prof. Hausi A. Müller for serving as my committee members and their cooperation, comments and constructive criticism. I would also like to thank you for your brilliant comments and suggestions and for making my research enjoyable.

This research is supported by the University of Bisha and the Ministry of Education of the Kingdom of Saudi Arabia.

DEDICATION

To my parents, **Mujib Alshahrani** and **Thanwa Alshahrani** for their prayers and encouragement.

To my lovely wife, **Maram Alshahrani** for always standing by me, and believing in me.

To **my homeland**.

Chapter 1

Introduction

1.1 Context

The term Internet of Things (IoT) was first coined by Kevin Ashton in 1999 in the context of supply chain management [5]. The most recent IoT platforms were designed based on a diverse mixture of readapting existing components, solutions, protocols, and platforms for different purposes in terms of their development and enhancement. IoT devices are often resource-constrained and deployed in unmonitored, physically unsecured environments. However, although there is an urgent need to secure IoT infrastructures, this necessity is confronted with the aforementioned resource limitations of the infrastructure underlying platforms and devices. One of the essential aspects of securing an IoT infrastructure is the device identity and the mechanisms to authenticate it. Authentication is one of the Achilles' heels of the current IoT infrastructure. As a matter of fact, many IoT devices have weak passwords or are still using manufacturer-issued default passwords, which make them vulnerable to botnets (e.g., the Mirai IoT botnet) [26] and exploit kits specially designed to target IoT networks. At the same time, hackers can connect rogue devices to

IoT networks using fake or multiple identities without being caught. The enactment of the above threats is made possible because the landscape of IoT authentication is still in its infancy [33]. Furthermore, existing authentication mechanisms involve heavy computations that cannot be afforded by IoT devices, which, as mentioned earlier, are resource-constrained. Additionally, these authentication mechanisms also require a degree of user intervention in terms of configuration and provisioning. Furthermore, many IoT devices have limited access, thus requiring the initial configuration to be protected from tampering, theft and other forms of compromise throughout the device's usable life, which, in many cases, could be years. One approach to address the computational and resource limitations of IoT platforms is to offload computational-intensive tasks to the cloud. However, despite the growing popularity of cloud computing, many organizations that maintain sensitive information are wary of fully offloading sensitive computations and resources to third-party cloud providers, which, in some cases, are located in different jurisdictions and might be subjected to multiple countries' regulations, where there may be opaque security practices. Although cloud computing platforms have powerful resources to perform the tasks that IoT devices cannot, they are frequently too far away to process the data and respond at the right time. Recently, fog computing has appeared as a "middle of the road" solution to help mitigate the above concerns about cloud computing [24].

1.2 Research Problem

Despite the growing interest in IoT products and services and the advancements in its underlying technology, IoT devices and networks are exposed to a wide variety of security threats, some of which are well-known and part of the existing attack arsenal against

conventional systems, while others involve novel attack vectors that are specific to the IoT technology and remain unknown until they are detected in the network [2, 28, 34, 64].

One of the principal aspects in making an IoT infrastructure secure is the underlying IoT node authentication mechanism. The resource-constrained IoT nodes cannot afford the current authentication mechanisms due to the complex computations involved in these mechanisms. Moreover, the existing authentication mechanisms involve a high level of user intervention in configuring and controlling the devices. Unlike conventional network devices, IoT nodes, which are usually deployed in unmonitored and unsecured environments, have limited access; hence, their initial configurations need to be protected against any kinds of external or insider threats or compromises during the lifetime of the IoT node. Additionally, due to the fact that most IoT devices are weak and resource-constrained, the cryptographic techniques used can easily be exploited and broken. Hence, compromising one node could lead to the compromise of the entire IoT network.

Moreover, authentication necessitates identification through a unique identifier. As time goes by, attackers can trace transactions linked to the same identity, thus leading to privacy breach. In an IoT domain such as smart home, transaction's traceability may result in tracing the lifestyle of the household or deducing from the data, sensitive information, such as health and credit history, living arrangements, and so on [44]. Hence, anonymity, unlinkability, and untraceability [43] are important key properties when designing and operating authentication mechanisms for IoT networks, as these prevent an adversary from obtaining the real identity of an IoT end device and from linking any given session to any other session of the same device.

Consequently, there is a necessity to introduce new mechanisms to authenticate and control IoT nodes in a secure way that must be compatible with the environmental and

engineering limitations underlying the IoT ecosystem. In this dissertation, we present different protocols to authenticate IoT nodes while addressing security and privacy, and performance considerations.

1.3 Approach

To address the aforementioned security challenges, we propose a new secure mutual authentication and key exchange scheme for the IoT based on transient or dynamic identities.

The dynamic identity of IoT nodes (DIdoT) is a new concept for uniquely identifying IoT nodes. The DIdoT is constructed from fixed and variable components, and evolving time-dependent component. The fixed component is created from fixed parameters of the IoT node, which is the node ID. The variable component is created from a random generator. The time-dependent component is created from the time stamp. These different components are hashed together using SHA-3 as a hashing algorithm, which provides a tunable parameter allowing a tradeoff between security and performance. The real identity is kept secret and never transmitted by the IoT node. This guarantees uniqueness of identities of the IoT nodes and mitigates the possibilities of identities theft attacks such as impersonation and sybil attacks. Another new concept is temporal keys (TKs) that change every session. The TKs are constructed from fixed and variable components. The fixed component is created from a fixed parameter of the IoT node, which is the node ID. The variable component is created from a random number generator. These two different components are hashed together using SHA-3, and as aforementioned, the hashing algorithm provides a tradeoff between security and performance. This continuously variable nature of the identity significantly limits the impact of brute-force attacks by limiting the session key lifetime.

According to a survey carried out by CA Technologies on the state of insider threat in 2018 [53], 90% of surveyed organizations felt that they were vulnerable to insider attacks and 53% of organizations pointed out that they have been the target of insider attacks during the year. Although most organizations focus on how to defend against external attacks, they sometimes put considerably less efforts into defending against internal attackers or rather ignore them. As the IoT devices are ubiquitous and unattended, internal attackers will play a big role in cyber security threats and may cause serious issues. Moreover, due to the fact that most IoT devices are weak and resource-constrained, the cryptography techniques can easily be exploited and broken. Hence, compromising one node could make it easy to take control of the other IoT devices; thus there is a need for automated access control that can prevent such compromise. Therefore, we will try to push the access control mechanisms to the IoT end devices, and we will make it impossible or more difficult to change them after being configured by the network administrator. This will help mitigate insider threats. This is made possible through using virtual segregation of IoT network (VSIoTN) and defining the IoT devices'capabilities. VSIoTN allows the IoT network administrators to virtually partition their networks based on the scope of interaction between these devices. The IoT devices are placed on different virtual domains based on their interactions with each other. This will help the Controller to manage and control the communication flows in the IoT smart home network. Moreover, each IoT device will be assigned an access capability according to its interaction with other IoT devices. Hence, not only are the IoT devices restricted by the virtual domain, but also by their capabilities.

Based on the above general framework, we propose two different authentication schemes. At the core of the first proposed authentication scheme is a lightweight challenge-response that relies on a cumulative chained hash. Each node maintains a database containing

hashes generated during previous authentication sessions. The authentication challenge will consist of requesting proof of knowledge of some previous entries from the database. Each authentication attempt will be based on a temporary secret key TK_N generated and sent by the node N being authenticated and shared between N and the controller CRN.

Securing IoT identities is the starting point of building the trust in IoT automated communication [49]. By securing IoT identities, we can maintain the CIA triad: confidentiality, integrity, and availability. If an attacker succeeds in impersonating an identity on an IoT node, all security measures, such as authentication and access protection, make no sense. We will improve and ensure the identity assurance by introducing a modified fog architecture to support and ensure high assurance in the identity of IoT nodes. This modified architecture helps to prevent identity theft such as spoofing and masquerading attacks.

The IoT integrates several wireless technologies to connect IoT devices. ZigBee is a popular wireless technology for that purpose, which also enables some fundamental security services, such as authentication and encryption. We propose a second authentication scheme that somehow imitates the Zigbee protocol ZigBee in the sense that each network involves a central device (Zigbee coordinator), which starts and maintains the communications between network nodes (Zigbee nodes). Our scheme also employs AES-based symmetric encryption between nodes similar to ZigBee, where only the controller that takes possession of the symmetric key of a node can encrypt the messages.

The security analysis, simulation, and implementation of the proposed authentication schemes show that our proposed protocols are efficient and resilient against known attack methods.

1.4 Contributions

In this thesis, we make two key contributions as follows:

1. A lightweight and anonymous mutual authentication and key exchange protocol based on dynamic identity and temporary session keys that change every session.

2. An anonymous device-to-device mutual authentication and key exchange scheme based on the ZigBee technique, designed for a smart home network, an important domain in the IoT.

1.4.1 Linkage of Scientific Papers

The proposed lightweight and anonymous mutual authentication and key exchange framework based on dynamic identity and temporary session keys (i.e., Contribution 1) was published in 1, 4, and 5 (listed in section 1.4.2). Paper 1 (Secure mutual authentication and automated access control for IoT smart home using cumulative keyed-hash chain) outlines a framework and its formal and informal security analyses in addition to the efficiency evaluation. Paper 4 (Design and Implementation of a Lightweight Authentication Framework for the Internet of Things (IoT)) illustrates the implementation of the proposed framework.

Both Papers 1 and 5 (Lightweight IoT Mutual Authentication Scheme based on Transient Identities and Transactions History) investigate the enforcement of security policies based on different methods, such as the division of devices using virtual domain segregation in an IoT smart home network, the concept of cumulative chained-hash, and a new architecture based on fog computing, which provides and support the identity assurance of IoT devices.

The proposed anonymous device-to-device mutual authentication and key exchange scheme based on the ZigBee technique (i.e., Contribution 2) was published in Paper 2

(Anonymous mutual IoT interdevice authentication and key agreement scheme based on the ZigBee technique) and Paper 6 (Anonymous IoT Mutual Inter-device Authentication Scheme based on Incremental Counter).

1.4.2 List of publications

[1] Alshahrani, M., Traore, I., & Woungang, I. (2019). Anonymous mutual IoT interdevice authentication and key agreement scheme based on the ZigBee technique. *Journal of Internet of Things*, Elsevier, Volume 7, September 2019.

[2] Alshahrani, M., & Traore, I. (2019). Secure mutual authentication and automated access control for IoT smart home using cumulative keyed-hash chain. *Journal of Information Security and Applications*, Elsevier, Volume 45, April 2019, Pages 156-175.

[3] Traore, I., Alshahrani, M., & Obaidat, M. S. (2018). State of the art and perspectives on traditional and emerging biometrics: A survey. *Journal of Security and Privacy*, Wiley, Vol. 1 Issue 6, November/December 2018.

[4] Alshahrani, M., Traore, I., & Woungang, I. (2019). Design and Implementation of a Lightweight Authentication Framework for the Internet of Things (IoT). *The 6th IEEE International Conference on Internet of Things: Systems, Management and Security (IOTSMS 2019)*, Granada, Spain, October 22-25 2019.

[5] Alshahrani, M., Traore, I., & Saad, S. (2019). Lightweight IoT Mutual Authentication Scheme based on Transient Identities and Transactions History. *The 12th International Symposium on Foundations Practice of Security (FPS2019)*, Toulouse, France, November 5-7 2019.

[6] Alshahrani, M., Traore, I., Woungang, I. (2019). Anonymous IoT Mutual InterdeviceAuthentication Scheme based on Incremental Counter. *The 7th International Conference on Future Internet of Things and Cloud (FiCloud 2019)*, Istanbul, Turkey, August 26-28 2019.

[7] Alshahrani, M. (2018). Emerging Biometrics Technologies. *In International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments*. Springer, Cham, 2017.

1.5 Dissertation Outline

The remainder of this dissertation is organized as follows.

Chapter 2 provides an overview of the literature underlying this research. It provides a quick introduction to the Internet of Things and authentication. Also, this chapter provides an outline of the related security methods and tools used in this research.

Chapter 3 summarizes and discusses related work on authentication and access control for the IoT.

Chapter 4 describes our first proposed authentication scheme and introduces the enforcement of the security policy. We also present the cumulative chained-hash mechanism and the new architecture based on fog computing.

Chapter 5 presents the second proposed authentication framework. The framework achieves different security properties, anonymity, unlinkability, and conditional traceability, in addition to the important dual properties of confidentiality and integrity.

Chapter 6 concludes the thesis by discussing the contributions of the research and outlining future work.

Chapter 2

Background

In this chapter, we provide background information on the Internet of Things and authentication, and also describe the Fog computing paradigm. Furthermore, we give an overview of the validation and evaluation methods and tools used in our work.

2.1 Internet of Things and Authentication

The Internet of Things (IoT) is one of the most recent emerging and advanced computing paradigms set forth in the 21st century, which connects both living and non-living things with non-living things into ecosystems. The IoT refers to everyday objects that can sense the environment around them and communicate that data to other objects and services via the Internet without any intervention from human or living bodies.

The IoT integrates several existing technologies, such as wireless sensor networks (WSN), which appeared since the 1980s. WSN technology is an essential component of IoT because it is composed of a collection of sensor nodes connected wirelessly to one another, which provide digital interfaces to the real-world things [37]. However, WSN is

different from IoT in a number of respects. One of the important differences is that WSN is comprised of huge number of connected sensors' nodes that are capable of performing sensing and data collection while IoT system consists of a large number of interconnected objects, things, sensors, devices, etc., which are able to provide value-added services such as location and analytic via utilizing intelligent data processing and management for several applications. Another important difference is that in the IoT infrastructure, it is required to provide the nodes with Internet connectivity, whereas, Internet connectivity is not required in WSN.

Furthermore, many IoT devices do not have the required compute power, memory, or storage to support the current authentication protocols, which rely on computationally intensive cryptographic algorithms, e.g., AES and RSA [1].

Authentication requires identification by means of a unique identifier, and over time transactions associated with the same identity can be traceable, leading to privacy breach. In an IoT environment such as smart home, transactions traceability can be used to track the lifestyle of the household or infer from the data, critical information such as health and credit history, living arrangements and patterns, and so on [47]. Anonymity and unlinkability are privacy-preserving mechanisms, which make user transactions traceability much harder.

Although a number of authentication schemes have been proposed for IoT, to our knowledge, none of these contributions has considered full anonymity of the IoT sensor nodes during the authentication or access control processes.

2.2 Fog Computing

Fog computing technology, also known as fog networking or fogging, was first introduced by Cisco in 2014 as an extension of cloud computing that would work on the edge of the end users'network [7, 8]. Fog computing is a decentralized computing infrastructure where data, compute, storage and applications are distributed between end-users and the cloud network. Fog computing basically extends cloud computing's services to the edge of the network, bringing the services of the cloud closer to where data are generated. The primary goal of fog computing is to enhance efficiency and reduce the size of the data transmitted to the cloud for storage, processing, and analysis. Fog computing applications comprise smart grids, smart cities, and smart buildings, to name a few. In fog computing, the intelligence resides in the local area network. Processing the data occurs in a local smart router or in a gateway, thus minimizing the volume of data transmitted to the cloud. Fog computing performs short-term analytics at the edge while the cloud performs long-term analytics.

2.3 Burrows–Abadi–Needham Logic

Burrows *et al.* [9] introduced the Burrows–Abadi–Needham (BAN) logic, which is used to describe and analyze the authentication protocols. The BAN logic has widely been used for the formal verification of security protocols and to provide the proof of correctness of any authentication protocol [61]. Hence, we capitalize on the widely-accepted BAN logic to prove that our authentication scheme provides secure mutual authentication between an IoT node N and the controller CRN. In this subsection, we present a summarized introduction about the essential symbols and rules of BAN logic.

2.3.1 A brief introduction of important symbols and rules of BAN

Let C (Client) and S (Server) be participators and let X and Y denote a parameter, a formula or an expression. We define the following notations:

- C | ≡ X: C believes the statement X.
- # (X): X is fresh.
- C | $\Longrightarrow$ X: C has jurisdiction over the statement X.
- C $\triangleleft$ X: C sees the statement X.
- C | $\sim$ X: C once said the statement X.
- (X, Y): X or Y is one part of the formula (X, Y).
- $\langle X \rangle_Y$: X combined with Y.
- C $\overset{K}{\leftrightarrow}$ S: K is a secret parameter shared (or to be shared) between C and S.
- C $\overset{X}{\rightleftharpoons}$ S: X is a secret known only to C and S, and possibly to parties trusted by them.

2.3.2 A brief introduction of BAN logic's rules

The following commonly used BAN logic rules are utilized to prove that the authentication scheme ensures secure mutual authentication and key agreement:

- Message-meaning rule: If C sees X encrypted with Y and if C believes Y is a secret key shared with S, then C believes S once said X.

$$\frac{C| \equiv C \overset{Y}{\leftrightarrow} S, C \triangleleft \langle X \rangle_Y}{C| \equiv S| \sim X}$$

- Nonce-verification rule: If C believes X is fresh and C believes S once said X, then C believes S believes X.

$$\frac{C| \equiv \#(X), C| \equiv S| \sim X}{C| \equiv S| \equiv X}$$

- Jurisdiction rule: If C believes S has jurisdiction over X and C believes S believes X, then C believes X.

$$\frac{C| \equiv S| \Longrightarrow X, C| \equiv S| \equiv X}{C| \equiv X}$$

- Freshness-conjuncatenation rule: If one part of a formula is fresh, then the entire formula must also be fresh, so if C believes X is fresh, then C believes X and Y are fresh.

$$\frac{C| \equiv \#(X)}{C| \equiv \#(X, Y)}$$

- Belief rule: If C believes X and Y, then C believes X.

$$\frac{C| \equiv (X, Y)}{C| \equiv X}$$

- Observation rule: If C sees X and Y, then C sees X.

$$\frac{C| \triangleleft (X, Y)}{C| \triangleleft X}$$

2.4 Attacker Model

A better understanding of threats helps us make better decisions about where to deploy defensive techniques. Dolev–Yao's threat model [13] is employed to anticipate any security issues in our IoT network model. The attacker model is based on the following two assumptions:

- Cryptography is secure:
  1. The attacker is not able to decrypt a message without the key.
  2. The attacker is not able to compute HMAC without the key.
  3. The attacker is not able to guess an encryption key or a nonce.
- The attacker has a complete control over the system, so it has the ability to do the following:
  1. The attacker is able to initiate any number of parallel protocol sessions.
  2. The attacker is aware of all the public data of the protocol.
  3. The attacker benefits from all the privileges/keys of bad agents.
  4. The attacker is able to read, store, and block every message in transit.
  5. The attacker is able to create and transmit messages.
  6. The attacker is able to construct and deconstruct messages.
  7. The attacker is able to encrypt/decrypt if the encryption/decryption key is known.

The Dolev-Yao threat model aids in evaluating the security features of the proposed scheme. Hence, the security analysis and simulation of our scheme are provided using this model.

2.5 AVISPA Tool

Armando *et al.* [4] introduced Automated Validation of Internet Security Protocols and Applications (AVISPA), which is a toolkit used to validate and assess the Internet Security Protocols and Applications. AVISPA is a widely used platform in the research community for security protocol validation, and to demonstrate proof-of-concept, the authors have expressed and evaluated the specifications of several industrial-scale security protocols currently being drafted or standardized [58]. AVISPA is a role-oriented language where each agent plays a distinct role during the execution of the given protocol. The security protocols can be defined and specified under the AVISPA tool using High-Level Protocol Specification Language (HLPSL). HLPSL's semantics rest on Lamport's Temporal Logic of Actions (TLA). The main goal of HLPSL is to provide a means for verifying security properties such as data secrecy and authentication in message exchanges between agents. HLPSL provides a separate section to define the security properties, named the goal section. Thus, the security protocol is determined, whether SAFE or not based on predefined goals. HLPSL specifications are automatically translated into a lower language named the Intermediate Format (IF) using the HLPSL2IF translator. The main goal of these translations and designing the IF language is to offer and serve an adequate input to the various back-ends of the AVISPA tool set.

AVISPA has the following four back-end tools:

- OFMC Model Checker: The On-the-Fly Model-Checker (OFMC) incorporates several symbolic techniques and algebraic properties to explore the state space in a demand-driven way.

- CL-AtSe Model Checker: The Constraint-Logic-based Attack Searcher (CL-AtSe)

translates any security protocol specification written as a transition relation in IF language into a set of constraints that are effectively used to discover attacks if any on the protocol.

- SATMC Model Checker: SAT-based Model checker
  (SATMC) constructs a propositional formula based on Transitional state obtained from the IF specification. The propositional formula represents any violation of the security properties, which can be translated into an attack.

- TA4SP Model Checker: The Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP) identifies the vulnerability of a protocol or predicts the protocol correctness by accurate estimation of the intruder's capabilities.

2.6 OMNeT++ Simulator

The OMNeT++ discrete event simulation environment was developed in 1997 by András Varga [57] at the Technical University of Budapest, Hungary. It is an extensible, modular, component-based C++ simulation library and framework, primarily designed to simulate and build computer networks and protocols. The network simulator OMNeT++ is introduced to be utilized for the research on the communication infrastructure; thus it is utilized in the modeling of wired and wireless communication networks, and protocols.

2.7 Summary

This chapter provided a brief overview of the concepts, paradigms, and tools used in our work. The next chapter focuses on the main theme of this dissertation by surveying and

discussing related work.

Chapter 3

Related Work

In this chapter, we revisit and discuss recent progress in IoT authentication, focusing on machine-to-machine interaction. We consider two main research directions: authentication and access control. Although several proposals on IoT authentication have been published, a very limited amount of work has been done on lightweight IoT authentication.

3.1 Authentication

Over the years, and more recently, researchers have focused on addressing the IoT authentication taking into consideration the constrained resources of IoT devices and their networks.

Ukil *et al.* [56] proposed a lightweight symmetric-key based authentication scheme for the IoT applications. In the proposed work, the authors embedded a low overhead security mechanism comprised of both authentication with integrated key management and encryption on constrained application protocol (CoAP) for vehicle tracking systems. They capitalized on the option field of the CoAP header to embed the authentication

mechanism, by introducing two new options, namely, AUTH and AUTH_MSG_TYPE. The former is used to enable the authentication mode, whereas the latter is used to indicate various messages for an authentication session establishment. Moreover, the authors also minimized the amount of handshaking for reliability on the vehicle when a high speed is reached.

Chu *et al.* [11] proposed an authentication scheme based on Elliptic Curve Cryptography (ECC) for public and private key pair in order to satisfy the requirements for non-monolithic IoT environments. The elliptic curve public parameters are initialized and computed during the initialization phase. Next, these parameters are used during the authentication phase.

Lee *et al.* [30] introduced a lightweight mutual authentication protocol based on symmetric XOR encryption for RFID-based IoT systems. In this work, the authors removed complex encryption schemes such as asymmetric encryption and one-way hash function.

Zhao et al. [67] introduced an asymmetric mutual authentication scheme for an IoT system. In the proposed scheme, authentication occurs between the terminal device and the platform. The scheme is based on hashing and feature extraction. The authors enhanced IoT security and decreased computation and communication costs by combining one-way hashing using SHA1 and feature extraction. This combination helped in avoiding any collision attacks. However, the scheme did not support the following key properties: anonymity, unlinkability, and untraceability.

Santoso et al. [46] presented an IoT authentication scheme for a smart home system. The proposed protocol relies on an AllJoyn framework and capitalizes on Elliptic Curve Cryptography (ECC) for authenticating IoT devices. The scheme assigns the Wi-Fi gateway to initialize system configuration and to authenticate IoT devices. In addition, user access is controlled by a mobile device using an Android application. However, the proposed scheme

does not provide full end-to-end security guarantees because some key properties, such as anonymity, unlinkability, and untraceability, are not satisfied. In addition, the scheme relies on public key infrastructure, which is not efficient in terms of storage and computation for IoT-constrained devices.

Gaikwad *et al.* [16] utilized the three-level secure Kerberos authentication for an IoT smart home system. The proposed scheme employed symmetric algorithms, such as the advanced encryption standard (AES) and hash algorithms, for security. In this work, the authors did not consider IoT-device-to-IoT-device interactions. Furthermore, the proposed scheme does not suitably cover transactions anonymity, unlinkability, and untraceability.

Shivraj *et al.* [48] proposed a two-factor One Time Password (OTP) technique based on a lightweight identity-based ECC scheme. This technique was better in terms of efficiency and security when compared to existing methods for two reasons. First, the Key Distribution Center (KDC) does not require any key storage. Second, it does not store the private and public keys of the other devices. This protocol consumed a small amount of resources without negatively impacting security. There are two drawbacks to this method, i.e., the lack of machine-to-machine(M2M) roaming where a device wishes to control another device in another gateway, and the lack of support for the instance where a device wants to control using a different security scheme.

Hernandez *et al.* [22] presented a number of lightweight authentication and authorization mechanisms based on Slim Extensible Authentication Protocol over LAN (SEAPOL), which is an enhanced version of Extensible Authentication Protocol over LAN (EAPOL). The proposed mechanisms helped to embed authentication and authorization functionality onto constrained devices. The EAPOL weighs down the constrained devices by obliging them to implement and execute EAPOL in addition to Datagram Transport Layer Security (DTLS).

However, these proposed mechanisms are not only able to optimize the interoperability among the siloed IoT devices, but they are also able to tackle the security and privacy concerns in the IoT Environment.

Moosavi *et al.* [40] developed an IoT authentication and authorization architecture to maintain the security, privacy, and accuracy of patients' data. The authors utilized the distributed smart e-health gateways as an assistance party for medical sensors that tackle the authentication and security tasks. The main protocol in the proposed architecture is certificate-based DTLS handshake protocol. The architecture was tested via a healthcare prototype structure based on Pandaboard, Texas Instruments (TI) SmartRF06 board and WiSMotes performing medical sensing tasks. The effectiveness of denial-of-service (DoS) attacks was mitigated through the distributed nature of the proposed architecture. The evaluation results showed that the proposed architecture pares down the communication burden by 26% and reduces the latency of communications by 16% when compared to the delegation-based architecture.

Park *et al.* [42] presented a secure authentication method using zero-knowledge proof for a smart home environment. In this approach, no secret key is involved between the node and the home gateway during the authentication process. Instead, the home gateway provides the node with a number during the authentication phase. This number can be used later by the node to prove its authenticity to the home gateway.

Choi *et al.* [10] proposed a firmware validation and updating scheme for IoT smart home devices. This scheme capitalized on the ID-based mutual authentication and the key derivation algorithm for firmware image distribution. Furthermore, they capitalized on a hash chain and fragmentation process to perform the integrity verification of the firmware image, where the firmware image is fragmented into pieces and put to the hash chain for

verification purposes. The security analysis results showed that the proposed scheme could provide security in terms of known session key security, no unknown key sharing, no key control, perfect forward secrecy, and no key-compromise impersonation.

Kumar *et al.* [29] proposed a secure, lightweight session key establishment scheme using a short authentication token for smart home devices. The short authentication token played a big role in ensuring the mutual authentication between the constrained device and the home gateway. Consequently, the authors achieved better efficiency in terms of computation and communication when compared with previous works. The verification of the proposed scheme was done using an AVISPA security analyzer tool, and it was ensured that the security of this protocol behaved as expected.

More recently, Risalat *et al.* [45] presented a secure mutual authentication protocol based on one-way hash function for the IoT. This mutual authentication protocol enables dynamic updates for the secret key value between the tag and the reader. The evaluation results showed that the proposed protocol is able to resist various attacks, such as man-in-the-middle attacks, replay attacks, eavesdropping attacks, and tracing and de-synchronization attacks. There is one drawback to this method, i.e. the secret value (SV) is dynamically updated in a way that makes it easy to guess its sequence number increment.

Wilson *et al.* [62] introduced a machine-to-machine mutual authentication protocol based on asymmetric cryptography for IoT smart home environment. The proposed protocol allows the smart home nodes to authenticate each other and agree on a dynamic shared secret in every communication session without any human involvement. The evaluation result using SPAN/AVISPA toolkit showed that the protocol is safe and secure against well-known attacks, such as man-in-the-middle attacks, replay attacks, and eavesdropping attacks. This protocol is dependent on the pseudo-random number generation methods to

produce the secret keys. However, this method does not provide complete randomness. Moreover, although this protocol is effective, current constrained resource devices are not capable of handling asymmetric cryptography. The security might be compromised due to the disadvantages of a long time of key generation and exchange between nodes, smart home, and controller in the configuration phase.

Recently, Mishra *et al.* [39] introduced a strong authentication protocol using a smart card for an IoT-based wireless sensor network. The proposed protocol utilized password hash values and preshared keys to provide authentication between the gateway node and the sensor node. The authors stated that their authentication protocol achieved user anonymity and resisted various attacks. Authentication in IoT-device-to-IoT-device interactions was not supported in this work.

In a recent study, Dorri *et al.* [14] presented a blockchain-based approach as a security and privacy decentralized method associated with the distributed nature of IoT. The proposed framework refers to the end devices equipped with smart home as the miners, where each miner, in such a scheme, stores and preserves its blockchain that will manage and control secure communication path. The security and privacy analysis were conducted on the proposed system. The results showed that the overhead burden of the proposed system is manageable and minimal for the constrained resource devices.

3.2 Access Control

Most of the access control mechanisms that have been studied so far have been designed originally for non-resource constrained systems, such as a web-based system, which is not completely appropriate for highly resource-constrained IoT systems. Moreover, most of the

access control mechanisms were designed to control access between users and devices.

Anggorojati *et al.* [3] presented an access delegation model based on Capability-based Context-Aware Access Control (CCAAC) scheme designed for ubiquitous machine-to-machine communication in the IoT networks. This model ensures scalability, flexibility and secure authority delegation for distributed system via taking advantage of the identity-based capability-based access control approach, contextual information, and secure federated IoT. In this model, a capability is considered as the central element in access control mechanism, and an identifier is utilized to extend the scalability and to control the propagation of capability. Furthermore, the model benefits from context-awareness to facilitate dynamic access policy enforcement. The authors also introduced the term federated IoT, for the first time, as a baseline component that is responsible for managing the IoT network domain's principles in a federation by having some relevant rules and policies. This paper dealt with the access delegation between IoT devices, but it did not consider the automation of access control between the IoT devices.

Gusmeroli *et al.* [21] introduced a capability-based access control (CapBAC) system between users and IoT devices for managing the access control processes to services and information. The proposed system supports delegation of rights and sophisticated, customized access control. This work also did not consider the idea of automated access control between IoT devices in the bootstrapping phase.

Ye *et al.* [65] presented an efficient authentication and access control model for the Internet of Things (IoT). Upon the completion of the authentication phase, the authors implemented Attribute-based Access Control policy (ABAC) to ensure and restrict legitimate users to only access what they are authorized to access. ABAC policy defines role and identity as characteristics, and resources, such as sensitive information as resource attributes. In

this paper, the authors dealt only with the user-to-device access control and did not consider the automation of access control during the bootstrapping phase.

Taylor *et al.* [52] proposed a multi-tiered access control method between users and resource-constrained devices in the IoT network. The proposed approach integrates commonly available technologies and techniques such as physical proximity device, geo-location and encryption. Thus, it is inexpensive to implement. The proposed access control approach comprises five separated layers, namely, identification layer, transmission layer, verification layer, validation layer, and reporting layer. This work did not deal with the device-to-device access control.

3.3 Discussion

It appears from the above summary that most of the works on authentication protocols are dependent on symmetric and asymmetric algorithms, and it is known that asymmetric encryption requires much more computation than what the constrained resource devices in IoT can afford. However, a number of research studies have introduced the potential of using lightweight cryptographic functions such as hash function and bitwise XOR, but they did not consider strong mutual authentication; thus an efficient mutual authentication framework remains a challenge for the IoT ecosystem. Full mutual authentication can only be achieved by using a strong challenge-response mechanism. In addition, the access control process between IoT devices still requires human involvement. To our knowledge, no studies have automated this process yet. Our proposed approach, not only supports mutual lightweight cryptographic elements and entities, but it also introduces automated access controls for IoT. We introduce the access control process as part of the bootstrapping process, which currently

consists of authentication and key exchange processes. In the access control process, each device provides its domain-ID and capability as parameters during the bootstrapping process, which in turn facilitates the access control management process done by the smart home gateway. We also introduce a lightweight cumulative chained hash as a challenge-response authentication technique. This technique helps to secure the authentication process and create better assurance of the sender identity, and guarantee full mutual authentication. Our proposed approach also leverages the fog computing concept to secure the identities of IoT devices when they traverse the Internet. To make the bootstrapping process more lightweight, we capitalize on the lightweight cryptography algorithms such as one-way hash function and bitwise XOR encryption. The aforementioned techniques make our proposed schemes secure and robust. The next chapter presents a new secure, lightweight mutual authentication scheme for the IoT. It also presents different security mechanisms to improve IoT security.

Chapter 4

Secure Mutual Authentication and Automated Access Control for IoT Smart Home using Cumulative Keyed-hash chain

In this chapter, we present a secure, lightweight mutual authentication and key exchange protocol for the IoT smart home environment based on dynamic identity and cumulative Keyed-hash chain. Nodes can anonymously authenticate and establish a session with the controller node using dynamic identities and symmetric keys in an unlinkable manner. Moreover, the enforcement of a security policy between nodes is ensured by setting up virtual domain segregation and restricting nodes capabilities of sending and receiving instructions and commands to or from other nodes. A cumulative keyed-hash chain mechanism is introduced as a way to ensure the identity of the sender (through challenge-response). Furthermore, we capitalize on the fog computing concept to improve identity assurance.

In addition, we formally evaluate and prove the security of our protocol by using the Burrows-Abadi-Needham (BAN) logic and the Automated Validation of Internet Security Protocols and Applications (AVISPA) toolkit. Finally, we implement the protocol using the OMNeT++.

4.1 Security Mechanisms

In this section, we present a smart home network model based on fog computing, and introduce the security mechanisms and entities underlying our proposed protection scheme, and then discuss the threat model and attack surface it entails.

4.1.1 IoT smart home network model based on fog computing

Our proposed authentication protocol consists of three participants, namely, IoT node (N), controller node (CRN), and manufacturer fog node (MFR), as shown in Fig. 4.1. A local IoT network includes one or several nodes N and a controller node CRN. Nodes N are constrained resource devices while CRN is not. MFR can be either within or outside the IoT network. N can make direct communication with CRN, but communication between nodes is subject to CRN's permission. MFR, which will be in charge of keeping track of each node N and it's direct home CRN, is a capable computing device such as a gateway device, a smartphone, among others. A pre-trust relationship exists between the node N and the MFR.

MFR provisions node N with a MFR identity (ID_{MFR}) and Master secret key (K_{MFR}), which are assumed to be securely stored in the node's memory. When node N starts to authenticate itself with CRN which is unaware of ID_{MFR} and ID_N at the beginning of the

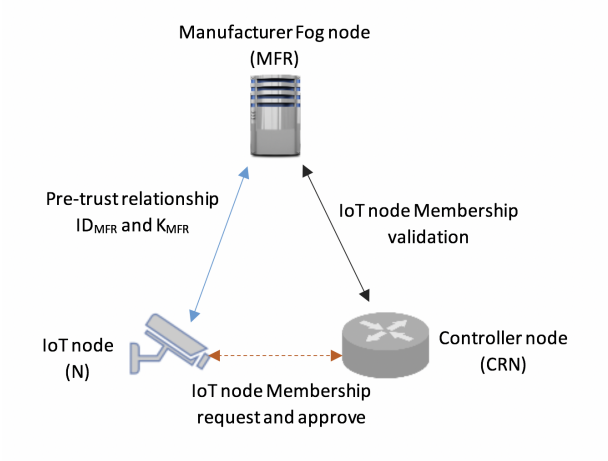

Figure 4.1: IoT smart home network model based on Fog computing

authentication process, it creates a Proof of Belonging (PoF) parameter to take advantage of the pre-established relationship in order to facilitate the IoT node membership validation process done by MFR and to inform the MFR of its real direct home CRN. PoF is created by hashing some secure parameters only shared between node N and MFR, such as Master secret key (K_{MFR}), along with other random fresh authentication parameters as explained in Subsection 4.2.3. In our model, we assume that there are different manufacturer fog nodes sharing the same database. Thus, all manufacturer fog nodes become aware of all IoT nodes' direct home controllers after the IoT nodes have been authenticated and registered. This assumption facilitates the data transfer process from one node in one IoT smart home to another node residing in different IoT smart home in a secure manner.

4.1.2 Automated access control

We propose an automated access control model based on the notion of Virtual segregation of IoT network and access capabilities. The concepts behind virtual domain segregation and devices' capabilities are outlined separately in the following subsections.

4.1.2.1 IoT virtual domain segregation

To apply the access control between nodes, we introduce a simple technique called Virtual segregation of IoT network (VSIoTN) as shown in Fig. 4.2. VSIoTN allows the IoT system administrators to virtually partition the IoT networks based on the IoT nodes' capabilities of sending and receiving instructions, and the scope of interaction between these nodes into different domains during the registration phase (presented in Subsection 4.2.2). This will help CRN to control the communication flows in the IoT smart home network. VSIoTN is the key to secure the IoT smart home network so that all IoT nodes that should communicate and interact with each other should be placed in the same domain. For example, in Fig. 4.3 door camera, door locker and front door lamp nodes reside in the same virtual domain denoted B, while bed, coffee, room lamp and shower nodes are put in the same domain denoted A.

However, there is a case where some nodes located in different virtual domains need to communicate with some nodes outside the range of their domains. For example, assume that camera node in virtual domain A and bed node in virtual domain B wish to communicate with air-conditioner node in different domain, which is neither located in virtual domain A nor in the virtual domain B. In this case, we introduce a common virtual domain-let's say-(C) where the shared nodes such as air-conditioner can be placed in, as shown in Fig. 4.3. The system administrator securely implants each node with its virtual domain ID (VD_N) and its capability type (Ca_N). This happens only once, during the registration phase (as discussed in Subsection 4.2), and it is the only occasion where human (administrator) intervention is required. Subsequently, as each node has its virtual domain ID (VD_N) and its capability type (Ca_N) securely implanted in its memory, it will automatically initialize the access control process each time it starts the authentication process with the Controller.

Hence, the subsequent steps beyond the registration phase will fully be automated.

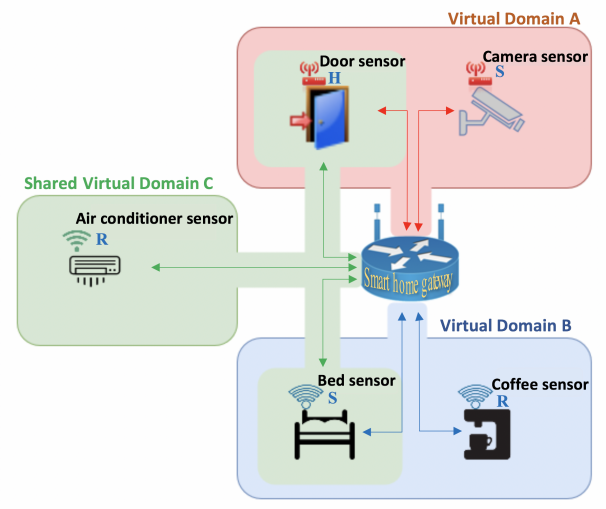

Figure 4.2: Division of Devices Based on Virtual Domain Segregation, with Shared Domain

4.1.2.2 Device capability assignment

In the IoT smart home network, nodes have three different types of access capabilities when considering the communications among nodes, not between nodes and the controller. Some nodes are only able to send and give instructions, some nodes are only able to receive and execute instructions, and the others are able to both give and execute instructions. Therefore, each node in the IoT network will be assigned one of the following capabilities: S denotes that the device is a sender, R denotes that the device is a receiver, and H denotes that the device is hybrid (sender and receiver). The Node capability assignment method complements the above virtual domain segregation strategy to strengthen the IoT network security. This will ensure that if a hacker compromises one device, her/his capability will not only be restricted by the virtual domain but also by the device capability. For example, in Fig. 4.3, if a hacker compromises coffee sensor, she cannot sabotage the other devices even if these resided in the same domain because the coffee sensor is assigned as a receiver,

and thus it cannot give instructions to other devices.

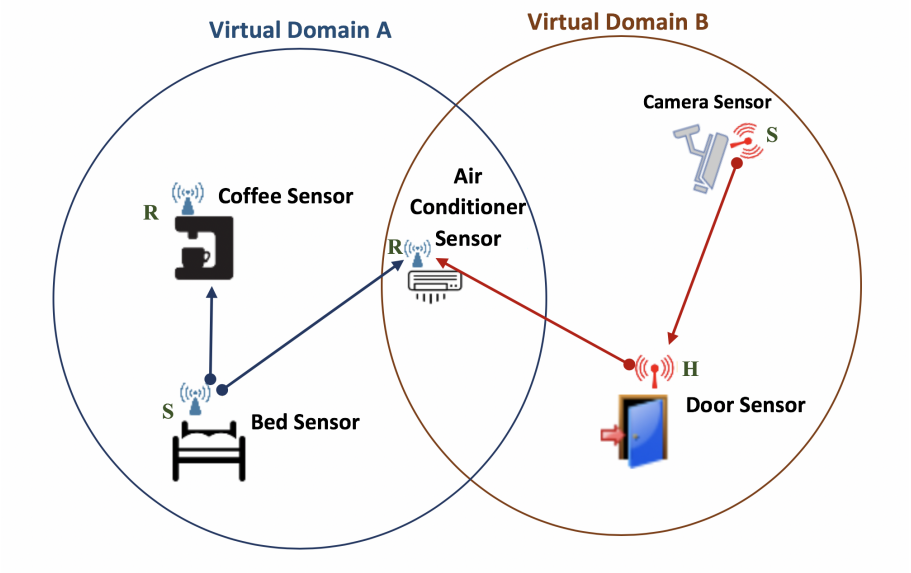

Figure 4.3: Capability Assignment and attack scenarios on IoT smart home network

4.1.3 Cumulative keyed-hash chain as a challenge response authentication

In order to have a full mutual authentication, a challenge-response authentication mechanism must be considered. Hence, we introduce a lightweight one-way challenge-response authentication mechanism. In this method, the sending node N first hashes the information that it wants to send to the controller and produces a hashed value-let's say-H_1. Second, it hashes and chains H_1 with two other values, namely, the shared secret key (TK_N) between itself and the controller node CRN, and the cumulative hash value ${}^{x}CH_F$, previously, stored in the Keyed-hash chain database, where x represents the sequence number or order of values. Then, it attaches the resultant value-let's say-H_F of the previous step in the authentication message to be sent to the controller node: $H_F = \mathrm{h}(H_1, TK_N, {}^{x}CH_F)$. Finally, it hashes the resultant value H_F again with the shared secret key TK_N and stores it in the Keyed-hash

chain database as a new value: ${}^{x+1}CH_F = \mathrm{h}(H_F, TK_N)$. The process is illustrated in Fig. 4.4.

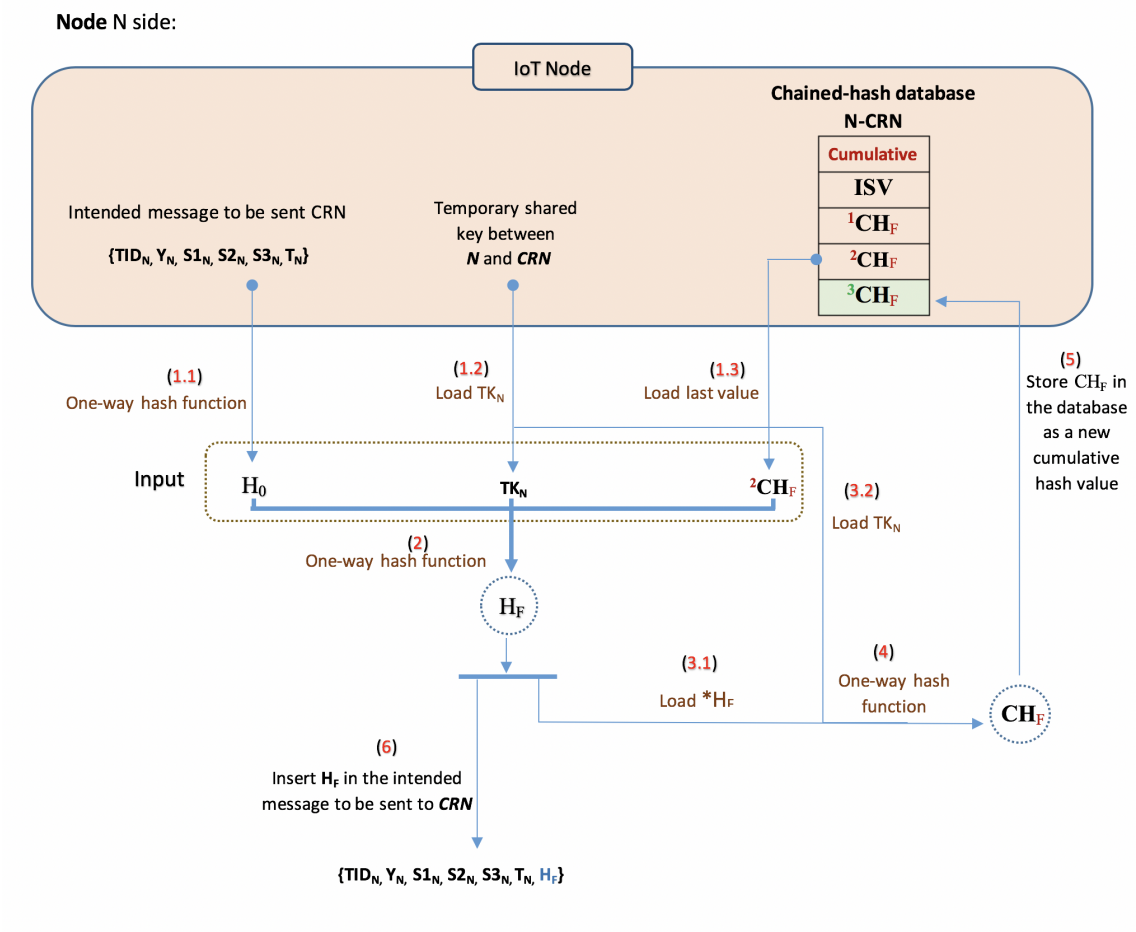

Figure 4.4: Cumulative Keyed-hash chain process at N side

On the other side, the controller node CRN will apply the same technique to ensure the authenticity of the received hashed value upon receipt of the data. First, it hashes all information that was received from N but the hash value-let's say- H_1'. Second, it hashes and chains H_1' with two other values, namely, the shared secret key (TK_N) between itself and the controller node CRN, and the cumulative hash value (${}^{x}CH_F$) (previously) stored in the Keyed-hash chain database, where x represents the order of values. Then, it compares the resultant value (${}^{*}H_F = \mathrm{h}(H_1', TK_N, {}^{x}CH_F)$) with the one that was received H_F. If it matches, it will authenticate the sender node N and hash the resultant value H_F again with

the shared secret key TK_N, and then store it in the Keyed-hash chain database as a new value: $^{x+1}CH_F$. Otherwise, it will reject the sender node N. The process is illustrated in Fig. 4.5.

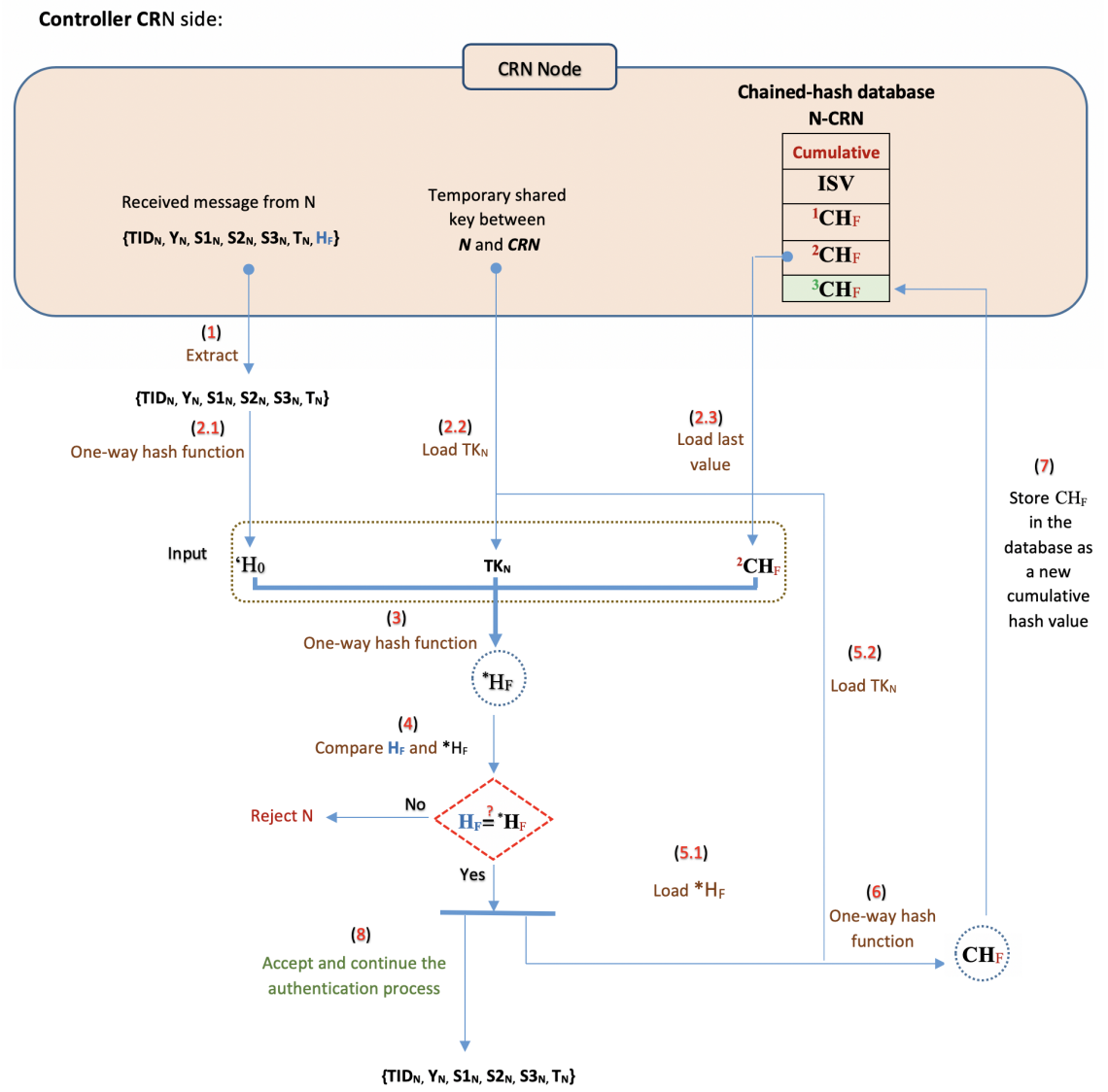

Figure 4.5: Cumulative keyed-hash chain process at CRN side

It is worth mentioning that our mechanism chains the blocks of sessional hash values together by hashing the new session hash value with two values, namely, temporal cryptographic key and previous cumulative hash value. Thus, our mechanism looks like a

blockchain technology at first blush, but it actually does not utilize blockchains per se. It can be argued that an attacker can compromise or find out the Keyed-hash chain entries (mainly the first one), then she would only need to know the secret key to crack this step of the protocol. However, finding the secret key is not easy since it changes in every session. Even if the adversary knows the secret key or the first hash, it is useless for her since an ephemeral session key is being used.

4.1.4 Challenge-response Mechanism based on Transaction History

As aforementioned in the previous subsection, N and CRN store and keep all cumulative hash values ${}^{x}CH_F$ in their synchronized chained-hash database N-CRN. These values can be utilized to introduce a historical factor for authenticating IoT nodes, as illustrated in Fig. 4.6. Authentication using history factor ensures mutual authentication through a challenge-response scheme and the mutual authentication is of high importance in IoT node-to-node authentication. This two-way challenge/response allows the controller to check the authenticity of the IoT node, and the IoT node to ensure that it is not communicating with a malicious controller. Cumulative hash history-based authentication counts on the ability of the IoT node and controller to show proof of knowledge of past cumulative hash values. The approach involves tracking and storing securely the cumulative hash values related to the interaction over time between the IoT node N and the controller CRN. In this process, when CRN receives an authentication request message from N, it starts the authentication process and triggers a challenge/response. CRN generates a challenge c (information about random cumulative hash value ${}^{x}CH_F$ stored previously), hashes the challenge c with the shared secret key and current timestamp h(TK_N,c, T_{CRN}), and sends it to N. N sends the response back using h(TK_N,r, T_N), where r is the response (cumulative hash value ${}^{x}CH_F$)

and T_N is a new timestamp for when the response is sent ($T_{CRN} < T_N$). CRN verifies the correctness of the received ${}^{x}CH_F$ value by comparing it with the one stored in its chained-hash database N-CRN. If they are equal, CRN will accept N and resume the authentication process. Otherwise, N will be rejected.

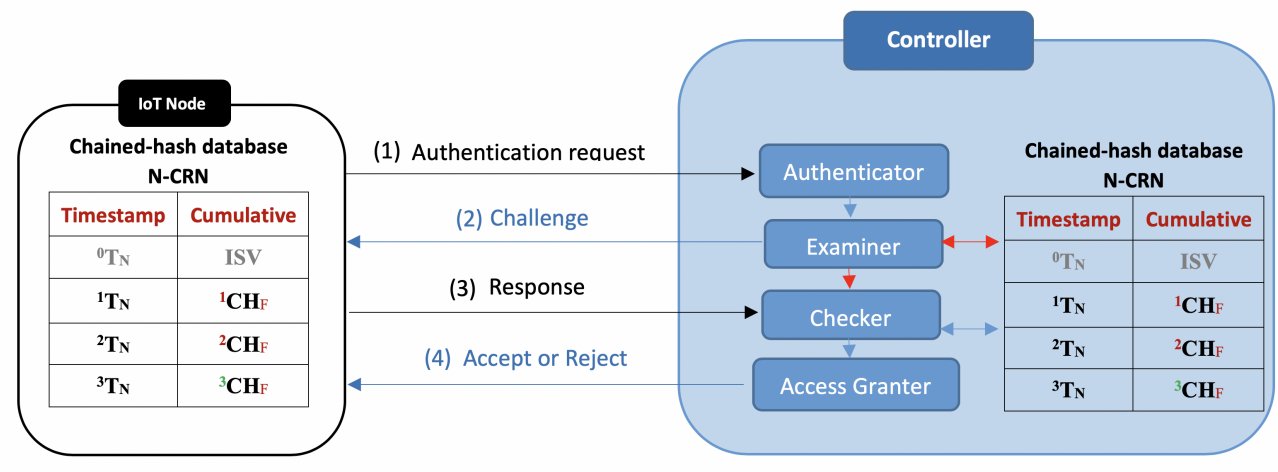

Figure 4.6: Historical authentication process

4.2 Proposed authentication protocol

The abstract notations used to describe our authentication protocol are listed in Table 4.1. There are three phases in our protocol: Initialization phase, registration phase, and authentication and key exchange phase.

The manufacturer will load the IoT devices with the device ID and secret key regarding the fog server. This will help the creation of a trusted communication/relation bridge between the manufacturer and home controller node. The system administrator (SA) is responsible for performing the initialization and registration phases. In the authentication phase, a node (N) and a Controller node (CRN) share the responsibilities for ensuring a secure anonymous mutual authentication and session key exchange. The relation between

the nodes N and CRN is based on a client-server model. We assume the node N is allowed to make a direct communication with CRN. We will focus on the design and development of our authentication protocol and discuss the aforementioned phases in details.

4.2.1 Initialization phase

In this phase, the system administrator (SA) is responsible to securely initialize the controller node CRN with the controller node master key (K_{CRN}). The required steps are listed below.

Step 1: The SA selects a random master secret key K_{CRN} for CRN.

4.2.2 Registration phase

In this phase, the SA securely registers the node N. The SA specifically implants the node's real identity (ID_N), the node's virtual domain ID (VD_N) and the node's capability type (Ca_N) into three authentication parameters, namely $S1_N$, $S2_N$ and $S3_N$, using one-way hash function and bitwise XOR operation. VD_N indicates which virtual domain the IoT node belongs to as explained in Subsection 4.1.2.1, and Ca_N specifies the IoT node's capability of sending and/or receiving instructions to or from other nodes as explained in Subsection 4.1.2.2. Moreover, the SA uses the controller node master key K_{CRN} and IoT node identity ID_N during the computation of $S1_N$, $S2_N$ and $S3_N$. In addition, we assume that the SA has a table that contains a number of virtual domains and a list of devices residing in each of these virtual domains (this information should be available in the Network Infrastructure Document). Moreover, the capability of each node is already determined. We assume a mechanism, e.g. accelerometer, is available and used by the manufacturer at the time of manufacturing the IoT nodes and used to generate the nodes' identities to ensure uniqueness of identities. So, the node's identity will be implanted in each node by the manufacturer in

a secure manner. The steps involved in registering the node N are listed below.

Step 1: The SA uses the unique node's identity ID_N that was securely inserted in the node by the manufacturer.

Step 2: The SA selects the proper virtual domain VD_N of node N.

Step 3: The SA selects the proper capability type Ca_N of node N.

Step 4: The SA computes three parameters $S1_N$, $S2_N$ and $S3_N$ as follows:

- $S1_N = ID_N \oplus h(K_{\text{CRN}}, ID_N)$.
- $S2_N = K_{\text{CRN}} \oplus S1_N \oplus VD_N$.
- $S3_N = Ca_N \oplus h(K_{\text{CRN}}, S2_N)$.

It is worth mentioning that the K_{CRN}, VD_N and Ca_N are only utilized to create $S1_N$, $S2_N$ and $S3_N$. K_{CRN}, VD_N and Ca_N will not be stored at any node but the CRN. Using the secure K_{CRN} and ID_N parameters in computing $S1_N$, $S2_N$ and $S3_N$ and applying the one-way hash functions make it almost impossible for an adversary to extract ID_N, VD_N and Ca_N from $S1_N$, $S2_N$ and $S3_N$ parameters in transit.

Step 5: The SA securely stores ID_{CRN}, $S1_N$, $S2_N$, $S3_N$ in the node N's memory.

4.2.3 Authentication and key exchange phase

In the authentication phase, the node N authenticates anonymously with the controller node CRN. The steps involved in the authentication between the node N and the controller node CRN are summarized in Fig. 4.7 and described in more details below.

1. **N** $\xrightarrow{(\mathbf{TID_N}, \mathbf{ID_{MFR}}, \mathbf{PoF}, \mathbf{Y_N}, \mathbf{S2_N}, \mathbf{S3_N}, \mathbf{T_N}, \mathbf{H_F})}$ **CRN**

Table 4.1: Notations used in our protocol

Notation	Description
SA	System Administrator
N	IoT Node
CRN	Controller node
MFR	Manufacturer fog node
ID_N	Real identity of sensor node N
TID_N	Temporary identity of sensor node N
K_{CRN}	Master secret key of CRN
K_{MFR}	Master secret key of Manufacturer MFR
TK_N	Temporary secret session key picked by N
R_N	Random number picked by N
$S1_N$, $S2_N$, $S3_N$	Authentication parameters stored in N's memory
ISV	Initialization Seed Value picked by N
PoF	Proof of belonging
X_N , Y_N	Additional parameters required for authentication
G_{CRN}	Authentication parameter computed by CRN
VD_N	Node's Virtual Domain number
Ca_N	Node capability type
T_N	A timestamp generated by node N
T_{CRN}	A timestamp generated by node CRN
T_{MFR}	A timestamp generated by node MFR
h(·)	Hash function
H_F	Keyed-chain of hashed Instructions
${}^{x}CH_F$	Cumulative Hashed Value
$\oplus$	Bitwise XOR operation
N $\rightarrow$ CRN: M	Node N sends the message M to controller CRN via a public channel

The node N prepares and computes the authentication message to be sent to the CRN. The aim is to securely send the ID_N, TK_N, VD_N and Ca_N parameters to the CRN. The steps involved in this process are listed below.

1.1. N computes $X_N = S1_N \oplus ID_N$ to be used during the authentication process.

1.2. N selects a random parameter R_N.

1.3. N computes $Y_N = X_N \oplus R_N$ to be used during the authentication process.

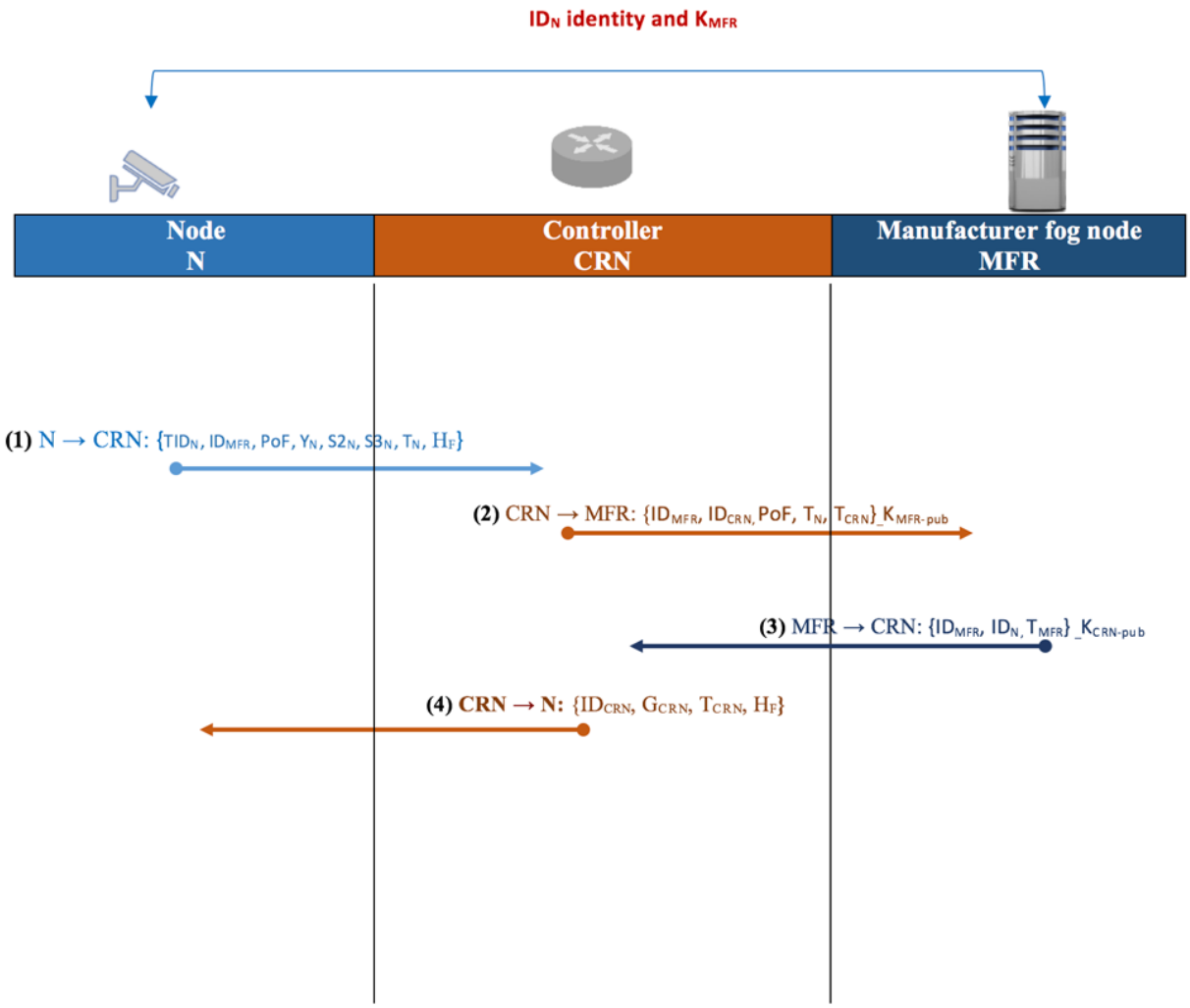

Figure 4.7: The authentication and key exchange phase of our protocol for centralized smart home environment

1.4. N computes a temporary session key $TK_N = h(ID_N, R_N)$. This will be used to secure communication with the CRN in a given session.

1.5. N generates a new timestamp T_N that will be used to avoid replay attack.

1.6. N computes a temporary identity $TID_N = h(ID_N \oplus T_N, R_N)$ that will support the anonymity of the node in a given session.

1.7. N hashes the timestamp, its own identity and the manufacturer identity (ID_{MFR})

to create the proof of belonging $PoF = h(h(ID_{\text{MFR}} \oplus K_{\text{MFR}}, ID_{\text{CRN}}), T_N) \oplus ID_N$

1.8. N computes H_F=h(h(TID_N, ID_{MFR}, PoF ,Y_N, $S2_N$, $S3_N$,T_N), TK_N, $^{x}CH_F$) that will be used as part of the cumulative keyed-hash chain for the challenge-response authentication.

1.9. N securely stores the TID_N and TK_N parameters in its internal database as shown in Table 4.2.

Table 4.2: Node N Database

Node Identifier	Timestamp	Temporary Identity	Temporary session key
ID_N	T_N	TID_N	TK_N

1.10. N sends the tuple (TID_N, ID_{MFR}, PoF ,Y_N, $S2_N$, $S3_N$, T_N, H_F) to the CRN.

1.11. N computes the new cumulative hashed value $^{x+1}CH_F$ = h(H_F,TK_N), and inserts it in the Cumulative Keyed-hash chain table as shown in Table 4.3.

Table 4.3: Cumulative Keyed-hash chain table N-CRN

Value
$^{x}CH_F$
$^{x+1}CH_F$

2. **CRN** $\xrightarrow{(\mathbf{ID_{MFR}}, \mathbf{ID_{CRN}}, \mathbf{PoF}, \mathbf{T_N}, \mathbf{T_{CRN}})\_\mathbf{K_{MFR\text{-}Pub}}}$ **MFR**

Upon the receipt of N's message by the CRN, the CRN generates a new timestamp T_{CRN} and forwards the tuple PoF, ID_{CRN}, T_{CRN} and T_N to the manufacturer fog node (MFR) in order to obtain the IoT node's real identity (ID_N).

2.1. CRN checks the validity of the received timestamp (t* − T_N < ΔT? continue: Drop), here t* is the time when the message is received and ΔT is the maximum transmission delay. The message is dropped if the predicate is not valid.

2.2. CRN generates a new timestamp T_{CRN}.

2.3. CRN encrypts the tuple ID_{MFR}, ID_{CRN}, PoF, T_N and T_{CRN} using the manufacturer fog node's public key.

2.4. CRN forwards the tuple (ID_{MFR}, ID_{CRN}, PoF, T_N, T_{CRN})$\__{\text{KMFR-pub}}$ to the MFR.

3. **MFR** $\xrightarrow{(\,\mathbf{ID_{MFR}},\,\mathbf{ID_N},\,\mathbf{T_{MFR}})\_\mathbf{K_{CRN\text{-}Pub}}}$ **CRN**.

Upon the receipt of CRN's message by the MFR, the MFR checks the validity of the timestamp T_{CRN} and extracts ID_N from the proof of belonging (PoF). If all the conditions are justified, the MFR updates the N information and adds its direct CRN address as shown in the Table 4.4. Then it inserts the ID_N in the message to be sent to the CRN. It generates a new timestamp T_{MFR} and then it forwards the tuple ID_N, ID_{CRN} and T_{MFR} to the CRN, so that the CRN can resume the authentication process with the node.

Table 4.4: Manufacturer database

Node Identity	Secret key of the manufacturer (MFR)	Direct Controller Node ID
ID_N	K_{MFR}	ID_{CRN}

The MFR performs the following steps:

3.1. MFR checks the validity of the received timestamp (t* $- T_{\text{CRN}} < \Delta$T? continue: Drop), here t* is the time when the message is received and ΔT is the maximum transmission delay. The message is dropped if the predicate is not valid.

3.2. MFR extracts the ID_N from the PoF as follows:

- MFR computes $Z = h(h(ID_{\text{MFR}} \oplus K_{\text{MFR}}, ID_{\text{CRN}}), T_N)$.
- MFR extracts $ID_N = PoF \oplus Z$.

3.3. MFR updates the node N information and adds ID_{CRN} as its direct controller.

3.4. MFR prepends ID_N to the message.

3.5. MFR generates a new timestamp T_{MFR}.

3.6. MFR encrypts the tuple $(ID_{\text{MFR}}, ID_N, T_{\text{MFR}})$ using CRN's public key.

3.7. MFR forwards the tuple $(ID_{\text{MFR}}, ID_N, T_{\text{MFR}})\_\text{K}_{\text{CRN-pub}}$ to the CRN node.

4. **CRN** $\xrightarrow{(\,\mathbf{ID_{CRN}}, \mathbf{G_{CRN}}, \mathbf{T_{CRN}}, \mathbf{H_F}\,)}$ **N**

Upon the receipt of MFR's message by the CRN, the CRN checks the validity of the timestamp T_{MFR}. If the condition is justified, the CRN continues to authenticate N. It uses the ID_N and K_{CRN} key to extract N's temporary session key (TK_N), node N's virtual domain (VD_N) and node N's capability (Ca_N) parameters. Then, the CRN checks the validity of the cumulative Keyed-hash chain H_F. The steps involved in this process are listed below.

4.1. CRN decrypts MFR message using its own private key.

4.2. CRN checks the validity of the received timestamp (t* $- T_{\text{MFR}} < \Delta$T? continue: Drop), here t* is the time when the message is received and ΔT is the maximum transmission delay. The message is dropped if the predicate is not valid.

4.3. CRN extracts the ID_N from the MFR message.

Now the CRN can use CRN's secret key K_{CRN} and node real identity ID_N to extract the N's parameters.

4.4. CRN computes $`S1_N = ID_N \oplus h(K_{\text{CRN}}, `ID_N)$.

4.5. CRN computes $`X_N = `S1_N \oplus `ID_N$.

4.6. CRN computes $`R_N = `X_N \oplus Y_N$.

4.7. CRN computes '$TID_N = h(`ID_N \oplus T_N, `R_N)$.

4.8. CRN checks the validity of the received TID_N ('$TID_N = TID_N$? Continue: Drop). The message is dropped if the condition is not met.

4.9. CRN computes '$TK_N = h(`ID_N, `R_N)$.

4.10. CRN extracts from its cumulative hash database the last value $^{x}CH_F$ stored in the previous authentication session and computes $*H_F$=h(h(TID_N, ID_{MFR}, PoF, Y_N, $S2_N$, $S3_N$, T_N),TK_N,$^{x}CH_F$).

4.11. CRN checks the validity of the received H_F ($*H_F = H_F$? Continue: Drop). The message is dropped if the condition is not met.

4.12. If the above condition is met, CRN will compute the new cumulative hashed value $^{x+1}CH_F$ = h(H_F,TK_N), and then insert it in the Cumulative Keyed-hash chain table as a new cumulative hashed value as shown in Table 4.5.

Table 4.5: Cumulative Keyed-hash chain table

Value
$^{x}CH_F$
$^{x+1}CH_F$

4.13. CRN computes the virtual domain '$VD_N = K_{CRN} \oplus `S1_N \oplus S2_N$.

4.14. CRN computes the capability type '$Ca_N = S3_N \oplus h(K_{CRN}, S2_N)$.

The CRN now can securely store N's parameters (ID_N, TID_N , TK_N, VD_N and Ca_N) in its database as shown in Table 4.6. These parameters will be used to establish connections with N until these parameters are updated by the node N. Now, the CRN prepares and computes the G_{CRN} parameter, which is a hash value of the tuple (ID_{CRN}, X_N, R_N, ID_N, TK_N, T_{CRN}), and sends the G_{CRN}

Table 4.6: Controller node CRN Database

Identity	Temporary Identity	Temporary Symmetric key	Node Virtual Domain ID	Node capability type	Fog server
ID_N	TID_N	TK_N	VD_N	Ca_N	ID_{MFR}

back to N so that N makes sure that the CRN was successfully able to extract its secure parameters.

4.15. CRN generates a new timestamp 'T_{CRN}.

4.16. CRN computes $G_{\text{CRN}} = h(X_N, R_N, ID_N, TK_N, T_{\text{CRN}})$.

4.17. CRN adds the tuple (G_{CRN}, T_{CRN}) to the message.

4.18. CRN computes H_F = h(h(ID_{CRN},G_{CRN}, 'T_{CRN}), TK_N, $^{x+1}CH_F$) and inserts it in the message.

4.19. CRN sends the tuple (ID_{CRN}, G_{CRN}, 'T_{CRN}, H_F) to N.

4.20. CRN computes the new cumulative hashed value $^{x+2}CH_F$ = h(H_F,TK_N), and inserts it in the Cumulative Keyed-hash chain table as shown in Table 4.7.

Table 4.7: CRN's Cumulative Keyed-hash chain table

Value
$^{x}CH_F$
$^{x+1}CH_F$
$^{x+2}CH_F$

5. **N**

Upon the receipt of CRN's message by N, N checks the validity of the timestamp T_{CRN}. If all is well, the node N continues and computes the 'G_{CRN} parameter and compares it with the received G_{CRN}. The steps involved in this process are listed below.

5.1. N checks the validity of the received timestamp (t* − T_{CRN} < ΔT? continue: Drop), here t* is the time when the message is received and ΔT is the maximum transmission delay. The message is dropped if the condition is not justified.

5.2. N extracts $^{x+1}CH_F$ from its cumulative hash database and computes $*H_F$ = h(h(ID_{CRN}, G_{CRN}, 'T_{CRN}), TK_N, $^{x+1}CH_F$).

5.3. N checks the validity of the received H_F ($*H_F = H_F$? Continue: Drop). The message is dropped if the condition is not met.

5.4. N computes the new cumulative hashed value $^{x+2}CH_F$ = h('H_F ,TK_N), and inserts it in the Cumulative Keyed-hash chain table as shown in Table 4.8.

Table 4.8: N's Cumulative Keyed-hash chain table

Value
$^{x}CH_F$
$^{x+1}CH_F$
$^{x+2}CH_F$

5.5. N computes 'G_{CRN} = h ($X_N, R_N, ID_N, TK_N, T_{\text{CRN}}$).

5.6. N checks the validity of the received G_{CRN} ('$G_{\text{CRN}} = G_{\text{CRN}}$? Continue:Drop). The message is dropped if the condition is not met.

At this moment, mutual authentication has been established between N and CRN.

At the start of a mutual authentication round trip, the more recent entry in the cumulative hash-chain is $^{x}CH_F$. At the end of a successful mutual authentication, two more values are added to the hash-chain, with indices (x+1) and (x+2), respectively. Those values are added progressively as the process unfolds, and must be removed in case where the authentication fails. If the authentication of N by CRN fails, N will be notified accordingly, and it will roll back the hash-chain update by removing the stored intermediary value $^{x+1}CH_F$. Else if the

authentication of CRN by N fails, CRN will be notified accordingly, and it will remove the last 2 updates. N will also roll back its hash-chain by removing the previously stored value at index (x+1).

4.2.3.1 First time authentication

The first time node N submits a request for authentication with the CRN, it initializes the head of the hash chain ${}^0CH_F$ by generating an initialization seed value (ISV). This initialization seed is sent unencrypted in the first transmitted packet in the authentication process, and used to generate the head of the hash chain at the CRN side. The first message in the authentication protocol, shown above, is updated to include the ISV as follows:

1. **N** $\xrightarrow{(\mathbf{TID_N}, \mathbf{ID_{MFR}}, \mathbf{PoF}, \mathbf{Y_N}, \mathbf{S2_N}, \mathbf{S3_N}, \mathbf{ISV}, \mathbf{T_N}, \mathbf{H_F})}$ **CRN**

In phase 4 of the protocol, after checking the validity of the TID_N (step 4.8), the CRN extracts and uses the ISV as the head ${}^0CH_F$ of the hash chain database at his side. One may say there is a potential risk when sending the ISV in plain text because an adversary can intercept this value. However, the ISV will be useless to the attacker for two reasons. First, as mentioned above in the step 4.10, the CRN will ensure the integrity of the ISV when computing $*H_F$, which is protected using one-way hash function of the message parameters, which include the timestamp T_N, and the temporary secret key TK_N, so any change to the ISV will be detected. Second, even if the attacker keeps the ISV and tries to reconstruct the cumulative hash value later on, she will fail because of the dynamic change of the temporary secret key TK_N in every session, as illustrated in Subsection 4.1.3.

4.3 Protocol Efficiency Evaluation

In this section, we analyse the overhead and efficiency of the proposed protocol in terms of storage, computation, communication and energy consumption.

4.3.1 Storage requirements

In our scheme, each IoT node is required to store its real identity ID_N, temporary identity TID_N, temporary secret key TK_N, manufacturer identity $\mathrm{ID_{MFR}}$, manufacturer master key $\mathrm{K_{MFR}}$, and the identity of the controller $\mathrm{ID_{CRN}}$. We use SHA-1 as an example of hash function, and the output of SHA-1 is 160 bits. By applying these settings, we obtain $|TID_N| = |S1_N| = |S2_N| = |S3_N| = |TK_N| = |^{\mathrm{x}}CH_F| = 160$ bits, while $|ID_N| = |ID_{\mathrm{CRN}}| = |ID_{\mathrm{MFR}}| = |K_{\mathrm{MFR}}| = 16$ bits, which is assumed short. On the other hand, CRN is required to store the tuple ID_N, $\mathrm{ID_{MFR}}$, TID_N, TK_N, $^{\mathrm{x}}CH_F$, VD_N, Ca_N, in addition to the CRN master key $\mathrm{ID_{CRN}}$. By applying these settings, we obtain $|TID_N| = |TK_N| = |^{\mathrm{x}}CH_F| = 160$ bits, while $|ID_N| = |ID_{\mathrm{MFR}}| = |K_{\mathrm{CRN}}| = |VD_N| = |Ca_N| = 16$ bits, which as aforementioned is assumed short.

Table 4.9: Storage cost of our scheme

Node	Storage cost (in bits)
N	1024
CRN	80m + 480

Note that the storage cost for the hash chain (for N and CRN) is covered by $|^{\mathrm{x}}CH_F| = 160$ bits. There are two different implementation options for the cumulative chain. In the first option, N and CRN keep only the last updated cumulative hash value $^{\mathrm{x}}CH_F$ and remove all previous values every time N and CRN complete a mutual authentication, so in this case each IoT node N is required to store 1024 bits, and the total storage required by CRN is

$(m \times (5 \times 160) + (3 \times 160)) = (80m + 480)$ bits, where m is the number of registered IoT nodes. The storage requirements are summarized in Table 4.9.

In the second option, N and CRN keep all cumulative hash values ${}^{x}CH_F$, so in this case each IoT node N is required to store $((160 \times n) + 864)$ bits where n represents the number of ${}^{x}CH_F$ values, and the total storage required by CRN is $(m \times (5 \times 16) + (160 \times n) + (2 \times 160)) = (80m + 160n + 320)$ bits, where m is the number of registered IoT nodes and n represents the number of ${}^{x}CH_F$ values.

Our proposed model is based on the first option, which is obviously much lighter in terms of storage cost.

4.3.2 Computation cost

Our proposed scheme is a computationally lightweight scheme designed for IoT smart home environment. The scheme ensures high security using only simple hash and XOR computations, thus consumes less storage than other related schemes. While the novelty of the proposed scheme is adding multiple security layers (e.g., access control and cumulative Keyed-hash chain) and ensuring IoT identities, it also provides relatively low computation cost.

Our scheme uses two operations as aforementioned, namely XOR operation and one-way hash function. Let T_h and T_{xor} be the computation times of one hash invocation and one XOR operation, respectively. Considering the authentication steps involved in our protocol and outlined in Fig. 4.7, the IoT node N performs 8 hash invocations and 5 XOR operations, which yields a total computation cost of $8 \times T_h + 5 \times T_{xor}$. The computation time of XOR operation is very trivial and can be ignored, so we can assume $T_{xor} \approx 0$. On the other hand, the controller node CRN performs 9 hash invocations and 7 XOR operations,

which yield a total computation cost of $9 \times T_h + 7 \times T_{\text{xor}}$. Therefore, the total computation cost of N is $8 \times T_h + 5 \times T_{\text{xor}} \approx 8T_h$, while the computation cost of CRN corresponds to $9 \times T_h + 7 \times T_{\text{xor}} \approx 9 \times T_h$. The computation cost are summarized in Table 4.10.

Table 4.10: Computation cost of our scheme

Node	Computation cost
N	$8\ T_h$
CRN	$9\ T_h$

4.3.3 Computation time and energy consumption

The invocation of SHA hash function takes 0.054 ms using a 32-bit Cortex-M3 micro-controller running at 72 MHz [31] [50]. It follows from Table 4.10 that an IoT sensor node N takes 0.43 ms, while the controller node CRN takes 0.49 ms. The same micro-controller, in ambient/room temperature (at 300^{o} K) in active mode consumes 36 mA under 3.3 V [50] . Therefore, the power consumed in active mode is 118.8 mW. This power consumption can be used to give a rough estimate of the energy consumed during computations. The sensor node N takes about 0.43 ms; therefore the energy consumed by N is $0.43 \times 118.8/1000 = 0.051$ mJ. On the other hand, the controller node CRN takes about 0.49 ms; therefore the energy consumed by CRN is $0.49 \times 118.8/1000 = 0.058$ mJ. These computation results are summarized in Table 4.11.

Table 4.11: Computation time and energy consumption of our scheme on 32- bit Cortex-M3 micro-controller at 72 MHz

Node	Computation time (ms)	Energy consumption (mJ)
N	0.43	0.051
CRN	0.49	0.058

4.3.4 Communication overheads

In the transmission (N $\rightarrow$ CRN), N sends the tuple, TID_N, $\mathrm{ID_{MFR}}$, PoF, Y_N , $S2_N$, $S3_N$, T_N, H_F. Assume $|T_N|$ = 32 bits, therefore, the size of this tuple is $6 \times 160 + 32 + 16 = 1008$ bits. In the transmission (CRN $\rightarrow$ N), CRN sends the tuple, $\mathrm{ID_{CRN}}$, $\mathrm{G_{CRN}}$, $\mathrm{T_{CRN}}$, CH_F. Assume $|T_{\mathrm{CRN}}|$ = 32 bits, hence, the size of this tuple is $2 \times 160 + 32 + 16 = 368$ bits. The communication overheads of our scheme is shown in Table 4.12.

Table 4.12: Communication cost of our scheme

Communication between nodes	Communication cost
N $\rightarrow$ CRN	1008 bits
CRN $\rightarrow$ N	368 bits

Transmission usually consumes more energy than computation does, and the cost of transmitting 1 bit depends on the hardware.

Table 4.13: Comparison of communication cost between the proposed scheme and other related schemes

Authentication scheme	Communication cost number of messages	Communication cost total number of bits
Ukil *et al.* [56]	4 Messages	7600
Shivraj *et al.* [48]	4 Messages	Unspecified
Moosavi *et al.* [40]	15 Messages	9520
Risalat *et al.* [45]	7 Messages	Unspecified
Proposed scheme	**2 Messages**	**1376**

In Table 4.13, we present a comparison between our proposed scheme and other related schemes in terms of communication cost based on transmissions in both directions between IoT node and gateway. We use the number of exchanged messages for a successful authentication as the key of the communication cost comparison. As shown in the table, our scheme requires two messages for successful mutual authentication, while the most similar

schemes to ours require higher number of messages and thus consume more energy than our proposed scheme. It can be observed that our proposed scheme is comparatively more cost-efficient.

4.4 Protocol Security Validation

In this section, we evaluate the security of our proposed protocol using three different approaches: informal security analysis, formal validation using theorem proving based on BAN logic, and formal model checking and simulation using AVISPA.

4.4.1 Security Analysis

In this section, we discuss various known attacks, and we elaborate on how our protocol successfully resists and stops these attacks.

4.4.1.1 Replay attack

A replay attack is defeated using timestamps. The timestamp T_N is generated by the sender node and then inserted in the transmitted message so that it ensures it cannot be replaced by the attacker. Furthermore, the attacker cannot find and reach the temporary secret key TK_N because she is totally unaware of the real identity of sensor node ID_N and the temporary secret parameter R_N, and thus any replay attack attempts will fail. Moreover, the concept of the cumulative-hash-chain, which was discussed in Subsection 4.1.3, will prevent any replay attack attempts. Hence, our protocol protects against the replay attack.

4.4.1.2 Eavesdropping attack

During the authentication phase, the attacker can easily intercept and record all parameters (TID_N, PoF, $S2_N$, $S3_N$, T_N, H_F) of the message in transit between N and CRN since they are all sent in plaintext (unencrypted). These parameters are useless to the attacker for some reasons. First, the temporary identity of sensor node TID_N changes in every session. Second, Y_N, $S2_N$, $S3_N$ parameters are constituted of secure parameters that are out of the attacker reach. Moreover, H_F is protected using the one-way hash function of the message parameters and the temporary secret key TK_N. In order for the attacker to know the temporary secret key TK_N, she needs to find the real identity of the IoT node ID_N and the temporary secret parameter R_N, but these parameters, as aforementioned, are protected and out of the attacker reach. Therefore, our protocol protects against the eavesdropping attack.

4.4.1.3 Impersonation attack

In this attack scenario, the attacker impersonates IoT node N and sends (TID_N, PoF, $S2_N$, $S3_N$, T_N, H_F), which might be previously eavesdropped during the transmission between N and CRN, to CRN. This attack will fail for certain reasons. First, the temporary identity of IoT node TID_N changes in every session. Second, H_F is protected using the one-way hash function of the message parameters, which include the timestamp T_N, and the temporary secret key TK_N. Thus, our protocol protects against the impersonation attack.

4.4.1.4 Man-in-the-middle attack

As discussed in the replay attack, eavesdropping and impersonation attacks, the man-in-the-middle attack is defeated due to the fact that the real identity of IoT node ID_N, the temporary

secret parameter R_N and the temporary secret key TK_N are totally unknown to the attacker. Moreover, the cumulative-hash-chain value H_F is protected using the temporary secret key TK_N. Therefore, the man in the middle attack is prevented.

4.4.1.5 Attack against the temporary secret key

The temporary secret key TK_N is computed as $TK_N = \mathrm{h}(ID_N, R_N)$ and protected using one-way hash function. As we said before, ID_N is assumed to be secured and R_N changes in every session. The TK_N is never sent during the authentication phase. Instead, it is computed at the receiving node, and thus it is secure and totally beyond the attacker's reach.

4.4.1.6 Node stolen database attack

If any IoT node is compromised, the other nodes will not be affected since the node does not store any secret keys of other nodes in its memory. In addition, the attacker is unable to obtain the master secret key of the controller K_{CRN} because it is only used in the registration phase to compute $S1_N$, $S2_N$ and $S3_N$, and then it is destroyed. Moreover, IoT node stores the real identities of the other IoT nodes that it is allowed to communicate with in order to ask the CRN to open sessions with them, but these identities are useless for the attacker because the attacker does not know how to use them to establish any session.

4.4.1.7 Forward/backward security

The forward/backward security aims to make sure that any past or future sessions keys will not be affected when any temporary session key TK_N is exposed. This forward/backward security service is achieved by our authentication scheme. Recall that our temporary session key is computed as $TK_N = \mathrm{h}(ID_N, R_N)$. ID_N and R_N are protected with the one-way hash

function h. ID_N is assumed to be secure and R_N dynamically changes with the sessions. Hence, the forward/backward security is achieved by our authentication scheme.

4.4.1.8 Session key Guessing Attack

The temporary session key TK_N is generated by the IoT node using a secure ID_N and a random secret parameter R_N. Since this temporary key is dependent on secure and random parameters, the attacker cannot obtain it from the protocol. The probability of guessing is so negligible that the attacker will fail to guess the correct parameters/ temporary session key given that this TK_N changes in every session.

4.4.1.9 Privacy breach

Our proposed scheme mitigates privacy breach by ensuring unlinkability and anonymity. These properties prevent any adversary from obtaining the real identity ID_N of any IoT node N and from linking any session to any other session of the same IoT node N. Consider the tuple TID_N, ID_{MFR}, PoF, Y_N , $S2_N$, $S3_N$, T_N, H_F from N to CRN . We have $TID_N = h(ID_N \oplus T_N, R_N)$, and since R_N is a fresh random number chosen by N in each session, the adversary cannot link any two different TID_N 's to the same N. The parameter $Y_N = X_N \oplus R_N$ is also fresh each session. The two parameters, $S2_N$ and $S3_N$, are constituted of secure parameters, such as the controller node master key K_{CRN}, that are out of the adversary's reach. The parameter H_F is protected using the one-way hash function of the message parameters, which include the timestamp T_N, and the temporary secret key TK_N. Now consider the tuple ID_{CRN}, G_{CRN}, T_{CRN}, H_F from CRN to N. The parameter $G_{\text{CRN}} = h(X_N, R_N, ID_N, TK_N, T_{\text{CRN}})$ is also fresh each session since R_N, TK_N, T_{CRN} are fresh. Our scheme ensures that the secure parameters, such as R_N and X_N, cannot be

discovered by the adversary in order to learn some secure fixed parameters. Therefore, the communication parameters are fresh, random, and independent in every established session. Moreover, the adversary is unable to relate two or more sessions to the same node N; thus, our scheme achieves anonymity of the nodes and unlinkability of the conducted sessions.

Another potential privacy threat is the fact that IoT nodes' data at rest stored in databases may be exploited by leveraging and gaining knowledge of the metadata. One of the renowned attacks that could target the metadata is inference attack [38], which consists of analyzing the data to illicitly obtain knowledge about the IoT node's sensitive data. In our future work, we will investigate how to overcome such privacy threat by adopting and adapting for IoT environment the privacy preservation approach proposed by Waqar et al. [60] for cloud environments. The approach consists of creating users' data privacy-preserving schema for database by separating the data in different privacy groups (exclusively private, partially private and non-private), and partitioning the database tables horizontally and vertically to ensure database normalization and prevent data leakage.

4.4.2 Formal proof based on BAN logic

4.4.2.1 Goals of the analysis of our authentication scheme

In this section, we define the main goals of the analysis of our authentication scheme as follows:

- G1: CRN believes N believes TK_N is a secure shared parameter between N and CRN.

$$CRN| \equiv N| \equiv (N \overset{TK_N}{\longleftrightarrow} CRN)$$

- G2: CRN believes TK_N is a secure shared parameter between N and CRN.

$$CRN| \equiv (N \overset{TK_N}{\longleftrightarrow} CRN)$$

- G3: N believes CRN believes ID_N is a secure shared parameter between N and CRN.

$$N| \equiv CRN| \equiv (N \overset{G_{\text{CRN}}}{\longleftrightarrow} CRN)$$

- G4: N believes ID_N is a secure shared parameter between N and CRN.

$$N| \equiv (N \overset{G_{\text{CRN}}}{\longleftrightarrow} CRN)$$

4.4.2.2 Messages transferred in the authentication

The idealized messages that are exchanged in the authentication phase between an IoT node N and the controller CRN are listed below:

- M1: N $\rightarrow$ CRN: $\langle$ N $\overset{TK_N}{\longleftrightarrow}$ CRN, R_N, X_N, T_N $\rangle_{\text{N} \overset{ID_N}{\longleftrightarrow} \text{CRN}}$
- M2: CRN $\rightarrow$ N: $\langle$ N $\overset{TK_N}{\longleftrightarrow}$ CRN, R_N, X_N, T_{CRN}, N $\overset{G_{\text{CRN}}}{\longleftrightarrow}$ CRN $\rangle_{\text{N} \overset{ID_N}{\longleftrightarrow} \text{CRN}}$

4.4.2.3 Introductory assumptions

The fundamental assumptions of our authentication scheme are as follows:

- A1: CRN believes ID_N is a secure shared parameter between N and CRN.

$$CRN| \equiv (N \overset{ID_N}{\longleftrightarrow} CRN)$$

- A2: CRN believes T_N is fresh.

$$CRN| \equiv \#(T_N)$$

- A3: CRN believes N believes ID_N is a secure shared parameter between N and CRN.

$$CRN| \equiv N| \equiv (N \overset{ID_N}{\longleftrightarrow} CRN)$$

- A4: N believes ID_N is a secure shared parameter between N and CRN.

$$N| \equiv (N \overset{ID_N}{\longleftrightarrow} CRN)$$

- A5: N believes R_N is fresh.

$$N| \equiv \#(R_N)$$

- A6: N believes CRN has jurisdiction over G_{CRN} which is a secure shared parameter between N and CRN.

$$N| \equiv CRN| \Rightarrow (N \overset{G_{\text{CRN}}}{\longleftrightarrow} CRN)$$

4.4.2.4 Analysis of our authentication scheme

We now start analyzing our authentication scheme in order to prove that our scheme achieves mutual authentication between N and CRN.

S1: From assumption A1 and message M1, and by applying the message meaning rule, we

derive:

$$\frac{CRN|\equiv(N\overset{ID_N}{\longleftrightarrow}CRN),CRN\triangleleft\langle N\overset{TK_N}{\longleftrightarrow}CRN,R_N,X_N,T_N\rangle_{\mathrm{N}} \quad N\overset{ID_N}{\longleftrightarrow}\mathrm{CRN}}{CRN|\equiv N\sim(N\overset{TK_N}{\longleftrightarrow}CRN,R_N,X_N,T_N)}$$

S2: From assumption A2 and by applying the freshness rule, we derive:

$$\frac{CRN|\equiv\#(TN)}{CRN|\equiv\#(N\overset{TK_N}{\longleftrightarrow}CRN,R_N,X_N,T_N)}$$

S3: From derivations S1 and S2, and by applying the nonce verification rule, we derive:

$$\frac{CRN|\equiv\#(N\overset{TK_N}{\longleftrightarrow}CRN,R_N,X_N,T_N),CRN|\equiv N|\sim(N\overset{TK_N}{\longleftrightarrow}CRN,R_N,X_N,T_N)}{CRN|\equiv N\equiv(N\overset{TK_N}{\longleftrightarrow}CRN,R_N,X_N,T_N)}$$

S4: From derivation S3 and by applying the belief rule, we derive:

$$\frac{CRN|\equiv N\equiv(N\overset{TK_N}{\longleftrightarrow}CRN,R_N,X_N,T_N)}{CRN|\equiv N\equiv(N\overset{TK_N}{\longleftrightarrow}CRN)}\ (\text{Goal G1})$$

S5: From assumption A3 and derivation S4, by applying the jurisdiction rule, we derive:

$$\frac{CRN|\equiv N\Rightarrow(N\overset{TK_N}{\longleftrightarrow}CRN),CRN|\equiv N\equiv(N\overset{TK_N}{\longleftrightarrow}CRN)}{CRN|\equiv(N\overset{TK_N}{\longleftrightarrow}CRN)}\ (\text{Goal G2})$$

S6: From assumption A4 and message M2, and by applying the message meaning rule, we derive:

$$\frac{N|\equiv(N\overset{ID_N}{\longleftrightarrow}CRN),N\triangleleft\langle TK_N,R_N,X_N,T_{\mathrm{CRN}},N\overset{G_{\mathrm{CRN}}}{\longleftrightarrow}CRN\rangle_{\mathrm{N}} \quad N\overset{ID_N}{\longleftrightarrow}\mathrm{CRN}}{N|\equiv CRN|\sim(TK_N,R_N,X_N,T_{\mathrm{CRN}},N\overset{G_{\mathrm{CRN}}}{\longleftrightarrow}CRN)}$$

S7: From assumption A5 and by applying the freshness rule, we derive:

$$\frac{N|\equiv\#(R_N)}{N|\equiv\#(TK_N,R_N,X_N,T_{\mathrm{CRN}},N\overset{G_{\mathrm{CRN}}}{\longleftrightarrow}CRN)}$$

S8: From derivations S6 and S7, by applying the nonce verification rule, we derive:

$$\frac{N|\equiv\#(TK_N,R_N,X_N,T_{\mathrm{CRN}},N\overset{G_{\mathrm{CRN}}}{\longleftrightarrow}CRN,N\overset{ID_N}{\longleftrightarrow}CRN,N|\equiv CRN|\sim(TK_N,R_N,X_N,T_{\mathrm{CRN}},N\overset{G_{\mathrm{CRN}}}{\longleftrightarrow}CRN))}{N|\equiv CRN|\equiv(TK_N,R_N,X_N,T_{\mathrm{CRN}},N\overset{G_{\mathrm{CRN}}}{\longleftrightarrow}CRN)}$$

S9: From derivation S8 and by applying the belief rule, we derive:

$$\frac{N|\equiv CRN|\equiv(TK_N,R_N,X_N,T_{\text{CRN}},N\overset{G_{\text{CRN}}}{\longleftrightarrow}CRN)}{N|\equiv CRN|\equiv(N\overset{G_{\text{CRN}}}{\longleftrightarrow}CRN)} \text{ (Goal G3)}$$

S10: From assumption A6 and derivation S9, by applying the jurisdiction rule:

$$\frac{N|\equiv CRN\Rightarrow(N\overset{G_{\text{CRN}}}{\longleftrightarrow}CRN),N|\equiv CRN\equiv(N\overset{G_{\text{CRN}}}{\longleftrightarrow}CRN)}{N|\equiv(N\overset{G_{\text{CRN}}}{\longleftrightarrow}CRN)} \text{ (Goal G4)}$$

Hence, our authentication scheme achieves mutual authentication and key agreement between N and CRN.

4.4.3 Simulation based on AVISPA tool

In this section, we show the details and results of the simulation of our protocol using the AVISPA tool.

4.4.3.1 Simulation details

Table 4.14 shows the abstract notations used to describe our authentication scheme and the corresponding AVISPA HLPSL scripting variables/functions.

We begin by determining the security goals for the simulation. We want to ensure the secrecy of a number of values, namely, KCRN, Kmfr, VDN, CaN, and IDN. Furthermore, we want to be certain that CRN and N authenticate each other successfully. Next, we write the HLPSL script for our authentication scheme. In our authentication scheme, six roles are defined, namely role_SA , which is played by the system administrator SA, role_N, which is played by the IoT node N, role_CRN, which is played by the controller node CRN, role_ MFR which is played by the manufacturer fog node, session, where the session role and all its declarations are defined, and environment, which instantiates all agents, variables,

Table 4.14: Abstract Notation and AVISPA HLPSL Scripting Variables/Functions for Protocol Specification

Scheme notations/functions	HLPSL syntax variables/function
SA	SA
N	N
CRN	CRN
MFR	MFR
ID_N	IDN
TID_N	TIDN
K_{CRN}	KCRN
K_{MFR}	kmfr
TK_N	TKN
R_N	RN
$S1_N$, $S2_N$, $S3_N$	S1N, S2N, S3N
PoF	PoF
X_N , Y_N	XN, YN
G_{CRN}	GCRN
VD_N	VDN
Ca_N	CaN
T_N	TN
T_{CRN}	TCRN
T_{MFR}	TMFR
h(·)	H(·)
$\oplus$	XOR

and functions. The channel between SA and N for the registration phase is secured using a symmetric key (SK) shared between SA and N. We assume this key is beyond the reach of an intruder.

```
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
role role_SA
(SA:agent,N:agent,CRN:agent,MFR:agent,SK:symmetric_key,KCRN,VDN,IDN,CaN:text,H:hash_func,Snd,Rcv:channel(dy))
played_by SA

def=
        local
                State:nat,
                S1N,S2N,S3N:text

                const secKCRN,secKmfr,secIDN,secCaN,secVDN,controller_node:protocol_id
        init
                State := 0
        transition

                1. State=0 /\ Rcv(start) =|> State':=1 /\ S1N':= xor(IDN,H(KCRN.IDN)) /\ S2N':= xor(xor
(KCRN,S1N'),VDN)/\S3N':=xor(CaN,H(KCRN.S2N')) /\ Snd({S1N'.S2N'.S3N'}_SK)

end role
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
```

Figure 4.8: HLPSL code for role System Administrator played by SA

Fig. 4.8 shows the role of the system administrator played by SA. The knowledge of the SA initially contains all agents in the protocol (SA, node (N), controller node (CRN) and manufacturer fog node (MFR)), the controller node secret key (KCRN), the node identity (IDN) which should be secured, the symmetric key (SK) between SA and N, the node virtual domain id (VDN), the node capability id (CaN), the hash function H(·) and the send/receive channels Snd/Rcv. The (dy) notation denotes that the channels are following the Dolev-Yao model. The keyword 'Played_by SA' denotes that the role admin is played by the agent SA. All local variables utilized in this role are defined under the local section. The variable State represents the local state index of this role and is declared as nat initialized to zero. S1N, S2N, and S3N are the local variables in SA role. These three variables will be calculated by SA for N. At state 0, the SA receives a start message "Rcv(start)" as a signal

to begin the protocol run. Using KCRN, VDN, CaN and IDN, the SA computes S1N':= xor(IDN,H(KCRN.IDN)), S2N':=xor(xor(KCRN,S1N'),VDN), S3N':=xor(CaN,H(KCRN. S2N')) as explained in our protocol scheme. Then, the SA sends the registration message (S1N'.S2N'.S3N') encrypted using the symmetric key SK. This message will arrive at node N role and will be seen as a received message in the node's role. The "end role" at the end of the SA role denotes the end of the role System administrator played by SA. It is worth mentioning that if any variable is marked as primed ('), it means that it is locally computed by the agent. Also, the symbol /\ denotes conjunction (logical AND). Notice that { }_ denotes the encryption where the encryption key is identified following an _ .

```
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
role role_N
(SA:agent,N:agent,CRN:agent,MFR:agent,SK:symmetric_key,IDN,IDCRN,IDMFR,Kmfr:text,H:hash_func,Snd,Rcv:channel(dy))

played_by N

def=
	local
		State:nat,
		S1N,S2N,S3N,TN,XN,RN,YN,TCRN,CaN,VDN,KCRN:text, TKN,TIDN:hash(text.text),GCRN:hash
(text.text.message.text),POF:message

const secKCRN,secKmfr,secIDN,secCaN,secVDN,secTKN,controller_node:protocol_id
	init
		State := 0
	transition
		2. State=0 /\ Rcv({S1N'.S2N'.S3N'}_SK) =|> State':=1 /\TN':=new()/\ XN':=xor(S1N',IDN)/\RN':=new
()/\YN':=xor(XN',RN')/\TKN':= H(IDN.RN')/\TIDN':= H(xor(IDN,TN').RN')/\POF':= xor(H(H(xor
(IDMFR,Kmfr).IDCRN).TN'),IDN) /\secret(IDN,secIDN,{SA,N,CRN,MFR}) /\ secret(KCRN,secKCRN,{SA,CRN}) /\ secret
(Kmfr,secKmfr,{N,MFR})/\secret (CaN,secCaN,{SA,CRN})/\secret (VDN,secVDN,{SA,CRN})/\ Snd
(IDCRN.IDMFR.POF'.TIDN'.S2N'.S3N'.YN'.TN')/\secret (TKN,secTKN,{N,CRN})
		6.State=1 /\ Rcv(GCRN'.TCRN')=|> State':=2 /\GCRN':=H(XN.IDN.TKN.TCRN')/\request
(N,CRN,controller_node,GCRN)/\secret(KCRN,secKCRN,{SA,CRN}) /\ secret (Kmfr,secKmfr,{N,MFR})/\secret (CaN,secCaN,
{SA,CRN})/\secret (VDN,secVDN,{SA,CRN})/\secret (TKN,secTKN,{N,CRN})/\secret(IDN,secIDN,{SA,N,CRN,MFR})

end role
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
```

Figure 4.9: HLPSL code for role IoT Node played by N

Fig. 4.9 shows the role of the Node N played by N. Node N is aware of all agents in the protocol (SA, N, CRN and MFR), Kmfr, her identity IDN which should be secured, the symmetric key (SK) between SA and N, controller node identity (ID_{CRN}) and manufacturer fog node identity (ID_{MFR}), the hash function H(·) and the send/receive channels Snd/Rcv. All local variables are declared under the local section. At the first transition "State 0", N

receives the registration message (S1N'.S2N'.S3N') on the Rcv channel from SA encrypted using the symmetric key SK. Note that the symmetric key SK is known to N, so the SPAN simulator presumes that N has learned the contents of the message. N is assumed to securely store these registration parameters in its memory. At that point, N knows ID_N, ID_{CRN}, ID_{MFR}, $S1_N$, $S2_N$, $S3_N$, so N can start the authentication process with CRN at any time. N creates and computes some authentication parameters in order to authenticate with the CRN. It picks fresh values for nonces TN and RN by assigning them to new(), which intuitively denotes that the values are randomly generated.

The computations for PoF, TIDN, XN and YN follow the description of our scheme. N sends (IDCRN.IDMFR.PoF'.TIDN' .S1N'.S2N'.S3N'.YN'.TN') to CRN. At the second transition "State 1", N receives the (GCRN'.TCRN') message from CRN. The computations of GCRN follow the description of our scheme. At this transition, if the GCRN appears as expected by N, the authentication has been done successfully between N and CRN. The "end role" at the end of the N role denotes the end of the role Node played by SA.

```
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
role role_CRN(SA:agent,N:agent,CRN:agent,MFR:agent,Kmfr_pub, KCRN_pub:
public_key,KCRN,IDCRN:text,H:hash_func,Snd,Rcv:channel(dy))

played_by CRN

def=
        local
                State:nat,
  S1N,S2N,S3N,TN,YN,IDN,XN,RN,TCRN,VDN,CaN,Kmfr,IDMFR,TMFR:text,TKN,TIDN:hash(text.text),GCRN:hash
(text.text.message.text),POF:message

const secKCRN,secKmfr,secIDN,secCaN,secVDN,secTKN,controller_node,crn_mfr_pof:protocol_id
      init
                State := 0
        transition
                3. State=0 /\ Rcv(IDCRN'.IDMFR'.POF'.TIDN'.S2N'.S3N'.YN'.TN') =|> State':=1 /\TCRN':=new()/\
secret(KCRN,secKCRN,{SA,CRN}) /\ secret (Kmfr,secKmfr,{N,MFR})/\secret (CaN,secCaN,{SA,CRN})/\secret (VDN,secVDN,
{SA,CRN})/\secret(IDN,secIDN,{SA,N,CRN,MFR})/\secret (TKN,secTKN,{N,CRN})/\Snd({IDCRN.POF'.TN.TCRN'}_Kmfr_pub)/
\witness(CRN,MFR,crn_mfr_pof,POF)

                5. State=1 /\ Rcv({IDMFR'.IDN'.TMFR'}_KCRN_pub)=|> State':=2 /\ S1N':=xor(IDN',H(KCRN.IDN'))/
\XN':=xor(S1N,IDN')/\RN':=xor(XN',YN)/\TIDN':= H(xor(IDN',TN).RN')/\TKN':= H(IDN'.RN')/\VDN':=xor(xor
(KCRN,S1N),S2N)/\CaN':=xor(S3N,H(KCRN.S2N))/\TCRN':=new()/\GCRN':=H(XN'.IDN'.TKN'.TCRN')/\ secret(KCRN,secKCRN,
{SA,CRN}) /\secret(IDN,secIDN,{SA,N,CRN,MFR}) /\secret (Kmfr,secKmfr,{N,MFR})/\secret (CaN,secCaN,{SA,CRN})/
\secret (VDN,secVDN,{SA,CRN})/\Snd(GCRN'.TCRN')/\witness(CRN,N,controller_node,GCRN)/\secret (TKN,secTKN,{N,CRN})

end role
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
```

Figure 4.10: HLPSL code for role Controller played by CRN

Fig. 4.10 shows the role of the controller node played by CRN. Node CRN is aware of all agents in the protocol (SA, N, CRN and MFR), her identity IDCRN, her secret key Kcrn, the public key of the manufacturer fog node Kmfr_pub, her own public key Kcrn_pub, the hash function H(·) and the send/receive channels Snd/Rcv. It is worth noticing that CRN does not know VDN, CaN or IDN. VDN, CaN and IDN parameters will be locally computed later using the received authentication parameters from N and MFR nodes. Moreover, CRN does not know the symmetric key SK because it does not need to communicate with SA using a secure channel. All local variables are defined under the local section. At the first transition "State 0", CRN receives the (IDCRN'.IDMFR'.PoF'.TIDN'.S2N'.S3N'.YN'.TN') message which was sent by N in role node played by N. As said previously, CRN does not know the IDN, so it cannot unfold the authentication parameters. Therefore, it has to communicate with the MFR in order to get IDN. At the second transition "State 1", CRN forwards {IDCRN.PoF'.TN'.TCRN}_Kmfr_pub) message to the MFR node encrypted using MFR's public key. After while at the same transition, CRN receives the ({ IDMFR'.IDN'.TMFR' }_Kcrn_pub) message which was sent by MFR in role node played by MFR as we will see later in the next figure. At the third transition "State 2", CRN unfolds and extracts N's S1N, TKN, VDN and CaN as it learned IDN from the received message. The computations and extraction for S1N, TKN, VDN and CaN follow the description of our scheme. CRN will send back (GCRN'.TCRN') message to N in order to complete the authentication process for N. CRN generates a fresh value for nonce TN and computes GCRN. The computation for GCRN follows the description of our scheme. The "end role" at the end of the CRN role denotes the end of the role controller played by CRN.

Fig. 4.11 shows the role of the manufacturer fog node played by MFR. Node MFR knows all agents in the protocol (SA, N, CRN and MFR), her identity IDMFR, and Node

```
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
role role_MFR(SA:agent,N:agent,CRN:agent,MFR:agent,Kmfr_pub, Kcrn_pub:
public_key,IDMFR,IDN,Kmfr:text,H:hash_func,Snd,Rcv:channel(dy))

played_by MFR

def=

local
              State:nat,

  TMFR,TCRN,TN,IDCRN,IDNC:text,POF:message

const secKCRN,secKmfr,secIDN,secCaN,secVDN,controller_node,crn_mfr_pof:protocol_id
init
       State := 0
       transition
              4. State=0 /\ Rcv({IDCRN'.POF'.TN'.TCRN'}_Kmfr_pub) =|> State':=1 /\ IDN':=xor(H(H(xor
(IDMFR,Kmfr).IDCRN).TN'),POF')/\request(MFR,CRN,crn_mfr_pof,POF)/\secret(IDN,secIDN,{SA,N,CRN,MFR})/\secret
(Kmfr,secKmfr,{N,MFR})/\TMFR':=new()/\Snd({IDMFR.IDN.TMFR'}_Kcrn_pub)/\secret(IDN,secIDN,{SA,N,CRN,MFR})/\sec
(Kmfr,secKmfr,{N,MFR})

end role
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Figure 4.11: HLPSL code for role Manufacturer fog node played by MFR

N identity (IDN)), the public key of controller CRN (Kcrn_pub), her own public key (Kmfr_pub), Kmfr, the hash function H(·) and the send/receive channels Snd/Rcv. All local variables are declared under the local section. At the first transition “State 0”, MFR receives the IDCRN’.PoF’.TN’.TCRN’_Kmfr_pub message which was sent by CRN in role controller played by CRN in the previous figure. At the second transition “State 1”, MFR extracts IDN from PoF parameter that was received in the “state 0” and retrieves the IDN to be sent to the CRN. Then it generates fresh value for nonce TMFR and sends IDMFR.IDN.TMFR’_Kcrn_pub message encrypted with the controller public key. The “end role” at the end of the MFR role denotes the end of the role manufacturer fog played by MFR.

Fig. 4.12 shows the session role where all the four agents’ roles are invoked and all the session parameters are defined. First, all known constant parameters and their declarations are presented. The predefined constants which are SK, Kmfr_pub, Kcrn_pub, KCRN, Kmfr, IDN, IDCRN, IDMFR, VDN, CaN are included in this role. A send and a receive

```
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
role session (SA:agent,N:agent,CRN:agent,MFR:agent,SK:symmetric_key,Kmfr_pub, Kcrn_pub:
public_key,KCRN,Kmfr,IDN,IDCRN,IDMFR,VDN,CaN:text,H:hash_func)

def=
        local
              SCH4,RCH4,SCH3,RCH3,SCH2,RCH2,SCH1,RCH1:channel(dy)

        composition

               role_SA(SA,N,CRN,MFR,SK,KCRN,VDN,IDN,CaN,H,SCH1,RCH1) /\

               role_N(SA,N,CRN,MFR,SK,IDN,IDCRN,IDMFR,Kmfr,H,SCH2,RCH2)/\

               role_CRN(SA,N,CRN,MFR,Kmfr_pub,Kcrn_pub,KCRN,IDCRN,H,SCH3,RCH3)/\

               role_MFR(SA,N,CRN,MFR,Kmfr_pub,Kcrn_pub,Kmfr,IDMFR,IDN,H,SCH4,RCH4)

end role
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
```

Figure 4.12: HLPSL code for role session

channels are assigned to each agent under the local section. Under the composition section, the role is invoked for each agent along with its constant values and functions. The system administrator role is invoked with SA, N, CRN and MFR as agents, KCRN, IDN, VDN and CaN as predefined constants, SK as a predefined symmetric key, H as the hash function, and SCH1 and RCH1 as the send and receive channels for SA. Similarly, the remaining roles are invoked in the same manner. The “end role” at the end of the session role denotes the end of this role.

Fig. 4.13 shows the environment role. In this role, one or more sessions are instantiated. First, all constants are instantiated and defined. The constants, sa, n, crn and mfr instantiate the four agents SA, N, CRN and MFR, respectively. The sk instantiates the symmetric key SK, and Kmfr_pub and Kcrn_pub instantiate the public keys Kmfr_pub and Kcrn_pub, respectively. kcrn, kmfr, idn, idcrn, idmfr, vdn and can instantiates KCRN, Kmfr, IDN, IDCRN, IDMFR, VDN and CaN, respectively. h instantiates the hash function H. The protocol identifiers which are secKCRN, secKmfr, secIDN, secCaN, secVDN, secTKN,

```
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
role environment()

def=

       const sa,n,crn,mfr:agent, sk:symmetric_key,kmfr_pub, kcrn_pub:
public_key,kcrn,kmfr,idn,idcrn,idmfr,vdn,can:text,h:hash_func, seckcrn,secKmfr,
secidn,seccan,secvdn,sectkn,controller_node,crn_mfr_pof:protocol_id

intruder_knowledge ={sa,n,crn,mfr,h}

     composition

       session(sa,n,crn,mfr,sk,kmfr_pub, kcrn_pub,kcrn,kmfr,idn,idcrn,idmfr,vdn,can,h)

end role
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
```

Figure 4.13: HLPSL code for role environment

controller_node, and crn_mfr_pof are also instantiated and defined. In the intruder knowledge section, all relevant values that the intruder is assumed to know before the execution are provided. The attacker is assumed to know sa, n, crn and mfr; she is also assumed to know the hash h. In the composition section, the session is instantiated with sa, n, crn, mfr, sk, kmfr_pub, kcrn_pub, kcrn, kmfr, idn, idcrn, idmfr, vdn, can and h instances. The "end role" at the end of the environment role denotes the end of this role.

```
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
goal

        secrecy_of secKCRN

        secrecy_of secKmfr

        secrecy_of secIDN

        secrecy_of secCaN

        secrecy_of secVDN

        secrecy_of secTKN

        authentication_on controller_node
        authentication_on crn_mfr_pof

end goal
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
```

Figure 4.14: HLPSL code for goal

Fig. 4.14 shows the simulation goals which are declared under the "goal" keyword using the protocol identifiers declared as 'protocol_id'. The simulator is dictated to check the secrecy of KCRN at different states using secrecy_of secKCRN', 'secrecy_of secKmfr', 'secrecy_of secKIDN', 'secrecy_of secKCaN', 'secrecy_of secVDN', and 'secrecy_of secTKN'. The authentication is checked using 'authentication_on controller_node' and 'authentication_on crn_mfr_pof'. The "end role" at the end of the goal section denotes the end of this role.

```
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
environment()
|
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%
```

Figure 4.15: HLPSL code for the call of the main role environment

Fig. 4.15 finally shows the call of main role 'environment'.

4.4.3.2 Simulation results

The simulation results of our proposed protocol are based on the two widely-accepted back-end model checkers, namely, OFMC and CL-AtSe. We use a security protocol animator (SPAN) to interactively build a Message Sequence Chart (MSC) of the protocol execution from the HLPSL specification outlined earlier.

Moreover, SPAN automatically creates attacks on HLPSL specification using the well-known "Dolev-Yao" intruder model. The SPAN protocol simulation's MSC corresponding to our HLPSL specification is shown in Fig. 4.16.

Fig. 4.17 shows the protocol execution animator using SPAN software. It illustrates the full execution of our protocol under the intruder's control.

In AVISPA tool, the security properties such as authentication, integrity and secrecy

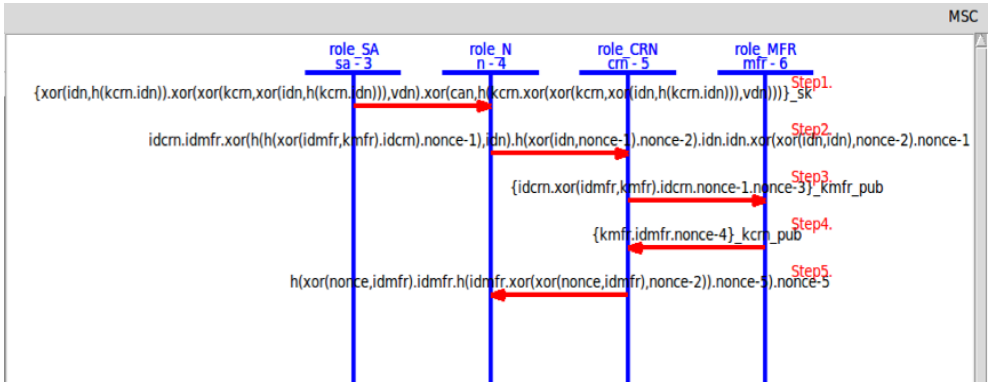

Figure 4.16: Protocol Simulation in AVISPA

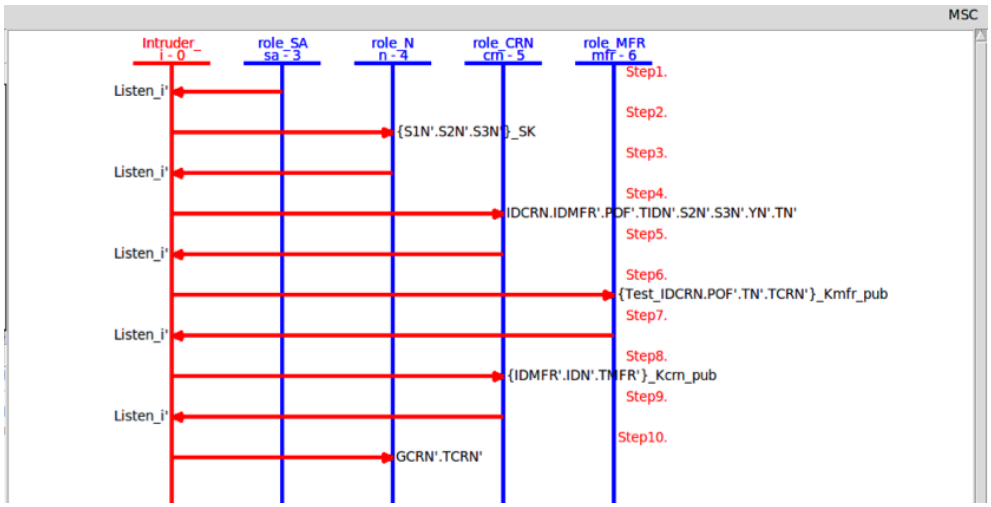

Figure 4.17: The full execution of the protocol using SPAN

are specified in a separate section. Thus, when SPAN is executed, it verifies if the protocol satisfies the specified properties. SPAN will generate the attack trace if any attack is found and it will consider the protocol unsafe. The simulation results of our protocol were achieved by the OFMC back-end checker and the CL-AtSe back-end checker. Fig 4.18 presents the CL-AtSe back-end checker report which assures that our scheme is SAFE and satisfies all the specified security goals.

Fig. 4.19 presents the OFMC back-end checker report which shows that our scheme is SAFE, thus meets the specified security goals.

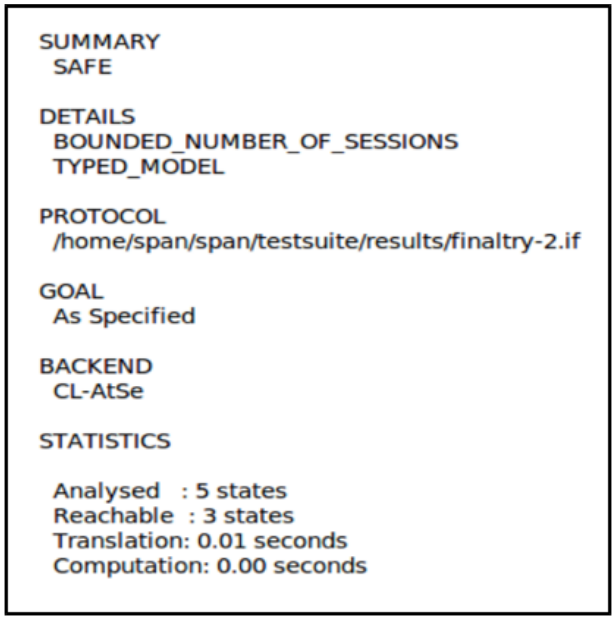

Figure 4.18: CL-AtSe summary report

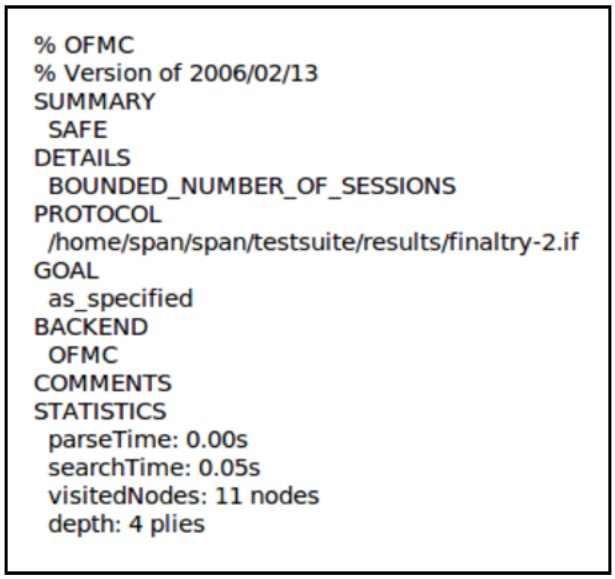

Figure 4.19: OFMC summary report

However, as our code includes XOR operation, we were not able to use the TA4SP back-end checker due to its limitation in supporting XOR operation. The SATMC model checker has reported NOT SUPPORTED.

4.5 Implementation using OMNeT++

The implementation of the proposed scheme is accomplished using the OMNeT++.

In our implementation, the top-level model is the system model, which includes the complete simulation setup for our scenario and is referred to as the "network". The system contains multiple modules implemented in C++, using the OMNeT++ simulator. These modules communicate by message passing. Modules send messages directly to their destinations. The simulation was performed for different network sizes (10, 50, and 100 nodes). Fig. 4.20 gives an overview of the simulation scenario.

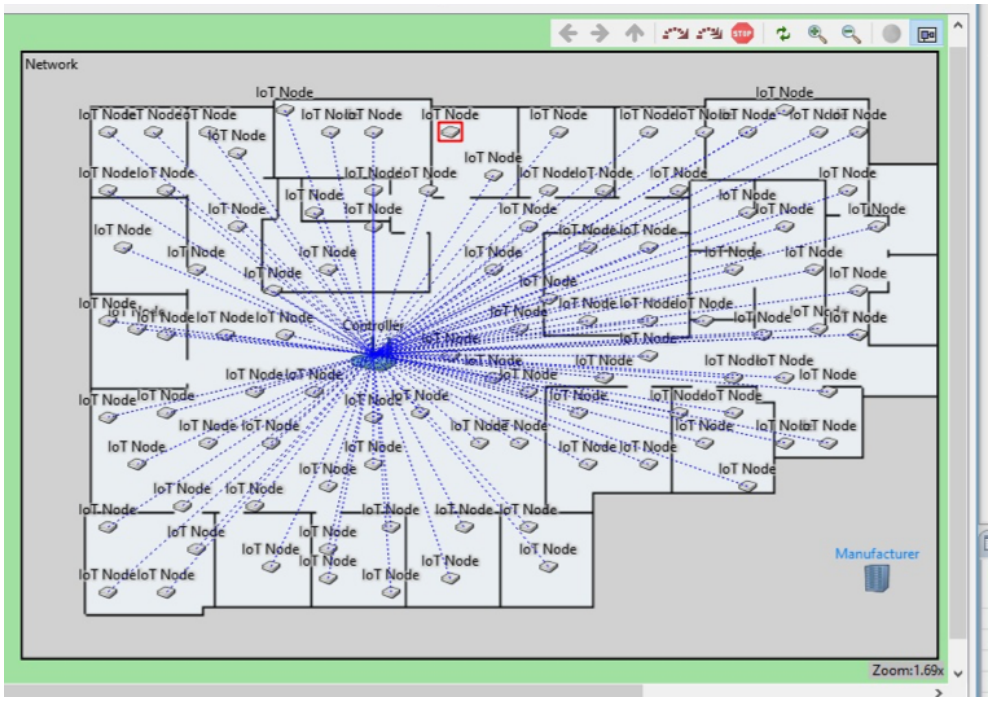

Figure 4.20: OMNeT++ simulation environment

4.5.1 System main modules

The model consists of three separate parts: the first module is the Manufacturer, which simulates the behavior of the Manufacturer (*MFR*); the second module is the Controller,

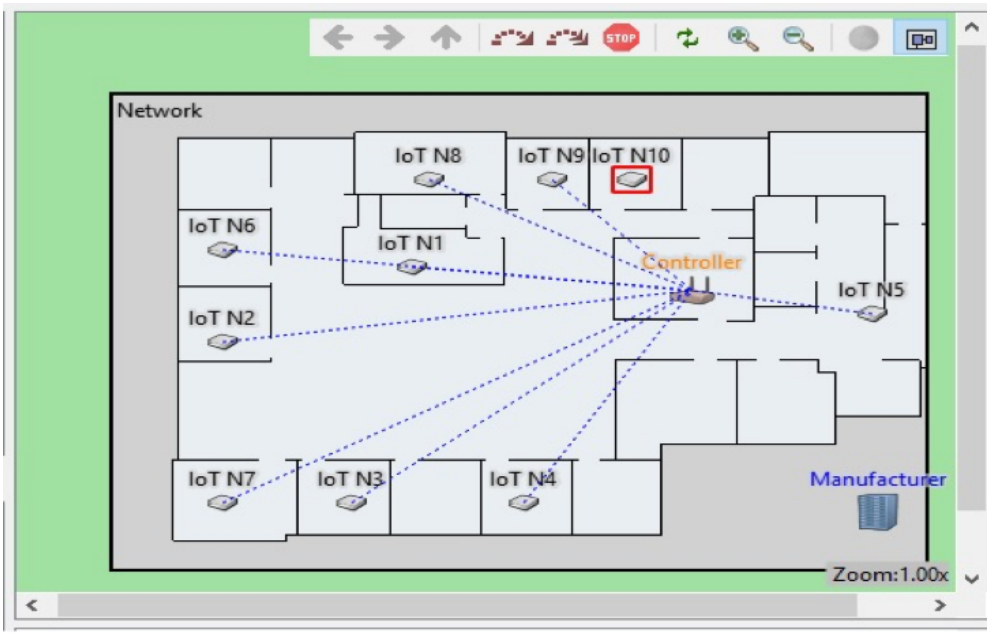

Figure 4.21: The modules of the system

which simulates the behavior of the Controller (*CRN*); and the third module is IoT *N*, which simulates IoT node (*Nx*), where 1 ≤ x ≤ n and n represents the number of IoT nodes in the network. The modules of the system are shown in Fig. 4.21.

4.5.1.1 IoT node (N) module

The IoT node module is able to effect the "knowledge"and behavior of the IoT nodes. During the initialization of the class node, it fills the node instance with its predefined parameters, including $S1_N$, $S2_N$ and $S3_N$, ID_{MFR}, ID_{CRN}, and K_{MFR}. During the handling of messages process, the authentication parameters will be generated by a pseudo-random generator in OMNeT++. The IoT node parameters will be labeled in the database to avoid reading the same IoT node parameters twice (i.e., no duplicated node in the system). Next, the authentication message is created and sent to the controller using wireless connections.

Later on, when the IoT node receives the authentication message from the controller,

it performs some operation to verify and check the authenticity of the message. Finally, it either accepts or rejects the message based on the outcome of the verification process.

4.5.1.2 Controller (CRN) module

The controller module is able to effect the "knowledge"and behavior of the controller. During the initialization of the class controller, the controller loads the predefined parameters ID_{CRN} and K_{CRN}. When the controller receives the authentication message from the IoT node module, it checks the timestamp, and encrypts some of the authentication parameters and forwards them to the Manufacturer module in order to obtain the IoT node real identity (ID_N), which allows it to process the authentication message.

Later on, when the controller receives the node *ID* from the Manufacturer, it resumes the authentication process of the IoT node and performs some operations to verify and check the authenticity of the IoT node message. Next, the controller constructs the authentication message and sends it to the IoT node.

4.5.1.3 Manufacturer (MFR) module

The Manufacturer module is able to effect the "knowledge"and behavior of the Controller. During the initialization of the class Manufacturer, the Manufacturer loads the predefined parameters ID_{MFR} and K_{MFR}. When the Manufacturer receives the authentication message from the controller module, it decrypts it and checks the timestamp, and finds out if a particular IoT node belongs to it by performing some operations. If the IoT node is authentic, the Manufacturer constructs the authentication message, which contains the IoT node (ID_N). Next, it decrypts it and sends it to the Controller (*CRN*).

Table 4.15: The various message types

Message types	Description
SMGncrn	The authentication message IoT node → Controller
SMGcrnn	The authentication message Controller → IoT node
SMGcrnmfr	The message Controller → Manufacturer
SMGmfrcrn	The message Manufacturer → Controller

4.5.2 The structure of messages

There are four message types in the simulation, namely SMGncrn, SMGcrnn, SMGcrnmfr, and SMGmfrcrn. Each message has different fields. Table 4.15 shows the various message types which were used in the simulation.

4.5.3 Message line module

The message line module is a FIFO (First In, First Out) buffer, which resides between Controller and IoT nodes. This indicates the limited ability of the controller because FIFO is usually a small memory, which operates on first in first out basis; no packets are prioritized over any other packets. The Message Queue class uses built-in OMNeT++ cQueue class to store messages. The size parameter defined in the NED file gives the number of the storable messages. It starts to drop incoming packets when the buffer is full.

4.5.4 The communication between the modules

Fig. 4.22 shows the message flow chart, which describes the communication between the various modules. A successful authentication process can be seen in the diagram.

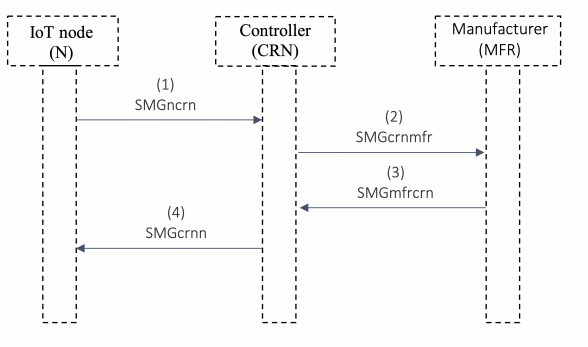

Figure 4.22: The message flow chart diagram

4.5.5 Implementation of attacks

We handle some attacks which can occur in the system, such as data modification. Data modification is performed by a malicious node (attacker) to disturb the network. The attacker can replay the message (replay attack), relay and possibly alter the communications between two parties (Man in the Middle attack), and masquerade as another node (masquerade attack). We add a Hacker module which is able to effect the "knowledge"and behavior of the hacker as shown in Fig. 4.23.

4.5.5.1 Replay attack

A replay attack is a form of network attack in which a message is intercepted and maliciously repeated or delayed. In this attack, the malicious node intercepts the message between the IoT node and the controller, then it resends it to the controller as shown in Fig. 4.24.

The controller can detect the replay attack by examining the timestamp; so the controller successfully defeats this attack as shown in Fig. 4.25.

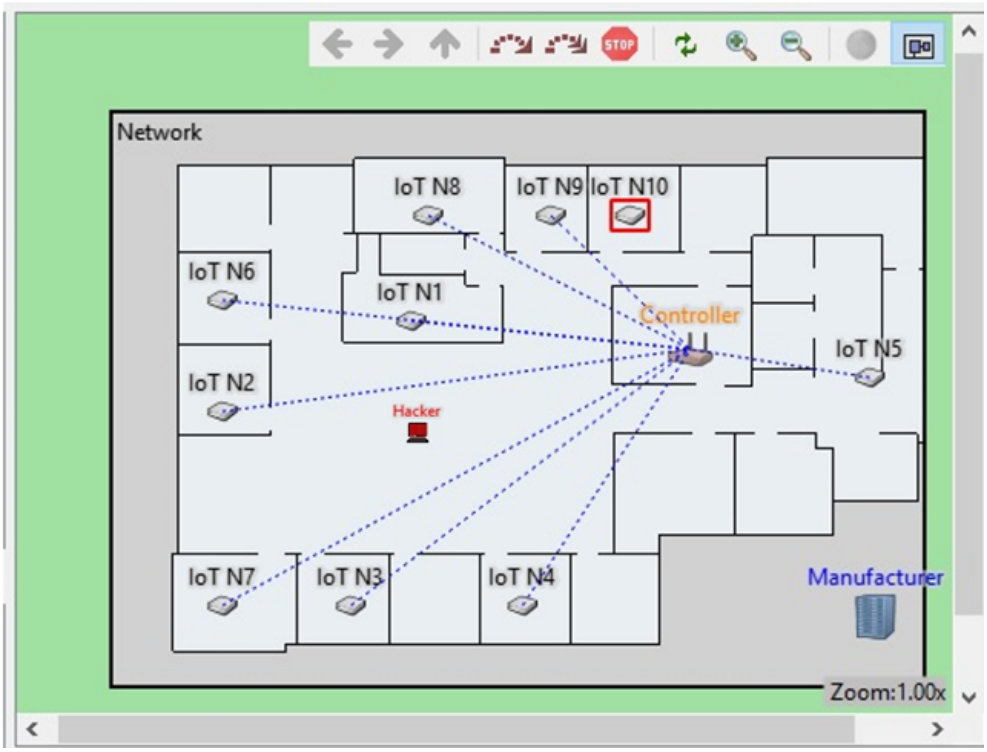

Figure 4.23: Attacker node

4.5.5.2 Man-In-The-Middle (MITM) attack

A *MITM* attack is an attack where the attacker secretly relays and probably tampers with the communications between two nodes who believe they are directly communicating with each other. This attack can be detected and defeated using timestamp and cumulative hashed value H_F, similar to replay attack.

4.5.5.3 Spoofing attack

A spoofing attack is a form of network attack in which a malicious node can successfully masquerade as another legitimate IoT node by falsifying data such as identity, to gain illicit access. In this attack, the malicious node intercepts the message between the IoT node and the controller, and then it spoofs the legitimate IoT node's *TID* to masquerade as the

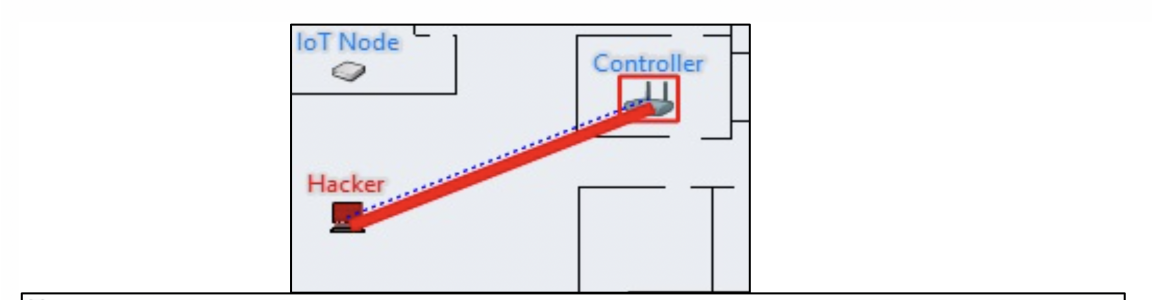

Figure 4.24: Attacker node

```
** Event #7  t=101000  Network.CRN (Controller, id=2)  on  (IoT::NCRN, id=11)
INFO (Controller)Network.CRN:  Message is corrupted due to delay (Replay attack)
```

Figure 4.25: Replay attack is defeated by the controller

authentic IoT node as shown in Fig. 4.26.

```
** Event #6  t=100000  Network.Hack (Hacker, id=4)  on  (IoT::NCRN, id=3)
INFO (Hacker)Network.Hack: a new message is being sent with spoofed TID to CRN
```

Figure 4.26: Spoofing attack

However, the controller can detect this attack since the *TID* comprises of some fresh and secure parameters, like ID_N, T_N, and R_N, which should only be generated by the legitimate IoT node as shown in Fig. 4.27.

4.5.6 Simulation Results

In this section, we present the simulation results in terms of computation cost, communication overhead, scalability, and packet loss.

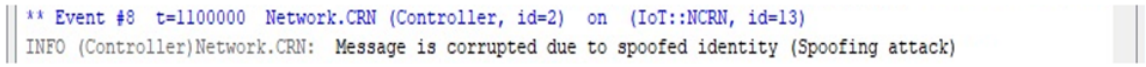

Figure 4.27: Spoofing attack is defeated by the controller

4.5.6.1 Computation cost of the IoT node (N) and the controller (CRN)

The implementation of the protocol consists of the following elementary operations: SHA-1 cryptographic hash and exclusive OR (XOR) operation. In order to provide a precise computation cost of the IoT node (*N*), the experimental data reported in [25] are utilized. Kilinc et al. [25] have estimated the running time of different cryptographic operations by using the Pairing-Based Cryptography (PBC) library version 0.5.12 [36] [35] running on a device with Ubuntu 12.04.1 32-bit operating system, CPU: 2.2 GHz, and RAM: 2.0 GB. They stated that the time of RSA signature function, block cipher encryption, block cipher decryption, selecting a random number, a hash function operation, and a hash-based message authentication code operation are approximately 3.8500 ms, 0.0046 ms, 0.0046 ms, 0.539 ms, 0.0023 ms, and 0.0046 ms, respectively. In addition, they reported that the time of XOR operation is negligible.

In this setting, the IoT node *N* performs 8 hash invocations and 5 XOR operations, which yields a total computation cost of $(7 \times T_h) + (1 \times T_{HMAC}) + (5 \times T_{xor})$. Therefore, the computation cost of *N* is $7T_h + T_{HMAC} \approx (7 \times 0.0023 \text{ ms}) + (1 \times 0.0046 \text{ ms}) = 0.0207$ ms. Furthermore, *N* needs to generate one random number, which yields a total computation cost of 0.539 ms. Hence, the total computation cost of *N* is 0.0207 ms + 0.539 ms = 0.5597 ms. On the other hand, the controller node *CRN* performs 9 hash invocations and 7 XOR operations, which yield a total computation cost of $(8 \times T_h) + (1 \times T_{HMAC}) + (7 \times T_{xor})$. The computation cost of *CRN* corresponds to $8T_h + T_{HMAC} \approx (8 \times 0.0023 \text{ ms}) + (1 \times 0.0046 \text{ ms}) = 0.023$ ms. The computation cost of IoT node (*N*) and Controller (*CRN*) are

Table 4.16: Computation cost of IoT node (N) and the Controller (CRN)

Node	**Computation cost**
IoT node (N)	0.5597 ms
Controller (CRN)	0.023 ms per node

summarized in Table 4.16.

We also present a comparison between our proposed scheme and other related schemes [56], [45], [32], [63] in terms of the computational cost of the controller (gateway/server) as shown in Table 4.17.

- In [56], the server performs 2 encryption and 2 decryption operations, 2 XOR operations, and generates one random number, which yield a total computation cost of $(2 \times T_{ENCB}) + (2 \times T_{DECB}) + (2 \times T_{xor}) + (1 \times T_{RNG})$. The computation cost of the server corresponds to $2T_{ENCB} + 2T_{DECB} + 1T_{RNG} \approx (2 \times 0.0046$ ms) + (2 × 0.0046 ms) + (1 × 0.539 ms) = 0.5574 ms.

- In [45], the server performs 1 encryption and 1 decryption operations, 4 hash invocations, and 7 XOR operations, which yield a total computation cost of $(1 \times T_{ENCB}) + (1 \times T_{DECB}) + (4 \times T_h) + (7 \times T_{xor})$. The computation cost of the server corresponds to $1T_{ENCB} + 1T_{DECB} + 4\ T_h \approx (1 \times 0.0046$ ms) + (1 × 0.0046 ms) + (5 × 0.0023 ms) = 0.0207 ms.

- In [32], the server performs 2 encryption and 2 decryption operations, 4 hash invocations, and generates one random number, which yield a total computation cost of $(3 \times T_{ENCB}) + (3 \times T_{DECB}) + (4 \times T_h) + (1 \times T_{RNG})$. The computation cost of the server corresponds to $(3 \times T_{ENCB}) + (3 \times T_{DECB}) + (4 \times T_h) + (1 \times T_{RNG}) \approx (3 \times 0.0046$ ms) + (3 × 0.0046 ms) + (4 × 0.0023 ms) + (1 × 0.539 ms) = 0.5758 ms.

- In [63], the server performs 1 decryption operations, 6 hash invocations, and generates one random number and one signature function, which yield a total computation cost of $(1 \times T_{ENCB}) + (6 \times T_h) + (6 \times T_{xor}) + (1 \times T_{RNG}) + (1 \times T_{SIGN})$.The computation cost of the server corresponds to $1T_{DECB} + 6T_h + 6T_{XOR} + 1T_{RNG} + 1T_{SIGN} \approx (1 \times 0.0046$ ms) + (6 × 0.0023 ms) + (1 × 0.539 ms) +(1 × 3.8500 ms) = 4.4074 ms.

Table 4.17 shows the comparison of the computational cost of our scheme to the aforementioned related schemes.

Table 4.17: Comparison of Computational cost of controller (gateway/server)

Node	**Ukil et al. [56]**	**Risalat et al. [45]**	**Lohachab [32]**	**Wu2 et al. [63]**	**Our scheme**
Computation cost of the controller/gateway per node in ms	0.5574	0.0207	0.5758	4.4074	0.023

As Table 4.17 shows, our scheme requires less computational cost than in [56] [32] [63] and a little more than in [45]. Our scheme requires just a little more computation cost than in [45] (approximately 0.0023 ms) because our scheme incorporates different security features and mechanisms like cumulative Keyed-hash chain, anonymity, etc. We also provide a comparison graph for the computational costs, as shown in Fig. 4.28.

4.5.6.2 Communication overhead

The communication overhead is the total number of packets transferred or transmitted from one node to another (total size in bits). In the transmission ($N \rightarrow CRN$), N sends the tuple, TID_N, ID_{MFR}, PoF, Y_N, $S2_N$, $S3_N$, T_N, H_F. Assume $|T_N|$ = 32 bits, therefore, the size of this tuple is 6 × 160 + 32 + 16 = 1008 bits. In the transmission ($CRN \rightarrow N$), CRN sends the tuple, ID_{CRN}, G_{CRN}, T_{CRN}, CH_F. Assume $|T_{CRN}|$ = 32 bits, hence, the size of this tuple is

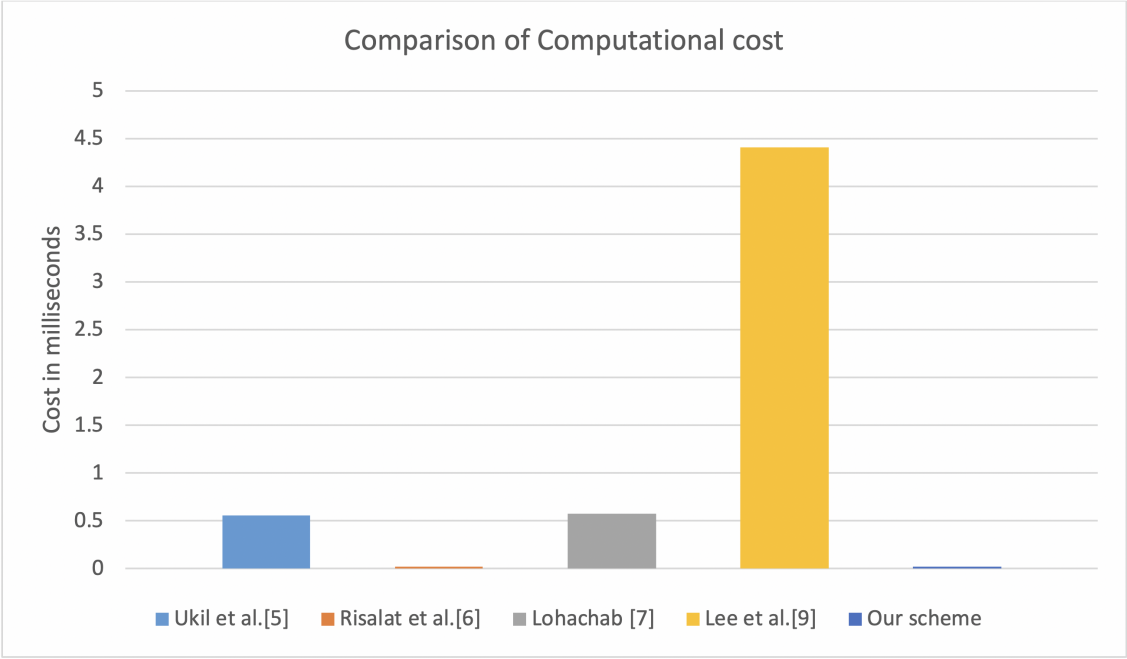

Figure 4.28: Comparison graph for computation cost

$2 \times 160 + 32 + 16 = 368$ bits. The communication overheads of our scheme is shown in Table 4.18.

Table 4.18: Communication cost in bits between *N* and *CRN*

Number of Nodes	**N → CRN**	**CRN → N**
1	1008	360
10	10080	3680
50	50400	18400
100	100800	36800
1000	1008000	368000

We separately present a graph in Fig. 4.29 that shows the communication cost between the controller and IoT node in the number of bits as the number of IoT nodes grows.

Fig. 4.29 provides the communication cost of our scheme for an increasing number of nodes. It can be noticed that as the number of IoT nodes increases, the communication cost in bits increases steadily as each successful authentication between IoT node and the

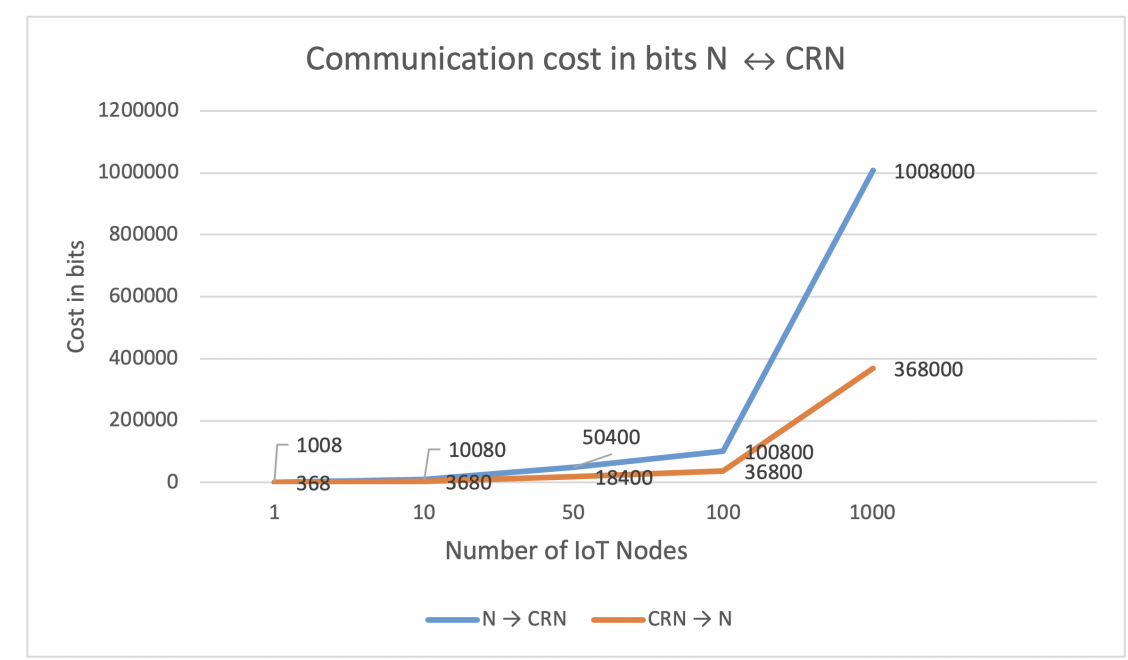

Figure 4.29: Communication cost in bits between N and CRN

controller requires total of 1008 bits (from *N* to *CRN*) + 368 bits (from *CRN* to *N*) = 1376 bits. In Table 4.19, we present a comparison between our proposed scheme and the aforementioned related schemes [56] [48] [40] [23] [45] in terms of communication cost based on transmissions in both directions between IoT node and gateway. We use the number of exchanged messages and the number of bits for a successful authentication as the key of the communication cost comparison. As shown in the table, our scheme requires two messages and a total of 1376 bits for successful mutual authentication, while the most similar schemes to ours require a higher number of messages and thus consume more energy than our proposed scheme. It can be observed that our proposed scheme is comparatively more cost-efficient.

Fig. 4.30 and Fig. 4.31 show the comparison between our proposed scheme and the related schemes in terms of number of exchanged messages and total number of bits, respectively.

Table 4.19: Comparison of communication cost between the proposed scheme and other related schemes

Authentication scheme	**Communication cost number of messages**	**Communication cost total number of bits**
Ukil et al. [56]	4 Messages	7600
Shivraj et al. [48]	4 Messages	Unspecified
Moosavi et al. [40]	15 Messages	9520
Hummen et al. [23]	15 Messages	12872
Risalat et al. [45]	7 Messages	Unspecified
Proposed scheme	2 Messages	1376

4.5.6.3 Scalability

Scalability is a critical factor for successful network operation. Easy node addition facilitates and improves network scalability. However, the addition of a new node is a challenging task due to two main reasons: (i) the new node could be a malicious node; or (ii) the new node could be a clone of a compromised node. In our proposed scheme, a new node addition is very simple because the new node addition request message is securely forwarded by the controller (*CRN*) to the Manufacturer (*MFR*). Suppose a new IoT node (*N*) needs to be added into the network; the *CRN* first securely passes some of the node *N*'s authentication parameters to *MFR* for verification of true belonging. Thereafter, *MFR* has to perform the same procedures as outlined in Section 4.2.3. By completing the aforementioned process, *N* will become a legitimate member of the Smart home network. Furthermore, as discussed in Subsection 4.5.6.1, the computation time required by the controller to authenticate a single IoT node is 0.023 ms. The computation time of the controller (*CRN*) is shown in Table 4.20.

Fig. 4.32 shows a graph that depicts the required computation time in second as the number of IoT nodes increases.

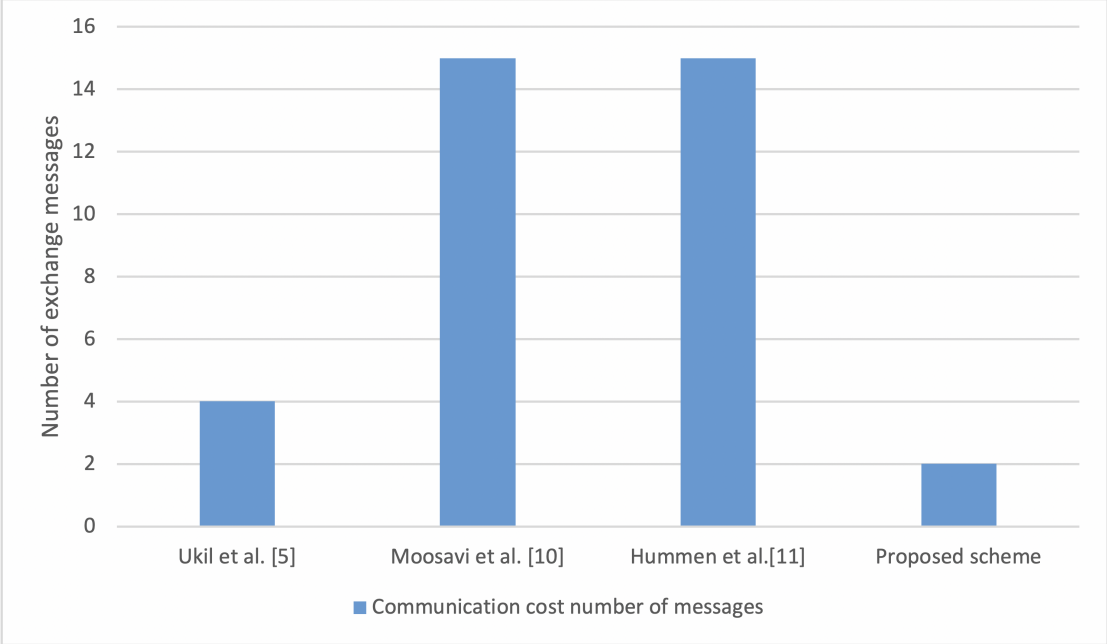

Figure 4.30: Comparsion of communication cost in terms of number of exchange messages

Table 4.20: Computation time of the controller (CRN)

Number of Nodes	Computation time in second
1	0.000023
10	0.00023
50	0.00115
100	0.0023
1000	0.023

It can be seen in Fig. 4.32, that as the number of IoT nodes increases, the required computation cost of *CRN* increases steadily. The results show that for a single IoT node, the required computation cost by *CRN* to authenticate the IoT node is 0.000023 s. For 10 IoT nodes, the required computation cost by *CRN* to authenticate these IoT nodes is 0.00023, and for 1000 IoT nodes, the required computation cost by *CRN* to authenticate these nodes is 0.023 s.

CRN is required to store the tuple ID_N, ID_{MFR}, TID_N, TK_N, ${}^{x}CH_F$, VD_N, Ca_N, in

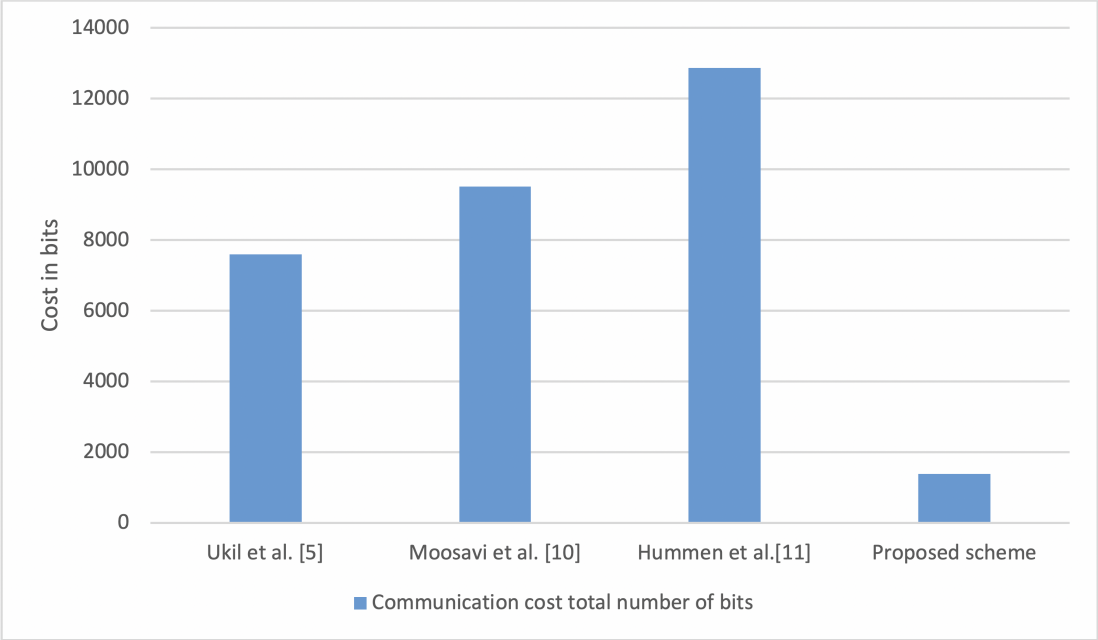

Figure 4.31: Comparison of communication cost in terms of total number of bits

addition to the *CRN* master key K_{CRN}. By applying these settings, we obtain $|TID_N|$ = $|TK_N|$ = $|^{x}CH_F|$ = 160 bits, while $|ID_N|$ = $|ID_{MFR}|$ = $|K_{CRN}|$ = $|VD_N|$ = $|Ca_N|$ = 16 bits, which as aforementioned is assumed short. Thus, the total storage required by *CRN* is (m × (5 × 16) + (160 × n) + (2 × 160)) = (80m + 160n +320) bits, where m is the number of registered IoT nodes and n represents the number of $^{x}CH_F$ values. The storage cost of the controller is shown in Table 4.21.

Table 4.21: Storage cost of the controller (CRN)

Number of Nodes	**Storage cost in bits**
1	560
10	2720
50	12320
100	24320
1000	240320

Figure 4.33 illustrates the storage cost of the controller in bits as the number of IoT

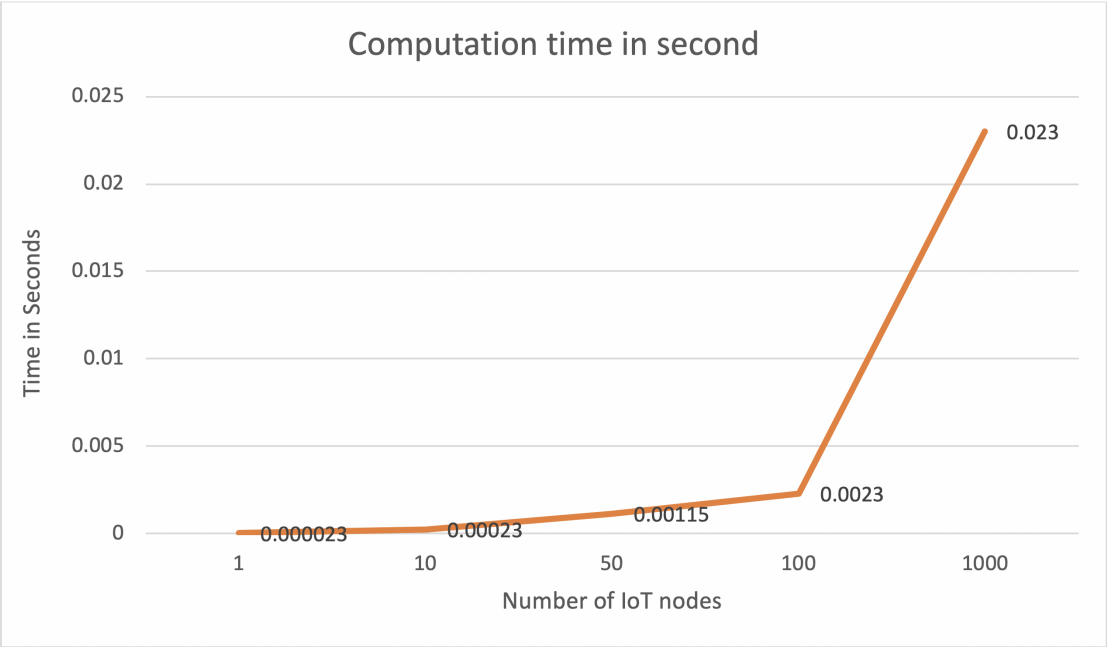

Figure 4.32: Computation time of the controller (CRN)

nodes grows.

It can be seen in Fig. 4.33, that as the number of IoT nodes increase, the required storage cost of *CRN* increases moderately. Also, we can see that for a network of 1000 IoT nodes, the required storage cost of *CRN* is approximately 240,320 bits (29.3KB). Moreover, adding a new IoT node *N* to the smart home network is easy and dynamic. When *N* wants to join in the smart home network, it uses the predefined parameters, namely *S1*, *S2*, *S3*, ID_N and a secure key shared with *MFR* (K_{MFR}), which are stored in its memory, to generate an authentication message. Next, the authentication process between *CRN* and the IoT node is executed. After the authentication process is successfully completed, *N* is authenticated and added to the smart home network. Furthermore, the information of IoT nodes are indexed by unique real identities (ID_S) by the controller *CRN*; thus, providing efficient searching.

Based on the above discussion, it can be concluded that the proposed scheme is lightweight, scalable, and efficient: the scheme only requires lightweight cryptographic

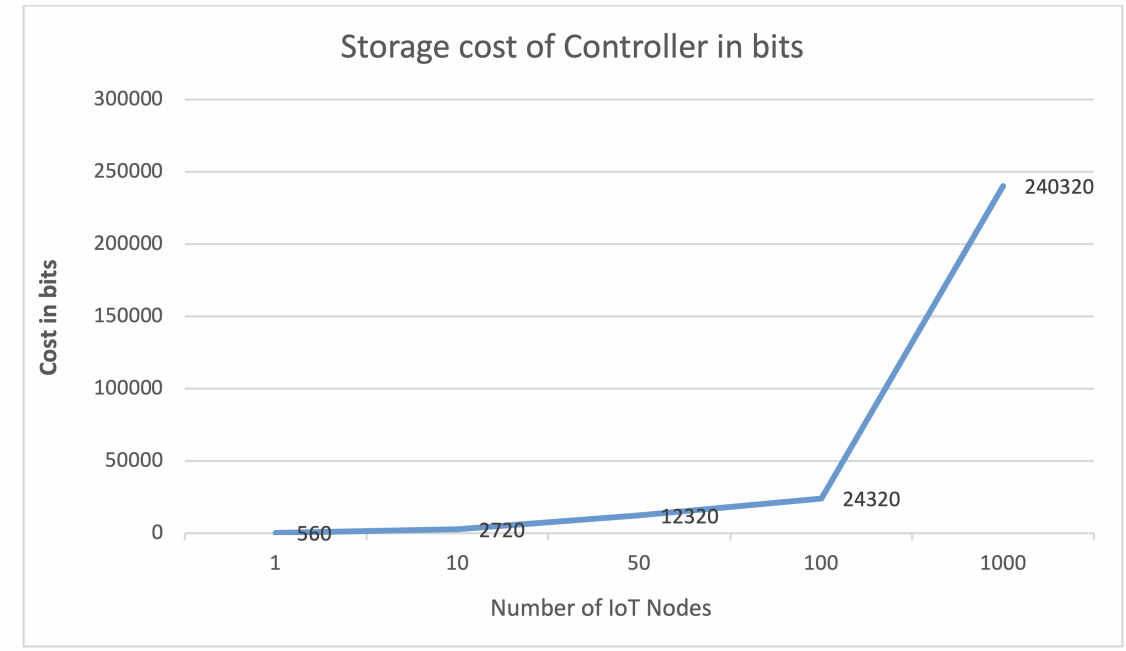

Figure 4.33: Storage cost of the controller (CRN)

operations (i.e., hash function and XOR (exclusive-or) operations). Moreover, considering the memory constraint on the IoT nodes, each IoT node is required to store only 1008 bits; and on the other hand, the *CRN* is required to store 560 bits per IoT node. Hence, the computational and storage requirements on both *N* and *CRN* are low. Hence, the scheme is scalable.

4.5.6.4 Packet loss

We measure the packet loss ratio as the average number of packets discarded divided by the total number of packets transmitted during communication. We transmit all authentication messages in interval times of 0.001 seconds. No packet loss was observed for the authentication messages that were simulated in both sides, *N* and *CRN*.

4.6 Summary

The focus of this chapter was on devising a secure lightweight and key exchange protocol for various devices in the IoT network based on a fog computing-based smart home network model. We also presented a secure method based on network segregation and nodes' capabilities to mitigate insider threats. Furthermore, we took advantage of fog computing and presented a new architecture that helps to support and improve identity assurance. Moreover, we improve the security of the proposed scheme using temporary identities and secret session keys that change in every session and are exchanged in an unlinkable manner. The efficiency of the scheme is evaluated informally, and its security is validated through simulation using SPAN/ AVISPA tool and formally using the BAN logic. Finally, we implemented the protocol using the OMNeT++ platform.

The next chapter introduces a new authentication scheme to assure a high level of security and privacy protection in the IoT environment, especially in the smart homes.

Chapter 5

Anonymous Mutual IoT Interdevice Authentication and Key Agreement Scheme Based on the ZigBee Technique

Like many other traditional authentication schemes, IoT authentication is not immune to cybersecurity attacks [15, 33, 51]. However, until recently, most of the published authentication protocols have depended on public key encryption, although IoT end devices may not have sufficient resources to support the required operations. At the same time, some existing authentication protocols have focused their attention and effort towards the authentication between the server and IoT devices, disregarding the presence of interdevice authentication (among IoT devices). Some protocols can achieve only partial coverage of the essential security properties required or have unsatisfactory performance.

This chapter introduces an anonymous device-to-device mutual authentication and key exchange scheme based on the ZigBee technique and designed for a smart home network, an important domain in the IoT. The proposed protocol relies on symmetric encryption and

enables IoT devices to authenticate in the network and agree on a shared secret session key when communicating with each other via a trusted intermediary (home controller). To achieve perfect forward secrecy, the session keys are changed frequently after every communication session. The proposed scheme achieves secure, anonymous authentication with the unlinkability and untraceability of IoT devices' transactions. The overhead and efficiency of the proposed scheme are analyzed and compared with other related schemes. In addition, the security of the scheme is evaluated using three different methods: informal analysis, formal analysis using the Burrows–Abadi–Needham logic (BAN), and a model check using the automated validation of internet security protocols and applications (AVISPA) toolkit.

5.1 Network Model and and Security Goals

In this section, we provide an overview of the ZigBee network and introduce the network and attacker models underlying the proposed IoT authentication scheme.

5.1.1 ZigBee Network

A typical Zigbee-based network is comprised of a ZigBee coordinator and ZigBee nodes. The ZigBee coordinator is always the central device and is aware of all the nodes within its network; it is also in charge of managing the information about each node and starting and maintaining the network. The ZigBee coordinator controls each ZigBee node, such as IP camera, door lock, and so forth, in the network. Any communication between any two ZigBee nodes must go through the coordinator before reaching the destination node [19]. Thus, every ZigBee network must have a network coordinator. On the other hand, ZigBee nodes usually only are devices that interact with the physical world. There are several

topologies that are supported by ZigBee, including star, mesh, and cluster tree. The star and mesh topologies are the most common ZigBee types [54]. In the star topology, which is depicted by Fig.5.1, the coordinator is not only responsible for routing the packets in the network, but also for initiating and maintaining the devices on the network. End devices can communicate only through the coordinator.

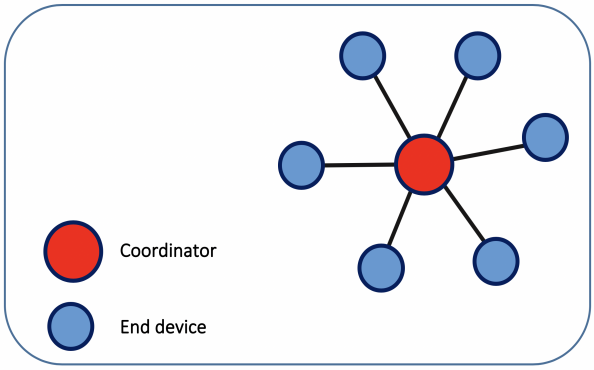

Figure 5.1: Star Topology

5.1.2 IoT Network Model

The proposed IoT authentication protocol consists of two kinds of participants: the controller and the devices. The controller denoted as C is always the central device in charge of starting and maintaining the network, and it is aware of all the devices' real identities. The controller is also responsible for forwarding messages between devices and granting a secure channel. Simply put, the controller controls each device denoted as N, such as IP camera, door lock, and so forth, in the IoT network, and any communication between any two devices is subject to the controller's permission, as shown in Fig. 5.2.

If the communication is permitted, the controller computes a temporary key and distributes it to each IoT device so that it can be used to secure communication between these devices. The network is more similar to the mesh network topology, as shown in Fig.5.3.

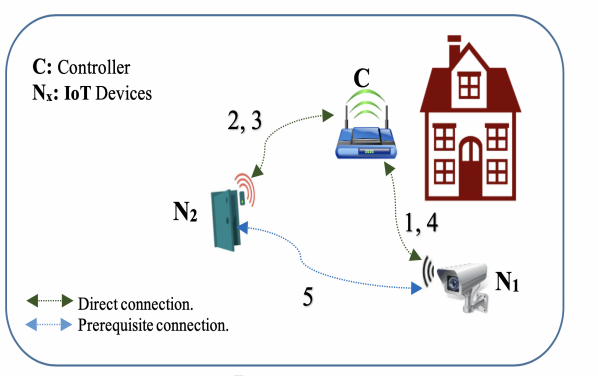

Figure 5.2: IoT Network Model

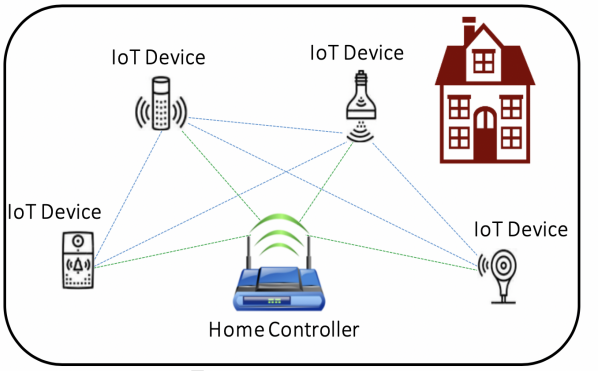

Figure 5.3: A mesh network topology

In our scheme, data security is ensured by encrypting the payload using the AES algorithm with a 128 bits key length (16 Bytes) [6]. In addition, a one-way cryptographic hash function with an incremental counter of 128 bits length (16 Bytes as a nonce) is used to validate and ensure the integrity of the data transmitted between the two devices. This is achieved by using the keyed hash message authentication code (HMAC) [27], which is appended to the following message: HMAC = h(CTR, M).

Message headers are sent in a clear but authenticated way, while the message payload is encrypted and then authenticated. In this way, we ensure that the device only reads a valid message because the HMAC filters out the invalid ciphertext before decryption, thus protecting against any attacks. In addition, the HMAC does not reveal any information

about the plaintext in the message.

The number of IoT devices can vary throughout the lifetime of the network. Moreover, IoT devices could possibly be destroyed or stolen. Moreover, because the addition of new IoT devices to the network is highly likely, this addition must be expected within IoT security protocols.

Our proposed scheme enables the dynamic addition of IoT devices to the network without causing any changes in the present security states of the network. The addition of a dynamic IoT device is solely limited by the controller, which is assumed to be a powerful device because the controller stores separate information for each IoT device in the network.

5.1.3 Security Design Goals

To achieve anonymous authentication for a trustworthy IoT, our design should achieve the following security goals:

- **Anonymity**: This is achieved based on both pseudonyms and trust so that devices can communicate anonymously with the controller and with each other without exchanging their real identities.
- **Confidentiality**: This is achieved through symmetric encryption.
- I**ntegrity**: This is achieved by using a keyed HMAC based on a trusted incremental counter.
- **Untraceability**: This ensures that the IoT devices' IDs and messages can only be traced by the trusted controller.
- **Unlinkability**: The IoT device uses a pseudonym when transmitting data. The intruder cannot trace back the real device's identity from the pseudonym. The use of

pseudonyms also ensures that when an IoT device uses network resources multiple times, it will be challenging and difficult to link these uses together by intruders. Thus, the communication sessions are not linkable, and the same property applies to both the real device's identity and the pseudonym.

- **Forward/backward security**: This ensures that if the sender's private key is compromised, the past and future messages will remain confidential.

5.2 Proposed approach

Our proposed scheme is a symmetric encryption-based authentication scheme with key agreement. The symmetric encryption is based on an AES 128/CCM (counter with CBC-MAC) mode. Furthermore, this scheme is payload embedded, that is, the authentication and key exchange process are embedded in the payload. In this way, the handshaking overhead is reduced.

In this section, the various phases in the life cycle of an IoT device from the manufacturing phase to the key exchange phase are laid out. Two common problems in IoT cryptography are further explored, as follows:

1. Device-to-device authentication: where a device determines that one or more devices are indeed who they say they are.

2. Key exchange: where two or more devices agree on a shared key that can later be used to establish an encrypted communications channel.

In our proposed protocol, an IoT device (N) and a controller node (C) are responsible for ensuring secure, anonymous mutual authentication and key exchange. In more detail, the

proposed protocol framework consists of a pre-deployment phase, a registration phase, and a protocol execution phase. The protocol execution phase is broken down into two other phases: the device-to-device authentication and key exchange and the device-to-device communication. In the proposed scheme, all IoT entities will be referred to as IoT devices, with the centralized device named the controller. As aforementioned, our scheme utilizes symmetric encryption between devices similar to ZigBee, in which only the controller that possesses the symmetric keys of the IoT devices can make a direct connection with these devices and encrypt the messages. The devices encrypt their messages using their symmetric keys when communicating with the controller, and these messages can only be read by the controller. Furthermore, communication across the entire network occur from one device to another and are subject to the controller's permission. If the communication is allowed, the controller will generate a temporary session key and share it with the communication participants. For example, an IP camera wishing to send a signal to open a home's door lock will go through the controller so that the controller can act and allow the communication session to be established between the IP camera and the home's door lock. The temporary session keys, which change for every communication session, satisfy the security properties of perfect forward secrecy. The security parameters of the IoT network entities are summarized in Table 5.1.

Table 5.1: Security of IoT network

Security property ↓ Phases →	Controller (C) ↔ IoT device (N)	IoT device (N) ↔ IoT device (N)
Confidentiality with symmetric encryption	Symmetric key (K_N)	Temporary shared key (TK)
Integrity and Authenticity With keyed-hash Message Authentication Code (HMAC)	Incremental counter (CTR) as a nonce	Nonce (ns)

Furthermore, during the communication process, IoT devices use dynamic IDs (pseudonyms) instead of using their real IDs. Dynamic IDs offer the anonymity of senders' IDs, receivers' IDs, and the sender-receiver relationship, in addition to device untraceability and end-to-end

flow untraceability.

The abstract notations used to describe our authentication protocol are listed in Table 5.2. The framework phases are explained below.

5.2.1 Predeployment phase

During the pre-deployment phase, each IoT device is provided with an identity (ID) and assumed to possess a "fingerprint" generated from an accelerometer [66]. The accelerometer is embedded in the device's inner chips during the device manufacturing process. The main benefit of such an accelerometer is to generate a relevant, unique key for the device [59]. The uniqueness of the generated key in the network would be guaranteed based on the diversity of the manufacturing principles and methods. The key will be considered the symmetric key for a device in the network. Moreover, we assume that each IoT device knows the controller's public key.

5.2.2 Entities Registration Phase

When the system administrator (SA) receives the IoT devices, he or she will register all devices using their real identities (IDs) in the home controller (C). In this way, C can identify the registered IoT devices and differentiate between them during the authentication process. In addition, SA can capitalize on the access control list to prevent illicit access to and control access between IoT devices using the IoT devices' real identities (IDs). For example, it may prevent device 1 (ID_{N1}) from accessing, say, device 7 (ID_{N7}). We emphasize the fact that the IoT devices' real identities (IDs) are kept secure and never transmitted in plain text because we use only the dynamic IDs during authentication and communication.

Remark 1. The identity ID_N represents the secure permanent real identity for the IoT device, which is never transmitted in plain text. Every IoT device has a unique identity ID and knows the identities of the connected devices that it should communicate with. These permanent real identities facilitate the authentication process between the controller and IoT devices, as shown in Fig. 5.4.

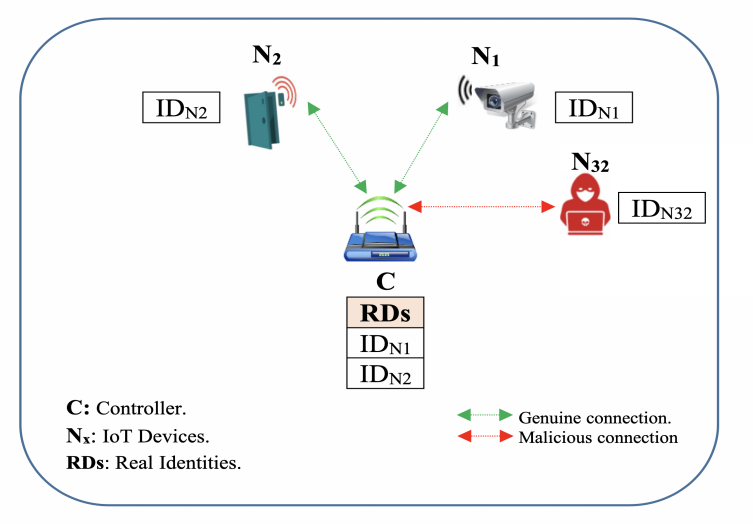

Figure 5.4: Identity management

5.2.3 Authentication phase

The entity authentication process is performed in the system configuration phase. The first step in the authentication process starts with the IoT device and is based on the fact that the controller's public key K_C is known by all network devices and, each IoT device is initially provided with a real identity ID by the manufacturer, as discussed in subsection 5.2.1. This ID facilitates the validation process done by the controller, as explained later in this section. The IoT device will send its symmetric key K_N and the counter CTR_N to the controller using K_C because the key exchange has to be in a secure channel. The counter CTR_N is incremented in every message passing between the IoT device and the controller.

Fig. 5.5 describes the steps involved in the authentication stage.

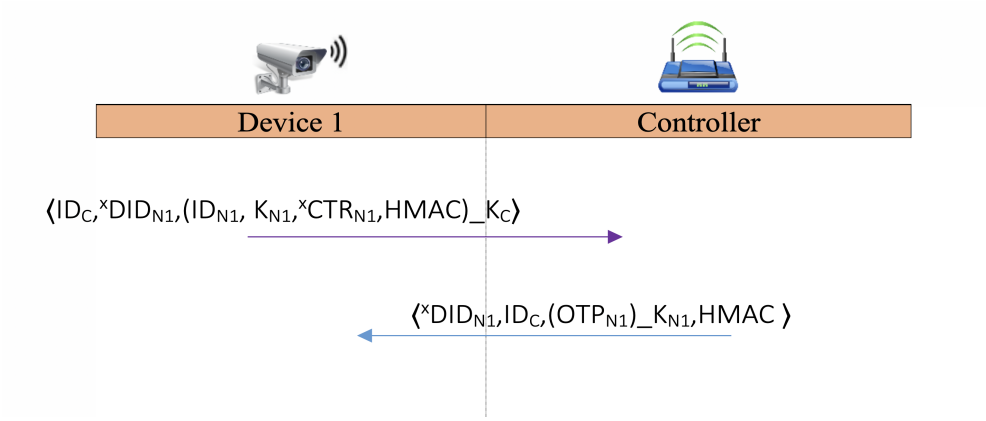

Figure 5.5: Entity's authentication process

The following steps summarize the authentication phase between N1 and C:

Step A1 N1 $\rightarrow C : \langle ID_{N1}, K_{N1}, {}^{x}CTR_{N1}, HMAC)\_K_C \rangle$. N performs as follows.

First, IoT device N1 generates two parameters, namely, K_{N1} and ${}^{x}CTR_{N1}$. K_{N1} is the IoT device symmetric key that is used to encrypt data between C and N1. ${}^{x}CTR_{N1}$ is a counter that is incremented on every message, where x denotes the sequence number corresponding to the previous counter record stored in the database for a given device. Such counter is used to ensure the freshness of the protocol instance, help in the validation process of the integrity of encrypted data, and ensure IoT device anonymity and unlinkability properties. Even if the symmetric key is compromised, the counter ${}^{x}CTR_{N1}$ will withstand and help protect against replay and data integrity attacks.

N1 creates a dynamic identity ${}^{x}DID_{N1}$ =h(ID_{N1},${}^{x}CTR_{N1}$) that changes for every session. Because ${}^{x}CTR_{N1}$ is different in each protocol instance, ${}^{x}DID_{N1}$ will be unique in each session and can only be tracked by controller C; hence, the untraceability

Table 5.2: Notations used in our protocol

Notation	Description
SA	System Administrator
N1	IoT Device 1
N2	IoT Device 2
C	Controller
K_{N1}	Device 1 symmetric key
K_{N2}	Device 2 symmetric key
K_{12}	Temporary shared key between N1 and N2
K_C	Controller's public key
K_{CP}	Controller's private key
OTP_{N1}	One-time password for N1
ID_{N1}	Real device 1 identity
ID_{N2}	Real device 2 identity
ID_C	Controller identity
DID_{N1}	Dynamic identity of device 1
DID_{N1}	Dynamic identity of device 2
CTR_{N1}	Counter value between N1 and C
CTR_{N2}	Counter value between N2 and C
ns	Nonce
PoB	Proof of Belonging
PoC	Proof of Controller
HMAC	keyed-hash message authentication code
M	All message parameters but HMAC
Msg	All message parameters
N $\rightarrow$ C:Msg	Device N sends the message Msg to controller C via a public channel

propriety is achieved. In the first authentication message, N1 concatenates ID_{N1}, K_{N1} and ${}^{x}CTR_{N1}$ and stores them in M=($ID_{N1}||K_{N1}||{}^{x}CTR_{N1}$). Next, N1 calculates the HMAC=h(${}^{x}CTR_{N1}$,M). In this way, we guarantee that each HMAC is calculated based on a dynamic key so that if one key is compromised, we can still detect the attack. Next, N encrypts M and HMAC, and sends the message Msg to the controller C. However, in all other messages, HMAC is sent unencrypted. Finally, at the end

of the step, N1 increments ${}^{x}CTR_{N1}$ by 1 (which becomes ${}^{x+1}CTR_{N1}$) and updates its database, as shown in Table 5.3.

Table 5.3: Device 1 Database

Device 1 (N1)	
Device Identity	ID_{N1}
Dynamic of Device Identity	${}^{x}DID_{N1}$
Device One-time password	-
Counter	${}^{x+1}CTR_{N1}$
Symmetric key	K_{N1}
Controller Key	K_C

Step A2 C $\rightarrow N1 : \langle {}^{x}DID_{N1}, ID_C, (OTP_{N1})\_K_{N1}, HMAC \rangle$. C performs as follows.

Once controller C receives the message, it decrypts the message using K_{CP} and looks the ID_{N1} up and validates it. Then, it computes *HMAC =(${}^{x}CTR_{N1}$,M) and ${}^{*x}DID_{N1}$ = h(ID_{N1},${}^{x}CTR_{N1}$) and verifies whether *HMAC and ${}^{*x}DID_{N1}$ are equal to the received HMAC and ${}^{x}DID_{N1}$, respectively. C stores N1 information, namely K_{N1}, ${}^{x}CTR_{N1}$ and ${}^{x}DID_{N1}$, in its database. The controller has a random number generator, which contributes to generating OTP_{N1} (used for the first communication established in the configuration stage) for all devices in the network. C increments ${}^{x}CTR_{N1}$ by 1 (which becomes ${}^{x+1}CTR_{N1}$). Next, C encrypts OTP_{N1} using K_{N1} and concatenates the message parameters as M=(${}^{x}DID_{N1} \| ID_C \| (OTP_{N1})\_K_{N1}$). After that, C computes HMAC=h(${}^{x+1}CTR_{N1}$,M). Then, C sends the message Msg to N1. C increments ${}^{x+1}CTR_{N1}$ again by 1 (which becomes ${}^{x+2}CTR_{N1}$) and computes ${}^{x+1}DID_{N1}$= h(ID_{N1},${}^{x+2}CTR_{N1}$) for the upcoming session. Finally, C updates N1 information, as shown in Table 5.4.

Table 5.4: Controller Database

	N1	N2
Device Identity	ID_{N1}	
Dynamic of Device Identity	${}^{x}DID_{N1}$	
Device One-time password	OTP_{N1}	
Device symmetric key	K_{N1}	
Counter value	${}^{x+2}CTR_{N1}$	

Step A3 N1: N1 performs as follows.

Upon receipt of the message by N1, N1 computes *HMAC = h(${}^{x+1}CTR_{N1}$,M). Then, it compares the HMAC received in the message with *HMAC. If there is a match, the message's integrity is correctly verified, and the message is considered to be a new message from C because ${}^{x+1}CTR_{N1}$ is fresh. After that, N1 decrypts the message using K_{N1} and extracts OTP_{N1}. N1 increments ${}^{x+1}CTR_{N1}$ by 1 (which becomes ${}^{x+2}CTR_{N1}$) and computes a new dynamic identity ${}^{x+1}DID_{N1}$= h(ID_{N1},${}^{x+2}CTR_{N1}$) for the upcoming session. Finally, N1 updates its database, as shown in Table 5.5.

Table 5.5: Device 1 Database (updated)

Device 1 (N1)	
Device Identity	ID_{N1}
Dynamic of Device Identity	${}^{x+1}DID_{N1}$
Device One-time password	OTP_{N1}
Counter	${}^{x+2}CTR_{N1}$
Symmetric key	K_{N1}
Controller Key	K_C

At this stage, the authentication process is done, and the controller will release a secure channel between N1 and itself when the device wishes to send or receive data.

5.2.4 Communication phase (Interdevice authentication and data transfer)

In the proposed scheme, communication can occur between the IoT devices and the controller or among the IoT devices. The first scenario was explained in the previous section. Now, we will explain the second scenario of communication among devices. Fig. 5.6 describes the device communication process. The most important aspect that needs to be addressed before going through the communication scenario is the security of the channel and the exchanged messages. The AES algorithm will be utilized to encrypt the exchanged data among the IoT network devices. As mentioned in the device authentication section, each device has a unique symmetric key, so the messages will be encrypted using the symmetric key when sent to the controller. However, for IoT device N1 (sender) to communicate with another IoT device N2 (receiver) in the smart home network, the sender initially forwards this communication request to the controller, which will check some secure parameters such as nonce, incremental counter, and HMAC, and if these parameters are valid, it will check its database for the receiver's information and establish a secure communication session between N1 and N2 using some authentication parameters, such as proof of control (PoC), nonce (Ns), and temporary shared key TK_{12}, as explained in this subsection. PoC is used to prove to the receiving IoT device that the controller is aware of the data request sent by another IoT device because PoC contains secret parameters.

The communication phase occurs after successful completion of the authentication phase. In this phase, N1 wishes to send data to N2. To deliver the data from N1 to N2, the sequence of steps that are taken is summarized as follows:

Step C1 N1 $\rightarrow C : \langle ID_C, ^{x+1}DID_{N1}, PoB, (ID_{N2})\_K_{N1}, HMAC\rangle$. N1 performs as fol-

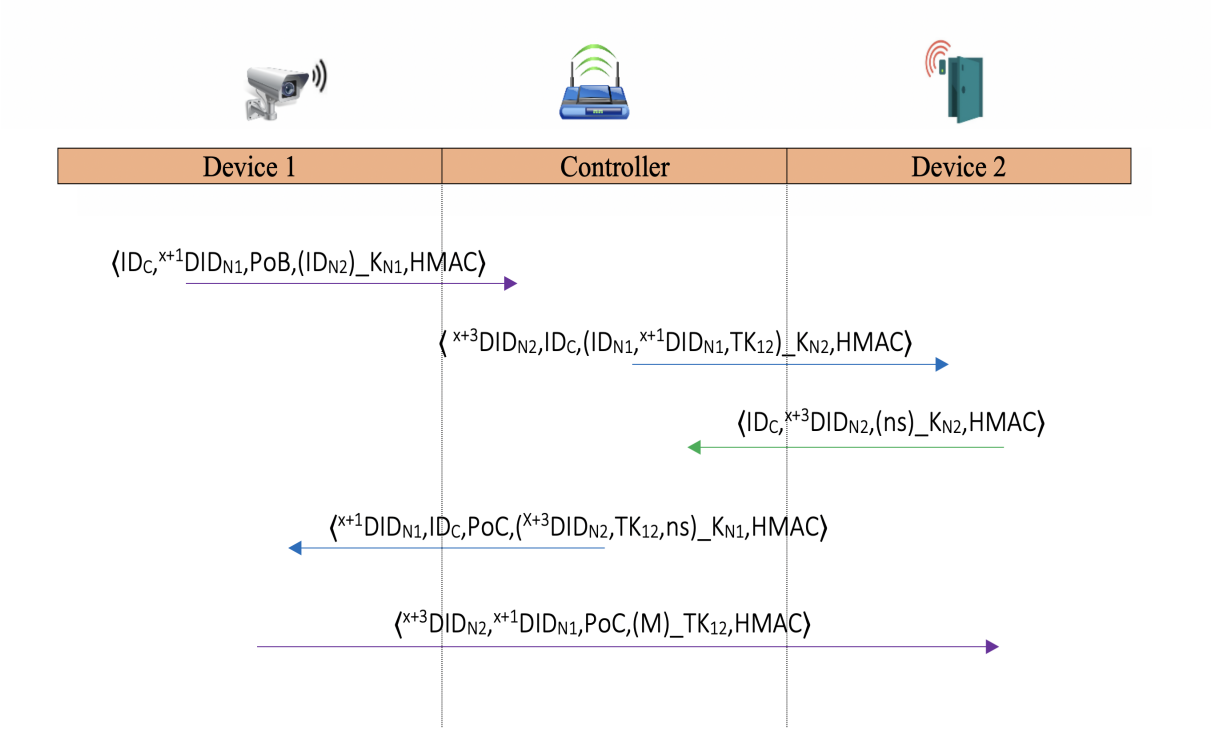

Figure 5.6: Entity's authentication and communication between IoT devices

lows.

When N1 wishes to communicate with another IoT device–say N2, where it will be assumed that N2 is an authentic device–it connects with the controller and asks for a secure connection with N2. N1 will load $^{x+1}DID_{N1}$ and OTP_{N1} and compute the proof of belonging PoB =h (OTP_{N1},$^{x+2}CTR_{N1}$). PoB is constituted of OTP_{N1} and $^{x+2}CTR_{N1}$. Thus, PoB proves to C the authenticity of N1 and the freshness of the message during the first data communication. Next, N1 loads the destination identity ID_{N2} and encrypts it using K_{N1}. Then, N1 concatenates the message parameters as M = ($ID_C \| ^{x+1}DID_{N1} \| PoB \| (ID_{N2})\_K_{N1}$). After that, N computes HMAC = h($^{x+2}CTR_{N1}$,M) and sends the message Msg to C to request a data channel. Finally, N1 increments the counter value (which becomes $^{x+3}CTR_{N1}$) and updates its database.

Step C2 C $\rightarrow N2 : \langle {}^{x+3}DID_{N2}, ID_C, (ID_{N1}, {}^{x+1}DID_{N1}, TK_{12})\_K_{N2},$ $HMAC\rangle$. N1 performs as follows.

Once C obtains the message, it computes *PoB=h(OTP_{N1} ,${}^{x+2}CTR_{N1}$) and *HMAC =h(${}^{x+2}CTR_{N1}$,M) and verifies whether *PoB and *HMAC are equal to the received PoB and HMAC, respectively. Otherwise, the message will be dropped and considered part of a replay attack. After that, C decrypts the received message using K_{N1} and extracts ID_{N2}. Then, C increments the counter value (which becomes ${}^{x+3}CTR_{N1}$) and updates its database.

Next, C prepares a message to be sent to N2. It first loads ${}^{x+3}DID_{N2}$ for ID_{N2} from its database and then generates the key TK_{12} using a random number generator, which will be used to encrypt the data communication between N1 and N2. This key will change for each session, ensuring that the data exchange will be safe every time. After that, it encrypts ID_{N1}, ${}^{x+1}DID_{N1}$ and TK_{12} using K_{N2} and concatenates the message parameters as M=(${}^{x+3}DID_{N2} \| ID_C \| (ID_{N1}, {}^{x+1}DID_{N1}, TK_{12})\_K_{N2}$). Then, it computes HMAC =h(${}^{x+4}CTR_{N2}$,M) based on the most recent counter value of N2– denoted as ${}^{x+4}CTR_{N2}$. Then, it sends the message Msg to N2. Finally, C increments N2's counter value (which becomes ${}^{x+5}CTR_{N2}$) and updates its database.

Step C3: **N2** $\rightarrow C : \langle ID_C, {}^{x+3}DID_{N2}, (ns)\_K_{N2}, HMAC\rangle$. N2 performs as follows.

Upon the receipt of the message by N2, N2 computes *HMAC =h(${}^{x+4}CTR_{N2}$, M) and verifies whether *HMAC is equal to the received HMAC. Next, N2 decrypts the received message using K_{N2} and extracts ID_{N1}, ${}^{x+1}DID_{N1}$, and TK_{12}. Then, N2 generates a nonce ns that is used to uniquely identify the request and prove the freshness of the established connection between N1 and N2 later on. This nonce is a

random number guaranteed not to be reused in the lifetime of the device generating it. After that, N2 encrypts ns using K_{N2} and concatenates the message parameters as M=($ID_C \| {}^{x+3}DID_{N2} \| (ns)\_K_{N2}$). N2 increments ${}^{x+4}CTR_{N2}$ by 1 (which becomes ${}^{x+5}CTR_{N2}$) and computes HMAC =h(${}^{x+5}CTR_{N2}$,M). Next, it sends the message Msg to C. Finally, N2 increments the counter value (which becomes ${}^{x+6}CTR_{N2}$) and updates its database.

Step C4: $\mathbf{C} \rightarrow N1 : \langle {}^{x+1}DID_{N1}, ID_C, ({}^{x+3}DID_{N2}, TK_{12}, ns)\_K_{N1}$ $, HMAC \rangle$. N1 performs as follows.

When C receives the message from N2, it will look ${}^{x+3}DID_{N2}$ up. Next, it will compute *HMAC=h(${}^{x+5}CTR_{N2}$, M) and compare it with the one in the received message. If both are correct, C decrypts the received message using K_{N2} and extracts ns. Then, C increments N2's counter value (which becomes ${}^{x+6}CTR_{N2}$) and updates its database.

Next, C will prepare a message to be sent to N1. It will first compute PoC=h(ID_{N2}, ID_{N1}, ${}^{x+3}DID_{N2}$,${}^{x+1}DID_{N1}, ns, TK_{12}, K_{N2}$), which is used to prove to N2 that C is aware of this request because PoC contains secret parameters, such as K_{N2}, which must be known only by C. Next, C loads ${}^{x+3}DID_{N2}, TK_{12}$ and ns and encrypts them using K_{N1}. Then, it concatenates the message parameters as M=(${}^{x+1}DID_{N1} \| ID_C \| PoC \| ({}^{x+3}DID_{N2}$ $, TK_{12}, ns)\_K_{N1}$) and computes HMAC=h(${}^{x+3}CTR_{N1}$,M). After that, it sends the message Msg to N1. Finally, C will store N1's and N2's counter values and their dynamic identities in its database.

Step C5: $\mathbf{N1} \rightarrow N2 : \langle {}^{x+3}DID_{N2}, {}^{x+1}DID_{N1}, PoC, (M)\_K_{12}, HMAC \rangle$. N1 performs as follows.

Upon receiving the message by N1, it will compute *HMAC=h($^{x+3}CTR_{N1}$, M) and verify whether *HMAC is equal to the received HMAC. If the verification is successful, N1 will decrypt the received message using K_{N1} and extract $^{X+3}DID_{N2}$, TK_{12} and ns. Otherwise, C will be rejected. After that, N1 will increment the counter value (which becomes $^{x+4}CTR_{N1}$) and update its database.

Next, N1 will prepare a message to be sent to N2. It will first encrypt the intended message using TK_{12} and then concatenate the message as M=($^{x+3}DID_{N2}||^{x+1}DID_{N1}||PoC||$ ($data$)_TK_{12}) and compute HMAC=h(ns,M). Then, it sends the message Msg to N2. Finally, it will compute $^{x+2}DID_{N1}$= h(ID_{N1},$^{x+4}CTR_{N1}$) for the upcoming session with C, and update its database.

Step C6: **N2**: N2 performs as follows.

Upon the receipt of the message, N2 computes *PoC = h(ID_{N2}, ID_{N1},$^{x+3}DID_{N2}$,$^{x+1}DID_{N1}$,ns,TK_{12},K_{N2}) and *HMAC=h(ns,M) and verifies whether *PoC and *HMAC are equal to the received PoC and HMAC, respectively. If the abovementioned parameters match, N2 will decrypt the received message using TK_{12} and accept data input. Finally, it will compute $^{x+4}DID_{N2}$= h(ID_{N2},$^{x+6}CTR_{N2}$) for the upcoming session and update its database.

This marks the end of the data communication process.

Remark 1. Notice that TK_{12} is a temporary key generated by the controller and is agreed on by the communicating parties. The temporary keys are utilized by IoT devices to establish direct communication channels among them.

Remark 2. As mentioned in subsection 5.2.2, each IoT device is aware of the devices that it should communicate with. The permanent real identities facilitate data communication between IoT devices, as shown in Fig. 5.7.

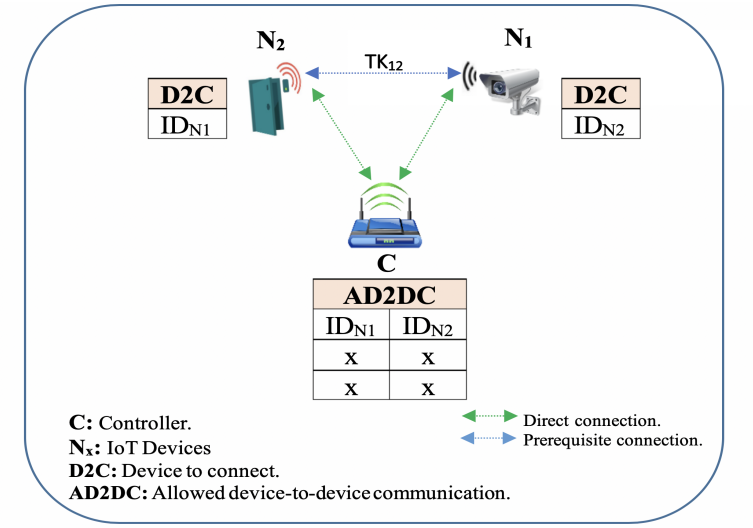

Figure 5.7: Data communication through the controller

Remark 3. The key used in HMAC calculations (ns) during the communications among IoT devices is randomly generated by the receiving IoT device and is only valid for the particular communication session.

$$HMAC = h(ns, M)$$

5.3 Protocol Efficiency Evaluation

In this section, we analyze the overhead and efficiency of the proposed protocol in terms of its storage, communication overhead, and performance.

5.3.1 Storage requirements

In our scheme, each IoT node is required to store its real identity ID_N, dynamic identity DID_N, and secret key K_N, incremental counter CTR_N, and the controller identity ID_C. We use SHA-1 as an example of hash function, and the output of SHA-1 is 160 bits. By applying these settings, we obtain $|DID_N|$ = $|CTR_N|$ =160 bits, while $|ID_N|$=$|ID_C|$= 16 bits, which is assumed short, and K_N =128 bits. On the other hand, C is required to store the tuple DID_N, ID_N, K_N CTR_N, in addition to ID_C. By applying these settings, we obtain $|DID_N|$ = $|CTR_N|$ = 160 bits, while $|ID_N|$=$|ID_C|$= 16 bits, and K_N =128 bits. Hence, the total storage required by each IoT node N is 480 bits, and the total storage required by controller C is (n × 464 + 16), where n is the number of registered IoT nodes. The storage requirements are summarized in Table 5.6.

Table 5.6: Storage cost of our scheme

Node	Storage cost (in bits)
N	480
C	464n + 16

5.3.2 Communication overheads

In the transmission (N → C), N sends the tuple, DID_N, ID_C, ID_N, K_N, CRT_N and HMAC. Therefore, the size of the transmitted tuple is (3 × 160)+(2 × 16) + 128 = 640 bits. In the transmission (C → N), C sends the tuple, ID_C, DID_N, OTP_N and HMAC of size ((2 × 16) + (2 × 160)) = 352 bits. The transmission (N1→ N2), N1 sends the tuple, DID_N1, DID_N2, PoC and HMAC of size (4 × 160) = 640 bits. The communication overheads are summarized in Table 5.7.

We separately present the communication overhead graph for our scheme in Fig. 5.8.

Table 5.7: Communication overheads of our scheme

Communication between devices	Communication cost
N → C	640 bits
C → N	352 bits
N1 → N2	640 bits

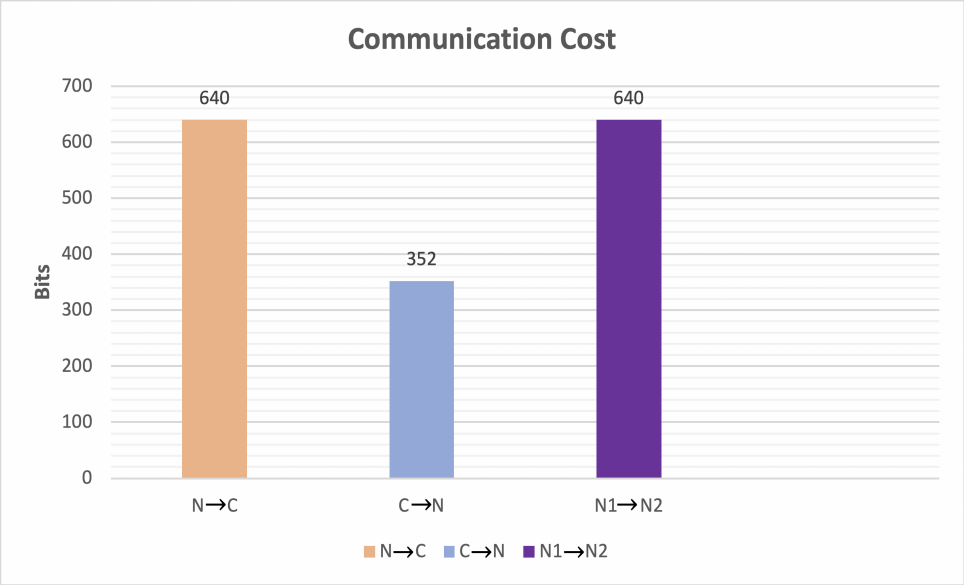

Figure 5.8: communication overheads graph for our scheme

5.3.3 Performance analysis

In this section, we conduct the performance analysis of our proposed scheme. We specifically compute the total cost required by the mathematical operations of the scheme and the number of terms involved in the communication or messages in transit. We also present a comparison between our proposed scheme and other related schemes [32, 45, 56] in terms of the computational cost of the hash operations (C_H), symmetric key encryption operations (C_E), symmetric key decryption operations (C_D), XOR operations (C_{XOR}), and modular operations (C_{MOD}). Table 5.8 shows the comparison of the computational cost of our scheme to other related schemes. As shown in Table 5.8, some of the related schemes are missing some phases; thus, these are marked as N/A, which indicates they are not available.

From Table 5.8, it can clearly be seen that the number of hash operations in our scheme is less than those in [32] and the same as those in [45]. In addition, our scheme does not involve any exclusive-OR operations as in [45, 56]. Furthermore, although our scheme consists of more phases, it involves almost the same total number of symmetric key encryption and decryption operations as the aforementioned related schemes. We also provide a comparison graph for the computational costs for authentication and data exchange N ↔ C/S, as shown in Fig. 5.9.

Table 5.8: Performance analysis of our scheme

Phases ↓ Schemes →	Ukil et al. [56]	Risalat et al. [45]	Lohachab [32]	Proposed scheme
Registration	N/A	N/A	$7C_H$	0
Authentication and data exchange N → C/S	$2C_{XOR} + 4C_E + 4C_D$	$14C_{XOR} + 8C_H + 3C_E + 3C_D$	$5C_H + 3C_E + 3C_D$	$4C_H + 2C_E + 2C_D$
Authentication and data exchange N → N	N/A	N/A	N/A	$4C_H + 3C_E + 3C_D$
Credential (ID and Secret key) change	N/A	N/A	$C_H + 2C_E + 2C_D$	C_H
Total (Maximum)	$2C_{XOR} + 4C_E + 4C_D$	$14C_{XOR} + 8C_H + 3C_E + 3C_D$	$13C_H + 5C_E + 5C_D$	$9C_H + 5C_E + 5C_D$

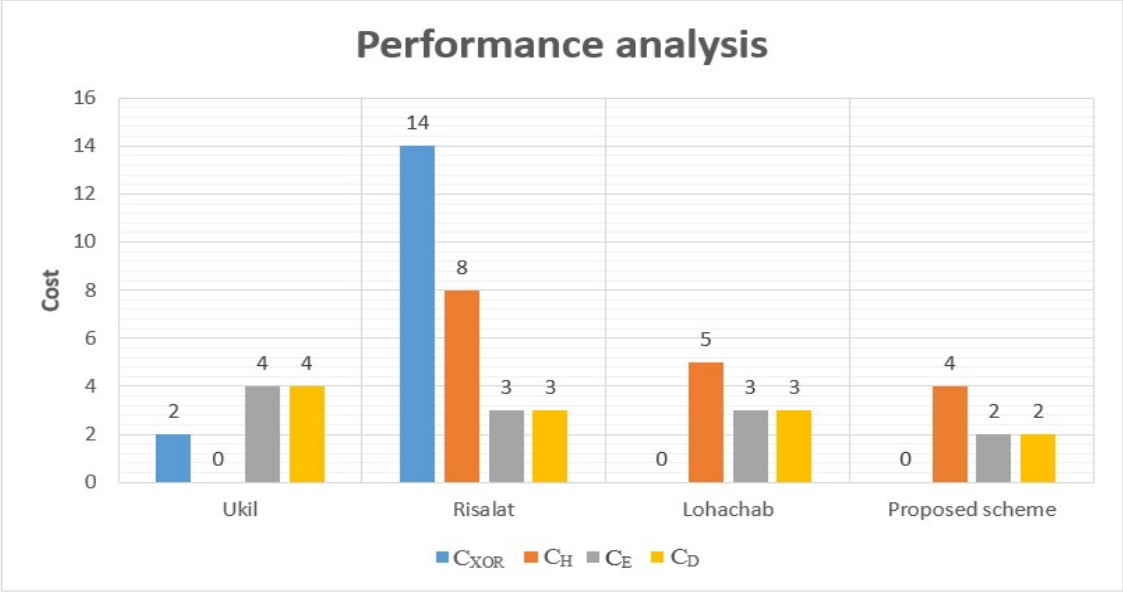

Figure 5.9: Comparison graph for computation cost for Authentication and data exchange N ↔C/S

5.4 Protocol security validation

In this section, we validate the security of our proposed protocol using three different approaches: informal security analysis, formal validation using theorem proving based on BAN logic, and formal model checking and simulation using AVISPA.

5.4.1 Security analysis

In this section, the security of the protocol is explored against various known attacks. We discuss various known attacks and elaborate on how our protocol successfully resists and stops these attacks.

5.4.1.1 Replay attack

A replay attack is defeated using the incremental counter ${}^{X}CTR$ and nonce ns. The counter ${}^{X}CTR$ is generated by the IoT device in the first message in the authentication and then used in the HMAC that is sent in every transmitted message, ensuring the message cannot be replaced by the attacker. Both the controller and the IoT device maintain the counter value, increment it after every session, and keep it synchronized; ns is generated by the receiving IoT device, and it is different in each session and never used again. This ensures that the message cannot be replaced by the attacker. Furthermore, the attacker cannot find and reach the temporary shared secret key TK_{12} because she or he is totally unaware of the real identity of IoT device ID_N and the counter ${}^{X}CTR$, which changes for every session. Consequently, the attacker does not know who the message is from or for; thus, any replay attack attempt will fail. Moreover, the HMAC prevents any replay attack attempts. Hence, our protocol protects against the replay attack.

5.4.1.2 Eavesdropping attack

In the authentication phase, the attacker can easily intercept the message in transit between C and N1. However, this kind of attack is prevented because we encrypt the message in transit using symmetric encryption. Moreover, the attacker cannot link the message to a particular device because we use dynamic identities DID_N that change in every session. Moreover, the information that is sent in clear text, such as DID_N and PoC, does not pose any threat because this information is constructed from random parameters that change in every session, such as CTR_N and ns.

5.4.1.3 Impersonation attack

In this attack, when the intruder eavesdrops the messages transmitted from N to C, she or he can use the intercepted information for malicious actions, such as impersonating the IoT device N1 and sending fabricated messages. This attack will not succeed for numerous reasons. First, the dynamic identity of the IoT device DID_N changes for every session. Second, HMAC is protected using the one-way hash function of the message, which includes CTR_N. Thus, our protocol protects against the impersonation attack.

5.4.1.4 Man-in-the-middle attack

As discussed above in the replay, eavesdropping, and impersonation attacks, the man-in-the-middle attack is defeated because the real identity of the IoT devices ID_N, the counter value CTR_N, the symmetric keys, the nonce ns, and the temporary shared key TK_{12} are unknown to the attacker. Moreover, the HMAC is protected using CTR_N. Therefore, the man-in-the-middle attack is prevented.

5.4.1.5 Attack against the temporary secret key

The temporary shared key TK_{12} is generated by controller C through random number generation and changes in every session.

5.4.1.6 Device stolen database attack

If an IoT device is hacked, the other devices will not be affected because the device does not store any secret keys of the other devices in its memory. In addition, these secret keys are different, thus overcoming the issue of having any secret key exposed; if an attacker discloses a secret key on one specific IoT device, the secret keys of the other IoT devices will not be compromised. In addition, the attacker is unable to obtain the temporary shared key because it is only used during a particular session, and then, it is destroyed. Moreover, the IoT device stores the real identities of the other IoT devices that it is allowed to communicate with to ask controller C to open sessions with them, but these identities are useless for the attacker because the attacker does not know how to use them to establish any session.

5.4.1.7 Forward/backward security

The objective of the forward/backward security property is to ensure that any past or future session keys will not be affected when any temporary session key TK_{12} is exposed. Recall that our temporary shared key is generated using a random number generator, and it changes for every session. Hence, forward/backward security is achieved by our authentication scheme.

5.4.1.8 Session key Guessing Attack

The temporary shared key TK_{12} is generated by controller C using a random number generator. Because TK_{12} depends on randomness, the attacker cannot obtain it from the protocol. The probability of guessing this is so negligible that the attacker will fail to guess the temporary shared key given that TK_{12} changes for every session.

5.4.1.9 Anonymity Unlinkability, and Untraceability properties

Consider the tuple ‹DID_{N1}, ID_C, ID_{N1}, K_{N1}, HMAC› from N to C. We have DID_N = h(ID_N,$^{X}CTR_N$), and because $^{X}CTR_N$ is a fresh random number chosen by N in each session, the attacker cannot link any two different TID_Ns to the same N and cannot trace a given IoT device to N's messages. The sensitive data, such as ID_{N1}, will always be encrypted. The parameter HMAC is protected using a one-way hash function of the message and the incremental counter CTR_N, which is out of the adversary's reach. Now, consider the tuple ‹ID_C, DID_N, OTP_N, HMAC› from C to N; these parameters follow the same principle as discussed above for the tuple from N to C. DID_{N1} remains the same for each session, while CTR_N is fresh in each message; thus, HMAC is different. Therefore, the use of the random values DID_N, HMAC, $^{X}CTR_N$ and ns, as well as the hash functions, ensure that all messages transmitted by N or C via public channels are anonymous. Furthermore, for two or more authentication messages that are sent by the same device, the attacker cannot decide whether these authentication messages come from the same device or not; this means that device N cannot be linked to different sessions. Hence, our proposed scheme provides anonymity and unlinkability, and the attacker cannot trace the devices by intercepting messages.

5.4.2 Formal proof based on BAN logic

5.4.2.1 Goals of the analysis of our authentication scheme

In this section, the main goals of the analysis of our authentication scheme are defined as follows:

- G1: C believes N believes K_N is a secure shared parameter between N and C.

$$C| \equiv N| \equiv (N \overset{K_N}{\leftrightarrow} C)$$

- G2: C believes K_N is a secure shared parameter between N and C.

$$C| \equiv (N \overset{K_N}{\leftrightarrow} C)$$

- G3: N believes C believes OTP is a secure shared parameter between N and C.

$$N| \equiv C| \equiv (N \overset{OTP}{\leftrightarrow} C)$$

- G4: N believes OTP is a secure shared parameter between N and C.

$$N| \equiv (N \overset{OTP}{\leftrightarrow} C)$$

5.4.2.2 Messages transferred in the authentication

The idealized messages that are exchanged in the authentication phase between an IoT device N and controller C are as follows:

- M1: N $\rightarrow$ C: $\langle$ N $\overset{K_N}{\leftrightarrow}$ C, $^{X}CTR,ID_N$ $\rangle_{\mathrm{N} \overset{ID_N}{\leftrightarrow} \mathrm{C}}$

- M2: C → N: $\langle$ N $\overset{K_N}{\leftrightarrow}$ C, ^{X+1}CTR, ID_N, N $\overset{OTP}{\longleftrightarrow}$ C $\rangle_{\text{N} \overset{ID_N}{\longleftrightarrow} \text{C}}$

5.4.2.3 Introductory assumptions

The fundamental assumptions of our authentication scheme are as follows:

- A1: C believes ID_N is a secure shared parameter between N and C.

$$C| \equiv (N \overset{ID_N}{\longleftrightarrow} C)$$

- A2: C believes ^{X}CTR is fresh.

$$CRN| \equiv \#(^{X}CTR)$$

- A3: C believes N believes ID_N is a secure shared parameter between N and C.

$$C| \equiv N| \equiv (N \overset{ID_N}{\longleftrightarrow} C)$$

- A4: N believes ID_N is a secure shared parameter between N and CRN.

$$N| \equiv (N \overset{ID_N}{\longleftrightarrow} C)$$

- A5: N believes ^{X+1}CTR is fresh.

$$N| \equiv \#(^{X+1}CTR)$$

- A6: N believes C has jurisdiction over OTP, which is a secure shared parameter

between N and C.

$$N| \equiv CRN| \Rightarrow (N \overset{OTP}{\longleftrightarrow} C)$$

5.4.2.4 Analysis of our authentication scheme

The analysis of our authentication scheme is shown below to prove that the scheme achieves mutual authentication between N and C.

D1: From assumption A1 and message M1, and by applying the message meaning rule, we derive the following:

$$\frac{C|\equiv(N \overset{ID_N}{\longleftrightarrow} C), C \triangleleft \langle N \overset{K_N}{\longleftrightarrow} C, {}^{X}CTR, ID_N \rangle_{\mathrm{N}} \overset{ID_N}{\longleftrightarrow} \mathrm{C}}{C|\equiv N \sim (N \overset{K_N}{\longleftrightarrow} C, {}^{X}CTR, ID_N)}$$

D2: From assumption A2 and by applying the freshness rule, we derive the following:

$$\frac{C|\equiv\#({}^{X}CTR)}{C||\equiv\#(N \overset{K_N}{\longleftrightarrow} C, {}^{X}CTR, ID_N)}$$

D3: From derivations D1 and D2 and by applying the nonce verification rule, we derive the following:

$$\frac{C|\equiv\#(N \overset{K_N}{\longleftrightarrow} C, {}^{X}CTR, ID_N), C|\equiv N|\sim(N \overset{K_N}{\longleftrightarrow} C, {}^{X}CTR, ID_N)}{C|\equiv N\equiv(N \overset{K_N}{\longleftrightarrow} C, {}^{X}CTR, ID_N)}$$

D4: From derivation D3 and by applying the belief rule, we derive the following:

$$\frac{C|\equiv N\equiv(N \overset{K_N}{\longleftrightarrow} C, {}^{X}CTR, ID_N)}{C|\equiv N\equiv(N \overset{K_N}{\longleftrightarrow} C)} \quad (\text{Goal G1})$$

D5: From assumption A3 and derivation D4 and by applying the jurisdiction rule, we derive the following:

$$\frac{C|\equiv N\Rightarrow(N \overset{K_N}{\longleftrightarrow} C), C|\equiv N\equiv(N \overset{K_N}{\longleftrightarrow} C)}{C|\equiv(N \overset{K_N}{\longleftrightarrow} C)} \quad (\text{Goal G2})$$

D6: From assumption A4 and message M2 and by applying the message meaning rule, we derive the following:

$$\frac{N|\equiv(N\overset{ID_N}{\longleftrightarrow}C),C\triangleleft\langle^{X+1}CTR,ID_N,N\overset{OTP}{\longleftrightarrow}C\rangle_{\mathrm{N}\overset{ID_N}{\longleftrightarrow}\mathrm{C}}}{N|\equiv C|\sim(^{X+1}CTR,ID_N,N\overset{OTP}{\longleftrightarrow}C)}$$

D7: From assumption A5 and by applying the freshness rule, we derive the following:

$$\frac{N|\equiv\#(^{X+1}CTR)}{N|\equiv\#(^{X+1}CTR,ID_N,N\overset{OTP}{\longleftrightarrow}C)}$$

D8: From derivations D6 and D7 and by applying the nonce verification rule, we derive the following:

$$\frac{N|\equiv C|\sim(^{X+1}CTR,ID_N,N\overset{OTP}{\longleftrightarrow}C),N|\equiv\#(^{X+1}CTR,ID_N,N\overset{OTP}{\longleftrightarrow}C)}{N|\equiv C|\equiv(^{X+1}CTR,ID_N,N\overset{OTP}{\longleftrightarrow}C}$$

D9: From derivation D8 and by applying the belief rule, we derive the following:

$$\frac{N|\equiv C|\equiv(^{X+1}CTR,ID_N,N\overset{OTP}{\longleftrightarrow}C)}{N|\equiv C|\equiv(N\overset{OTP}{\longleftrightarrow}C)}\ (\text{Goal G3})$$

D10: From assumption A6 and derivation D9 and by applying the jurisdiction rule, we derive the following:

$$\frac{N|\equiv C\Rightarrow(N\overset{OTP}{\longleftrightarrow}C),N|\equiv C\equiv(N\overset{OTP}{\longleftrightarrow}C)}{N|\equiv(N\overset{OTP}{\longleftrightarrow}C)}\ (\text{Goal G4})$$

Hence, our authentication scheme achieves mutual authentication and key agreement between N and C.

5.4.3 Simulation based on AVISPA tool

In this section, we show the details and results of the simulation of our protocol when using the AVISPA tool.

5.4.3.1 Simulation details

We begin by determining the security goals for the simulation. Our objective is to ensure the secrecy of a number of values, namely TK12, ns, sec_IDN2, CTRn1, CTRn2, and so forth. In addition, we want to be certain that C and N authenticate each other successfully. Next, we write the HLPSL script for our authentication scheme. We define five roles: (1) role_C, which is played by the controller, (2) role_N1, which is played by the IoT device N1, (3) role_N2, which is played by the IoT device N2, (4) session, which defines the session role and all its declarations, and (5) environment, which instantiates all agents, variables, and functions and describes the simulation's security goals. In the remainder of this section, we briefly describe each role.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role role_N2 (N2, C,N1: agent,
Ka,Kb,Kd: public_key,IDN2,IDC,IDN1:text,
H:hash_func,
SND, RCV: channel (dy))
played_by N2 def=

local State : nat,
Ctrn2,Ctr1n2,Ctr2n2,OTPn2,Ctr3n2,Ns: text,
Kn2,TK12:symmetric_key,DIDN2,DIDN1:hash(text.text),Mn2:hash
(text.message.text.text.message),HMACn2,HMACn12:hash(message.text),Mc:hash
(message.text.text.text),HMACc:hash(message.text),M1c2:hash
(text.message.message),Mc2:hash(message.text.message),Mn12:hash
(message.message.message),PoC:hash
(text.text.message.message.text.message.message),M:message

init State := 0

transition

0. State = 0 /\ RCV(start) =|>
State':= 2 /\ Ctrn2' := new()/\Kn2':=new()/\DIDN2':=H(IDN2.Ctrn2')/\Mn2':=H
(IDC.DIDN2'.IDN2.Ctrn2'.Kn2') /\HMACn2':=H(Mn2'.Ctrn2')/\SND(IDC.DIDN2'.
{IDN2.Ctrn2'.Kn2'.HMACn2'}_Kb)
/\ secret(Ctrn2',ctrn2,{N2,C}) /\secret(IDN2,sec_idn2,{N2,C,N1})

2. State = 2 /\ RCV(DIDN2.IDC.{OTPn2'}_Kn2.HMACc') =|>
State':= 4 /\Ctr1n2':=new()/\Mc':=H(DIDN2.IDC.OTPn2'.Ctr1n2')/\HMACc':=H(Mc'.Ctr1n2')/
\request(C,N2,n2_c_ctr1n2,Ctr1n2)

4. State = 4 /\ RCV(DIDN2.IDC.{IDN1'.DIDN1'.TK12'}_Kn2.HMACc') =|> State':= 6 /\Mc2':=H
(DIDN2.IDC.{IDN1.DIDN1.TK12'}_Kn2)/\Ctr2n2':=new()/\HMACc':=H(Mc2'.Ctr2n2') /\Ns':=new
()/\secret(Ctr2n2,ctr2n2,{N2,C})/\Ctr3n2':=new()/\M1c2':=H(IDC.DIDN2.{Ns'}_Kn2)/
\HMACn2':=H(M1c2'.Ctr3n2')/\ SND(IDC.DIDN2.{Ns'}_Kn2.HMACn2') /\secret(Ns,ns,{N2,C,N1})/
\secret(TK12,tk12,{N2,C,N1})/\secret(Ctr3n2,ctr3n2,{N2,C})
6. State = 6 /\ RCV(DIDN2.DIDN1'.PoC'.{M'}_TK12.HMACn12') =|> State':= 8 /\ PoC':=H
(IDN2.IDN1.DIDN2.DIDN1.Ns.TK12.Kn2) /\Mn12':=H(DIDN1.DIDN2.PoC'.{M'}_TK12)/\HMACn12':=H
(Mn12'.Ns)/\request(N2,N1,n1_n2_poc,PoC)

end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 5.10: Role_N2

Fig. 5.10 shows the specification for role_N2 that is played by N2. Device N2 is

aware of all agents in the protocol (N2, C, and N1), the identities IDN1 and IDN2, which should be secured, the controller device identity (IDC), the public key of the controller Kb, the hash function H(·), and the send/receive channels Snd/Rcv. The (dy) notation indicates that the channels are following the Dolev-Yao model. At the first state "state 0," N2 receives a start message "Rcv(start)" as a signal to begin the protocol run. N2 creates and computes some authentication parameters to authenticate with C; it picks a fresh value as a nonce Ctrn2 to be used as an incremental counter that is shared with C. Next, it generates a symmetric key Kn2 that will be shared with C and used to secure the communication sessions. The computations for DIDN2, Mn2, and HMACn2 follow the description of our scheme. N2 sends (IDC.DIDN2'.{IDN2.Ctrn2'.Kn2'.HMACn2'}_Kb) to C after being encrypted with C's public key, as described in our scheme. For the second transition "State 2," N2 receives the (DIDN2.IDC.{OTPn2'}_Kn2.HMACc') message from C. The computations of DIDN2 and HMACc' follow the description of our scheme. For this transition, if HMACc' appears as expected by N2, N2 accepts the message and continues processing the message. N2 then stores OTPn2' to be used in the first data communication message, as explained in our scheme. For the third transition "State 4," N2 receives the (DIDN2.IDC.{IDN1'.DIDN1'.TK12'}_Kn2.HMACc') message from C, which is about the communication session between N1 and N2. At this point, if HMACc' appears as expected by N2, N2 accepts the message and resumes processing the message. Next, N2 picks a fresh value as nonce Ns to be used as a secret key in the HMAC between N1 and N2. N2 stores TK12, DIDN1, and Ns for the N1 and N2 session, as explained above in the protocol definition. The computations for M1c2 and HMACn2 follow the description of our scheme. N2 sends (IDC.DIDN2.{Ns'}_Kn2.HMACn2' to C after being encrypted using Kn2. For the fourth transition "State 6," N2 receives the

(DIDN2.DIDN1'.PoC'.{M'}_TK12.HMACn12') message from N1. Here, the validation of PoC and HMACn12 follows the description of our scheme. If these values are authentic, N2 decrypts the message and accepts the data. The "end role" at the end of the N2 role denotes the end of the role's definition.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role role_C(N2,C,N1: agent,
Ka,Kb,Kd: public_key,IDN2,IDC,IDN1:text,
H:hash_func,
SND, RCV: channel (dy))
played_by C def=

local State : nat,
Ctrn2,Ctr1n2,OTPn2,Ctr2n2,Ctr3n2,Ns,
Ctrn1,Ctr1n1,OTPn1, Ctr2n1,Ctr3n1: text,
Kn2,Kn1,TK12:symmetric_key,DIDN2,DIDN1,PoB:hash(text.text),Mn2,Mn1:hash
(text.message.text.text.message),HMACn2,HMACn1:hash(message.text),Mc:hash
(message.text.text.text),HMACc:hash(message.text),Mc1:hash
(message.text.message),Mc2:hash(message.text.message),M1c2:hash
(text.message.message),M1n1:hash(text.message.message.message),M2n1:hash
(message.text.message.message),PoC:hash(text.text.message.message.text.message.message)

init State := 1

transition

1. State = 1 /\ RCV(IDC.DIDN2'.{IDN2.Ctrn2'.Kn2'.HMACn2'}_Kb) =|>
State':= 3 /\ DIDN2':=H(IDN2.Ctrn2') /\ Mn2':=H(IDC.DIDN2'.IDN2.Ctrn2'.Kn2')/\HMACn2':=H
(Mn2'.Ctrn2') /\Ctr1n2':= new()/\OTPn2':=new() /\ Mc':=H(DIDN2'.IDC.OTPn2'.Ctr1n2')/
\HMACc':=H(Mc'.Ctr1n2') /\SND(DIDN2.IDC.{OTPn2'}_Kn2.HMACc')/\secret(IDN2,sec_idn2,
{N2,C,N1})
/\ secret(Ctr1n2',ctr1n2,{N2,C})
/\ witness(C,N2,n2_c_ctr1n2,Ctr1n2')
3. State = 3 /\ RCV(IDC.DIDN1.{IDN1'.{Ctrn1'}_inv(Kd).Kn1'.HMACn1'}_Kb) =|> State':=5 /\
DIDN1':=H(IDN1'.Ctrn1') /\ Mn1':=H(IDC.DIDN1'.IDN1'.Ctrn1'.Kn1') /\ HMACn1':=H
(Mn1'.Ctrn1') /\secret(Ctrn1,ctrn1,{C,N1})/\ Ctr1n1' := new()/\OTPn1':=new()/\Mc1':=H
(DIDN1'.IDC.{OTPn1'}_Kn1') /\ HMACc':=H(Mc1'.Ctr1n1') /\ SND(DIDN1'.IDC.{OTPn1'}
_Kn1.HMACc')/\secret(Ctr1n1,ctr1n1,{C,N1})/\witness(C,N1,n1_c_ctr1n1,Ctr1n1)

5. State = 5 /\ RCV(IDC.DIDN1'.PoB'.{IDN2'}_Kn1.HMACn1')=|> State':=7 /\Ctr2n1':=new() /
\M1n1':=H(IDC.DIDN1'.PoB'.{IDN2'}_Kn1) /\HMACn1':=H(M1n1'.Ctr2n1')/\secret
(Ctr2n1,ctr2n1,{C,N1})/\request(N1,C,n1_c_pob,PoB)/\TK12':=new()/\Mc2':=H(DIDN2.IDC.
{IDN1.DIDN1.TK12'}_Kn2)/\Ctr2n2':=new()/\HMACc':=H(Mc2'.Ctr2n2')/\ SND(DIDN2.IDC.
{IDN1.DIDN1.TK12'}_Kn2.HMACc')/\secret(TK12,tk12,{N2,C,N1})/\secret(Ctr2n2,ctr2n2,{N2,C})

7. State = 7 /\ RCV(IDC.DIDN2'.{Ns'}_Kn2.HMACn2') =|> State':=9 /\ Ctr3n2':=new() /
\M1c2':=H(IDC.DIDN2'.{Ns'}_Kn2)/\HMACn2':=H(M1c2'.Ctr3n2')/\secret(Ctr3n2,ctr3n2,{N2,C})
/\PoC':=H(IDN2.IDN1.DIDN2.DIDN1.Ns.TK12.Kn2)/\M2n1':=H(DIDN1.IDC.PoC'.{DIDN2.TK12.Ns}
_Kn1)/\Ctr3n1':=new() /\ HMACc':=H(M2n1'.Ctr3n1') /\ SND(DIDN1.IDC.PoC'.{DIDN2.TK12.Ns}
_Kn1.HMACc')/\secret(Ns,ns,{N2,C,N1})/\secret(TK12,tk12,{N2,C,N1})
%%

end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 5.11: Role_C

Fig. 5.11 shows the specification for the role played by C. C knows all other agents in the protocol (N2, C, and N1), the identities of IDN1 and IDN2, which should be secured, the controller device's identity (IDC), the public key of the controller Kb, the hash function H(·), and the send/receive channels Snd/Rcv. The remaining part of the specification describes the different states of the protocol execution involving Role_C.

Fig. 5.12 shows the specification of the role played by N1. Device N1 is aware of all agents in the protocol (i.e., N2, C, and N1), the identities of IDN1 and IDN2, which should be secured, the controller device's identity (IDC), the public key of the controller Kb, the

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role role_N1 (N2,C, N1: agent,
Ka,Kb,Kd: public_key, IDN2,IDC,IDN1:text,
H:hash_func,
SND, RCV: channel (dy))
played_by N1 def=

local State : nat,
Ctrn1,Ctr1n1,Ctr2n1,Ctr3n1,OTPn1,Ns:text,Kn1,TK12,Kn2:symmetric_key,
DIDN1,DIDN2,PoB:hash(text.text),Mn1:hash
(text.message.text.text.message),HMACn1,HMACn12:hash(message.text),Mc1:hash
(message.text.message),HMACc:hash(message.text),M1n1:hash
(text.message.message.message),M2n1:hash(message.text.message.message),Mn12:hash
(message.message.message),PoC:hash(text.text.message.message.text.message.message),
M:message
init State := 30

transition

30. State = 30 /\ RCV(start) =|>
State':= 32 /\ Ctrn1' := new()/\DIDN1':=H(IDN1.Ctrn1')/\Kn1':= new()/\Mn1':=H
(IDC.DIDN1'.IDN1.Ctrn1'.Kn1') /\HMACn1':=H(Mn1'.Ctrn1') /\SND(IDC.DIDN1.{IDN1.{Ctrn1'}
_inv(Kd).Kn1'.HMACn1'}_Kb)/\secret(Ctrn1',ctrn1,{C,N1})

32. State = 32 /\ RCV(DIDN1.IDC.{OTPn1'}_Kn1.HMACc') =|> State':=34 /\Ctr1n1':=new()/
\Mc1':=H(DIDN1.IDC.{OTPn1'}_Kn1) /\ HMACc':=H(Mc1'.Ctr1n1') /\secret(Ctr1n1,ctr1n1,
{C,N1})/\request(N1,C,n1_c_ctr1n1,Ctr1n1)/\ Ctr2n1':=new() /\ PoB':=H(OTPn1'.Ctr2n1')/\
M1n1':=H(IDC.DIDN1.PoB'.{IDN2}_Kn1) /\HMACn1':=H(M1n1'.Ctr2n1') /\ SND(IDC.DIDN1.PoB'.
{IDN2}_Kn1.HMACn1')/\secret(IDN2,sec_idn2,{N2,C,N1})/\secret(Ctr2n1,ctr2n1,{C,N1})/\
witness(N1,C,n1_c_pob,PoB')

34. State = 34 /\ RCV(DIDN1.IDC.PoC'.{DIDN2'.TK12'.Ns'}_Kn1.HMACc') =|> State':=36 /\
M2n1':=H(DIDN1.IDC.PoC'.{DIDN2'.TK12'.Ns'}_Kn1)/\Ctr3n1':=new() /\ HMACc':=H
(M2n1'.Ctr3n1')/\secret(TK12,tk12,{N2,C,N1})/\M':=new()/\Mn12':=H(DIDN1.DIDN2.PoC.{M'}
_TK12)/\HMACn12':=H(Mn12'.Ns)/\ SND(DIDN2.DIDN1.PoC.{M'}_TK12.HMACn12')/\secret(Ns,ns,
{N2,C,N1})/\witness(N1,N2,n1_n2_poc,PoC')


end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 5.12: Role_N1

hash function H(·), and the send/receive channels Snd/Rcv.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role session(N2,C,N1: agent,Ka,Kb,Kd: public_key,IDN2,IDC,IDN1:text, H:hash_func) def=

local SA, RA, SB, RB,SD, RD: channel (dy)

composition

role_N2(N2,C,N1,Ka,Kb,Kd,IDN2,IDC,IDN1,H,SA,RA)
/\ role_C(N2,C,N1,Ka,Kb,Kd,IDN2,IDC,IDN1,H,SB,RB)
/\ role_N1(N2,C,N1,Ka,Kb,Kd,IDN2,IDC,IDN1,H,SD,RD)


end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 5.13: Role_session

Fig. 5.13 shows the specification of the session role where all three agents' roles are invoked and where all the session's parameters are specified. First, all parameters and their declarations are presented. The predefined constants, which are Kb, IDN2, IDC, and IDN1, are included in this role. Second, each agent is assigned a send channel and a receive

channel under the local section. Under the composition section, the role is invoked for each agent, along with its constant values and functions. The N2 role is invoked with N2, C, and N1 as agents; Ka, Kb, and Kd as public keys; IDN2, IDC, and IDN1 as predefined constants; H as the hash function; and SA and RA as the send and receive channels, respectively, for N2. Similarly, the remaining roles are invoked in the same way.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
role environment() def=

const n2, c,n1 : agent,
ka,kb,kd, ki : public_key, idn2,idc,idn1:text,
h:hash_func,
ctrn2,ctr1n2,ns,sec_idn2,tk12,ctr2n2,ctr3n2,ctrn1,ctr1n1,ctr2n1,
n2_c_ctr1n2,n1_c_pob,n1_n2_poc,n1_c_ctr1n1: protocol_id

intruder_knowledge = {n2, c,n1,ka,kb,kd, ki, inv(ki)}

composition

session(n2,c,n1,ka,kb,kd,idn2,idc,idn1,h)

end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 5.14: Role_environment

Fig. 5.14 shows the specification of the environment role. In this role, one or more sessions are instantiated. At the beginning, all constants are instantiated and defined. The constants n2, c, and n1 instantiate the three agents N2, C, and N1, respectively. Here, ka, kb, and kd instantiate the Ka, Kb, and Kd keys, respectively. Also, idn2, idc, and idn1 instantiate IDN2, IDC, and IDN1, respectively. Next, h instantiates the hash function H. The protocol identifiers–ctrn2, ctr1n2, ns, sec_idn2, tk12, ctr2n2, ctr3n2, ctrn1, ctr1n1, ctr2n1, n2__ctr1n2, n1_c_pob,

n1_n2_poc, n1_c_ctr1n–are instantiated and defined as well. In the intruder knowledge section, all the relevant parameters that the intruder is assumed to know are provided prior to the execution. We assume that the intruder knows n2, c, and n1, as well as the hash h. In the composition section, the session is instantiated with the instances (n2,c,n1,ka,kb,kd,idn2,idc,idn1,h).

Fig. 5.15 shows the simulation goals that are defined under the “goal” keyword using

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
goal
secrecy_of ctrn2
secrecy_of ctr1n2
secrecy_of ns
secrecy_of sec_idn2
secrecy_of tk12
secrecy_of ctr2n2
secrecy_of ctr3n2
secrecy_of ctrn1
secrecy_of ctr1n1
secrecy_of ctr2n1
authentication_on n2_c_ctr1n2
authentication_on n1_c_pob
authentication_on n1_n2_poc
authentication_on n1_c_ctr1n1

end goal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 5.15: Goal

the protocol identifiers declared as "protocol_id." The simulator is told to check the secrecy of the following parameters: ctrn2, ctr1n2, ns, sec_idn2, tk12, ctr2n2, ctr3n2, ctrn1, ctr1n1, and ctr2n1. The authentication is performed using "n2_c_ctr1n2," " n1_c_pob," "n1_n2_poc," and "n1_c_ctr1n."

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
environment()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 5.16: Environment

Fig. 5.16 finally demonstrates the call of the main role "environment".

5.4.3.2 Simulation results

In this section, the simulation results of our proposed protocol are presented. The results are based on the two widely accepted AVISPA back-end model checkers: OFMC and CL-AtSe. The security protocol animator (SPAN) is employed to interactively construct a message sequence chart (MSC) of the protocol execution from the HLPSL specification outlined earlier. Moreover, SPAN automatically initiates attacks on the HLPSL specification using the Dolev-Yao intruder model. The SPAN protocol simulation's MSC corresponding to our HLPSL specification is shown in Fig. 5.17.

Fig. 5.18 demonstrates the protocol execution using SPAN animator software [4]. It

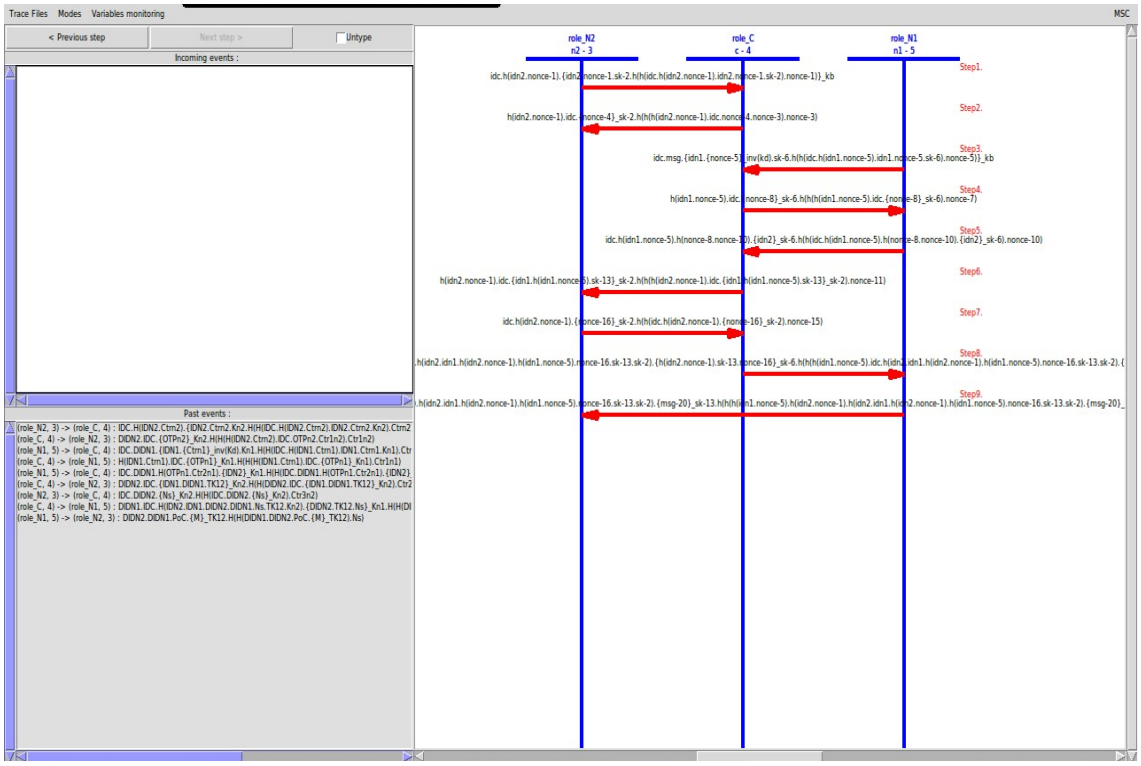

Figure 5.17: Snapshot of the protocol simulation in AVISPA

shows the full execution of our protocol under the intruder's control.

In the AVISPA tool, the security properties, such as authentication, integrity, and secrecy, are defined in a separate section. When SPAN is executed, it verifies if the protocol satisfies the specified properties. SPAN will generate the attack trace if any attack is found, and it will consider the protocol unsafe. Fig. 5.19 presents the CL-AtSe back-end checker report that assures that our scheme is safe and that is satisfies all the specified security goals.

Fig. 5.20 shows the OFMC back-end checker report, which states that our scheme is safe, thus meeting the specified security goals.

The other two model checkers—SATMC and TA4SP—reported "NOT SUPPORTED" and produced "INCONCLUSIVE" results. Our protocol includes XOR operations that are not supported by the TA4SP back-end checker.

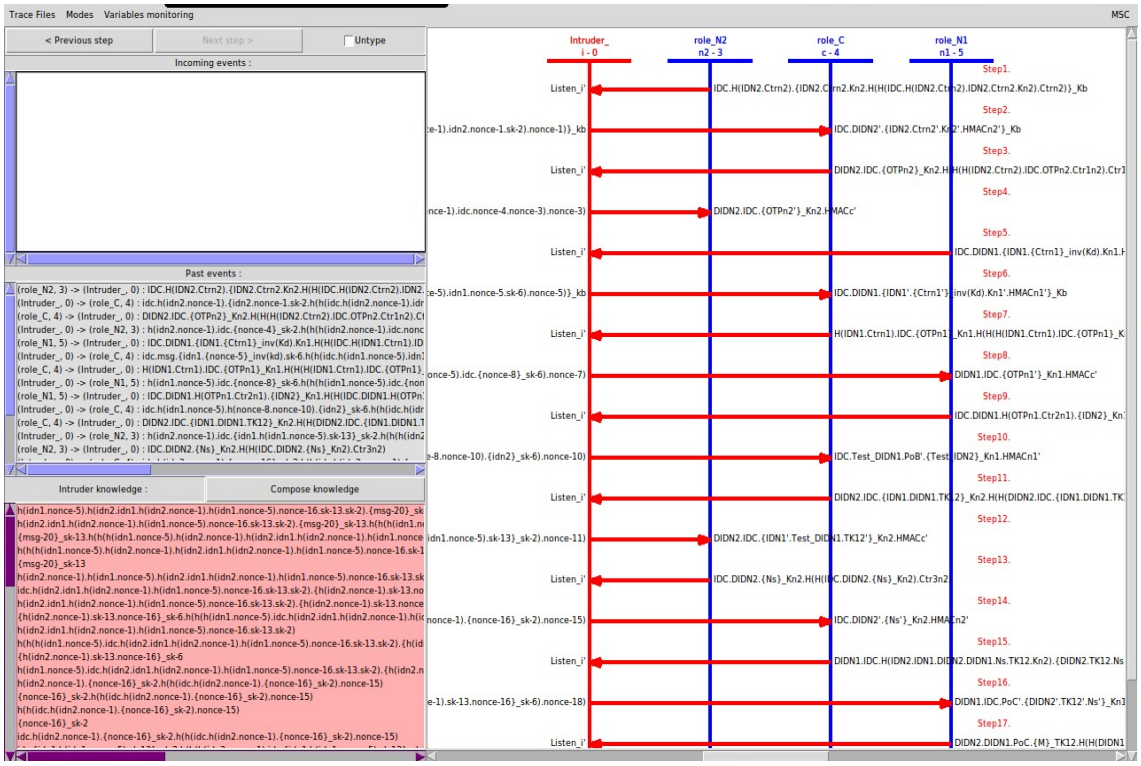

Figure 5.18: Snapshot of the protocol full execution using SPAN animator under the intruder's control

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /home/span/span/testsuite/results/ZB-v-4-11-5.if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 6 states
  Reachable  : 2 states
  Translation: 0.13 seconds
  Computation: 0.00 seconds
```

Figure 5.19: CL-AtSe summary report

```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /home/span/span/testsuite/results/ZB-v-4-11-5.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.06s
  visitedNodes: 16 nodes
  depth: 4 plies
```

Figure 5.20: OFMC summary report

5.5 Summary

The focus of this chapter was on introducing an authentication scheme for IoT systems that provides confidentiality, integrity, anonymity, unlinkability, and untraceability capabilities while achieving mutual authentication between the IoT and controller devices. The security of the proposed scheme was formally verified using the BAN logic and through simulation using the AVISPA toolset, and the results show that our scheme is safe. In addition, we conducted an informal security analysis of our scheme, showing that our scheme resists common attacks. Furthermore, we evaluated the efficiency of the scheme and compared it with other related schemes.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The Internet of Things (IoT) plays an important role in all facets of our daily lives. It benefits different fields, including healthcare, industrial appliances, sports, homes, etc. The IoT consists of billions of devices connected together over the internet that are able to gather and exchange data using IoT nodes and controllers.

The number of IoT devices is increasing at a very fast pace. It has been predicted by Gartner that the number of IoT devices will reach 20.4 billion by 2020 [17] [41] and 25 billion by 2025 [20]. The anticipated fast growth of IoT devices has led to serious security concerns for networks [18].

Cyber attackers are shifting their attention from traditional computers to IoT devices for malignant activities like exposing smart homeowner private information and/or to launch botnet attacks. Like for conventional networks, the security of IoT networks rests on how properly the authentication process is done. However, unlike conventional networks, the IoT infrastructure faces an uphill battle in deploying and operating strong authentication schemes

because of inherent limitations on the underlying storage and computation capability.

The first contribution of this work is the introduction of a secure lightweight and key exchange protocol for various devices in the IoT network using a fog computing-based smart home network model. We proposed a secure method based on network segregation and nodes' capabilities to mitigate insider threats. Moreover, the security of the proposed scheme is augmented using dynamic identities and temporary secret session keys that change in every session, and are exchanged in an unlinkable manner, improve the security of the proposed scheme.

The second contribution is the introduction of a mechanism to enforce the security policy during the authentication phase based on different methods, such as division of devices using virtual domain segregation in IoT smart home network to mitigate the outsider and insider threats. A new architecture based on fog computing was proposed to provide and support the identity assurance of IoT devices. This modified architecture was aimed at preventing identity theft (spoofing and masquerading attacks) by securing the real IoT nodes' identities and make them known only by trusted players. We also proposed using a cumulative Keyed-hash chain a challenge-response mechanism which, is aimed at improving IoT security. Finally, we proposed an authentication scheme for IoT systems that provides confidentiality, integrity, anonymity, unlinkability, and untraceability properties while achieving mutual authentication between IoT devices and controller devices.

Furthermore, through the rigorous formal and informal security analysis of our protocols using the BAN logic and AVISPA tool, we show that our protocols are resilient against known attack methods. Besides, our protocols achieved the key security properties (e.g., anonymity, unlinkability) with a relatively limited performance overhead. Finally, we compared our protocols with other proposed protocols and showed that our protocols are in general more

efficient than recently proposed protocols.

6.2 Future Work

Our plans for future work include the following:

- We will extend the proposed protocol suite to consider cases where the IoT node leaves one particular home network and joins another network.
- We will allow an IoT device in one home network to communicate with an IoT device in another home network regardless of the underlying communication protocols.
- We will also explore how to reinforce the current capability of our scheme to thwart impersonation by adapting continuous authentication schemes, such as the approach proposed by Tsai et al. [55]. In their work [55], Tsai et al. presented a passive continuous authentication system based on physiological and soft biometrics technologies, namely face recognition and clothes'color recognition, using interactive artificial bee colony algorithm. In the system, face recognition is considered the key component to controlling the authentication process, while soft biometric is seen as a supporting factor to overcome and remedy any potential security breach, such as account hijacking occurring while the user is temporarily away from the device.
- We will address or mitigate the impact of the key constraints of the proposed schemes, which is its reliance on a too-powerful controller, where the controller knows all the devices'sensitive data. Under these settings, as soon as the controller is compromised, an intruder can gain knowledge of all sensitive data and then can perform any malicious actions.

Bibliography

[1] N. Abbas, M. Asim, N. Tariq, T. Baker, and S. Abbas. A mechanism for securing iot-enabled applications at the fog layer. *Journal of Sensor and Actuator Networks*, 8(1):16, 2019.

[2] M. Abomhara et al. Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks. *Journal of Cyber Security and Mobility*, 4(1):65–88, 2015.

[3] B. Anggorojati, P. N. Mahalle, N. R. Prasad, and R. Prasad. Capability-based access control delegation model on the federated iot network. In *The 15th International Symposium on Wireless Personal Multimedia Communications*, pages 604–608. IEEE, 2012.

[4] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, et al. The avispa tool for the automated validation of internet security protocols and applications. In *International Conference on Computer Aided Verification*, pages 281–285. Springer, 2005.

[5] K. Ashton. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[6] U. Blumenthal, F. Maino, and K. McCloghrie. The advanced encryption standard (AES) cipher algorithm in the snmp user-based security model. *Internet Proposed Standard RFC*, 3826, 2004.

[7] F. Bonomi. Connected vehicles, the internet of things, and fog computing. In *The eighth ACM International Workshop on Vehicular Inter-networking (VANET), Las Vegas, USA*, pages 13–15, 2011.

[8] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, pages 13–16. ACM, 2012.

[9] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 426(1871):233–271, 1989.

[10] B.-C. Choi, S.-H. Lee, J.-C. Na, and J.-H. Lee. Secure firmware validation and update for consumer devices in home networking. *IEEE Transactions on Consumer Electronics*, 62(1):39–44, 2016.

[11] F. Chu, R. Zhang, R. Ni, and W. Dai. An improved identity authentication scheme for internet of things in heterogeneous networking environments. In *2013 16th International Conference on Network-Based Information Systems*, pages 589–593. IEEE, 2013.

[12] I. E. Commission. Iso/iec 27002: 2005. In *Information technology–Security techniques–Code of Practice for Information Security Management*. International Organization for Standardization, 2005.

[13] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[14] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram. Blockchain for iot security and privacy: The case study of a smart home. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 618–623. IEEE, 2017.

[15] G. Fortino, C. E. Palau, A. Guerrieri, N. Cuppens, F. Cuppens, H. Chaouchi, and A. Gabillon. Interoperability, safety and security in iot, 2018.

[16] P. P. Gaikwad, J. P. Gabhane, and S. S. Golait. 3-level secure kerberos authentication for smart home systems using iot. In *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, pages 262–268. IEEE, 2015.

[17] Gartner. Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016. `https://urlzs.com/LCEHn`. Accessed: 2019-06-30.

[18] A. Giaretta, S. Balasubramaniam, and M. Conti. Security vulnerabilities and countermeasures for target localization in bio-nanothings communication networks. *IEEE Transactions on Information Forensics and Security*, 11(4):665–676, 2015.

[19] K. Gill, S.-H. Yang, F. Yao, and X. Lu. A zigbee-based home automation system. *IEEE Transactions on Consumer Electronics*, 55(2):422–430, 2009.

[20] A. GSMA. The mobile economy (2013). *White Paper*, 2015.

[21] S. Gusmeroli, S. Piccione, and D. Rotondi. A capability-based security approach to manage access control in the internet of things. *Mathematical and Computer Modelling*, 58(5-6):1189–1205, 2013.

[22] J. L. Hernandez-Ramos, M. P. Pawlowski, A. J. Jara, A. F. Skarmeta, and L. Ladid. Toward a lightweight authentication and authorization framework for smart objects. *IEEE Journal on Selected Areas in Communications*, 33(4):690–702, 2015.

[23] R. Hummen, H. Shafagh, S. Raza, T. Voig, and K. Wehrle. Delegation-based authentication and authorization for the ip-based internet of things. In *2014 Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 284–292. Ieee, 2014.

[24] S. Khan, S. Parkinson, and Y. Qin. Fog computing security: a review of current applications and security solutions. *Journal of Cloud Computing*, 6(1):19, 2017.

[25] H. H. Kilinc and T. Yanik. A survey of sip authentication and key agreement schemes. *IEEE Communications Surveys & Tutorials*, 16(2):1005–1023, 2013.

[26] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[27] H. Krawczyk, R. Canetti, and M. Bellare. Hmac: Keyed-hashing for message authentication. *Network Working Group, RFC2104, Category: Informational*, 1997.

[28] N. Kshetri. Can blockchain strengthen the internet of things? *IT Professional*, 19(4):68–72, 2017.

[29] P. Kumar, A. Gurtov, J. Iinatti, M. Ylianttila, and M. Sain. Lightweight and secure session-key establishment scheme in smart home environments. *IEEE Sensors Journal*, 16(1):254–264, 2015.

[30] J.-Y. Lee, W.-C. Lin, and Y.-H. Huang. A lightweight authentication protocol for internet of things. In *2014 International Symposium on Next-Generation Electronics (ISNE)*, pages 1–2. IEEE, 2014.

[31] J. Liu, Q. Li, R. Yan, and R. Sun. Efficient authenticated key exchange protocols for wireless body area networks. *EURASIP Journal on Wireless Communications and Networking*, 2015(1):188, 2015.

[32] A. Lohachab. Ecc based inter-device authentication and authorization scheme using mqtt for IoT networks. *Journal of Information Security and Applications*, 46:1–12, 2019.

[33] Y. Lu and L. Da Xu. Internet of things (iot) cybersecurity research: a review of current research topics. *IEEE Internet of Things Journal*, 6(2):2103–2115, 2018.

[34] P. Luana. The iot threat landscape and top smart home vulnerabilities in 2018. 2018.

[35] B. Lynn. The pairing-based cryptography library. `https://crypto.stanford.edu/pbc/`. Accessed: 2019-06-30.

[36] B. Lynn. *On the implementation of pairing-based cryptosystems*. PhD thesis, Stanford University Stanford, California, June 2007.

[37] J. A. Manrique, J. S. Rueda-Rueda, and J. M. Portocarrero. Contrasting internet of things and wireless sensor network from a conceptual overview. In *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 252–257. IEEE, 2016.

[38] D. G. Marks. Inference in MLS database systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):46–55, 1996.

[39] D. Mishra, P. Vijayakumar, V. Sureshkumar, R. Amin, S. H. Islam, and P. Gope. Efficient authentication protocol for secure multimedia communications in iot-enabled wireless sensor networks. *Multimedia Tools and Applications*, 77(14):18295–18325, 2018.

[40] S. R. Moosavi, T. N. Gia, A.-M. Rahmani, E. Nigussie, S. Virtanen, J. Isoaho, and H. Tenhunen. Sea: a secure and efficient authentication and authorization architecture for iot-based healthcare using smart gateways. *Procedia Computer Science*, 52:452–459, 2015.

[41] Nokia. Nokia networks to power internet of things with 5G connectivity. `https://urlzs.com/55pv4`. Accessed: 2019-06-30.

[42] G. Park, B. Kim, and M.-s. Jun. A design of secure authentication method using zero knowledge proof in smart-home environment. In *Advances in Computer Science and Ubiquitous Computing*, pages 215–220. Springer, 2016.

[43] A. Pfitzmann and M. Köhntopp. Anonymity, unobservability, and pseudonymity—a proposal for terminology. In *Designing Privacy Enhancing Technologies*, pages 1–9. Springer, 2001.

[44] P. Porambage, C. Schmitt, P. Kumar, A. Gurtov, and M. Ylianttila. Two-phase authentication protocol for wireless sensor networks in distributed iot applications. In *2014 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 2728–2733. IEEE, 2014.

[45] N. A. M. Risalat, M. T. Hasan, M. S. Hossain, and M. M. Rahman. Advanced real time rfid mutual authentication protocol using dynamically updated secret value through encryption and decryption process. In *2017 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pages 788–793. IEEE, 2017.

[46] F. K. Santoso and N. C. Vun. Securing iot for smart home system. In *2015 International Symposium on Consumer Electronics (ISCE)*, pages 1–2. IEEE, 2015.

[47] N. Saputro, A. I. Yurekli, K. Akkaya, and A. S. Uluagac. Privacy preservation for iot used in smart buildings. In *Security and Privacy in Internet of Things (IoTs)*, pages 155–186. CRC Press, 2016.

[48] V. Shivraj, M. Rajan, M. Singh, and P. Balamuralidhar. One time password authentication scheme based on elliptic curves for internet of things (iot). In *2015 5th National Symposium on Information Technology: Towards New Smart World (NSITNSW)*, pages 1–6. IEEE, 2015.

[49] C. Simko. Iot security starts with identity. `https://www.globalsign.com/en/blog/iot-security-starts-with-device-identity/`. Accessed: 2019-06-30.

[50] STMicroelectronics. Cortex-m3 mcu with 512 kbytes flash, 72 mhz cpu. 2018. `www.st.com/en/microcontrollers/stm32f103ve.html`. Accessed: 2019-06-30.

[51] L. Tanczer, I. Brass, M. Elsden, M. Carr, and J. J. Blackstock. The united kingdom's emerging internet of things (iot) policy landscape. *Tanczer, LM, Brass, I., Elsden, M., Carr, M., & Blackstock, J.(2019). The United Kingdom's Emerging Internet of*

Things (IoT) Policy Landscape. In R. Ellis & V. Mohan (Eds.), Rewired: Cybersecurity Governance, pages 37–56, 2019.

[52] M. Taylor, D. Reilly, and B. Lempereur. An access control management protocol for internet of things devices. *Network Security*, 2017(7):11–17, 2017.

[53] C. Technologies. Insider threat 2018 report. `https://www.ca.com/content/dam/ca/us/files/ebook/insider-threat-report.pdf`. Accessed: 2019-06-30.

[54] A. Tomar. Introduction to zigbee technology. *Glob. Technol. Centre*, 1:1–24, 2011.

[55] P.-W. Tsai, M. K. Khan, J.-S. Pan, and B.-Y. Liao. Interactive artificial bee colony supported passive continuous authentication system. *IEEE Systems Journal*, 8(2):395–405, 2012.

[56] A. Ukil, S. Bandyopadhyay, A. Bhattacharyya, and A. Pal. Lightweight security scheme for vehicle tracking system using CoAP. In *Proceedings of the International Workshop on Adaptive Security*, page 3. ACM, 2013.

[57] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[58] L. Viganò. Automated security protocol analysis with the avispa tool. *Electronic Notes in Theoretical Computer Science*, 155:61–86, 2006.

[59] J. Voris, N. Saxena, and T. Halevi. Accelerometers and randomness: Perfect together. In *Proceedings of the Fourth ACM Conference on Wireless Network Security*, pages 115–126. ACM, 2011.

[60] A. Waqar, A. Raza, H. Abbas, and M. K. Khan. A framework for preservation of cloud users' data privacy using dynamic reconstruction of metadata. *Journal of Network and Computer Applications*, 36(1):235–248, 2013.

[61] J. Wen, M. Zhang, and X. Li. The study on the application of ban logic in formal analysis of authentication protocols. In *Proceedings of the 7th International Conference on Electronic Commerce*, pages 744–747. ACM, 2005.

[62] P. Wilson. Inter-device authentication protocol for the internet of things. Master's thesis, The University of Victoria, Department of Electrical and Computer Engineering, May 2017.

[63] C.-C. Wu, W.-B. Lee, and W.-J. Tsaur. A secure authentication scheme with anonymity for wireless communications. *IEEE Communications Letters*, 12(10):722–723, 2008.

[64] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, and Y. Jin. Security analysis on consumer and industrial iot devices. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 519–524. IEEE, 2016.

[65] N. Ye, Y. Zhu, R.-c. Wang, R. Malekian, and L. Qiao-Min. An efficient authentication and access control scheme for perception layer of internet of things. *Applied Mathematics & Information Sciences*, 8(4):1617, 2014.

[66] H. Yüzugüzel, J. Niemi, S. Kiranyaz, M. Gabbouj, and T. Heinz. Shakeme: Key generation from shared motion. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 2130–2133. IEEE, 2015.

[67] G. Zhao, X. Si, J. Wang, X. Long, and T. Hu. A novel mutual authentication scheme for internet of things. In *Proceedings of 2011 International Conference on Modelling, Identification and Control*, pages 563–566. IEEE, 2011.