

**PRIVACY PRESERVING IDENTIFICATION IN HOME
AUTOMATION SYSTEMS**

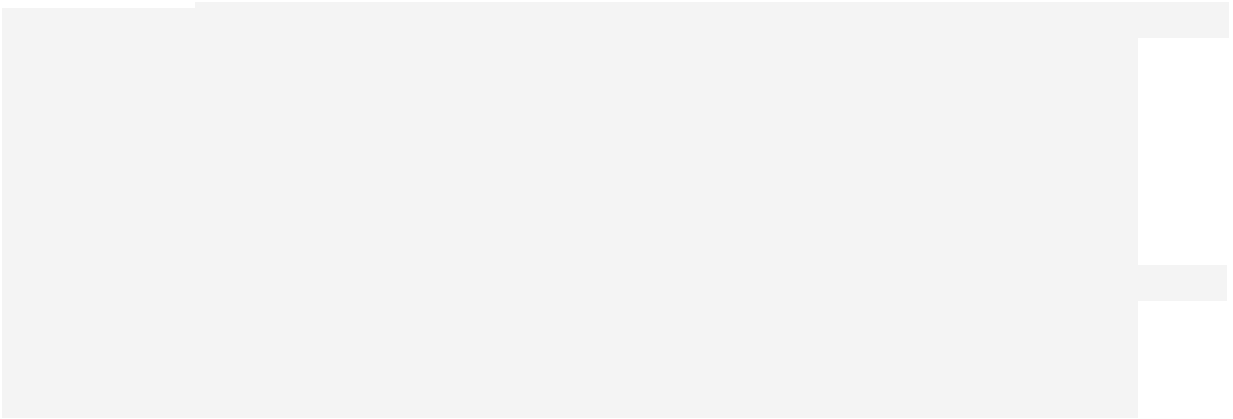
by
ŞEVVAL ŞİMŞEK

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science

Sabancı University
July 2021

**PRIVACY PRESERVING IDENTIFICATION IN HOME
AUTOMATION SYSTEMS**

Approved by:



Date of Approval: July 13, 2021

ŞEVVAL ŞİMŞEK 2021 ©

All Rights Reserved

ABSTRACT

PRIVACY PRESERVING IDENTIFICATION IN HOME AUTOMATION SYSTEMS

ŞEVVAL ŞİMŞEK

Computer Science and Engineering M.S. Thesis, July 2021

Thesis Supervisor: Prof. Albert Levi

Keywords: IoT, Home Automation Systems, Privacy, Authentication, Identification

Home Automation System (HAS) is a set of interconnected devices in a household that are accessible via the Internet. Homeowners can monitor and control the smart home appliances using HAS, but if the system is not structured properly, intruders can gain access to one or more of the devices in the system. In this thesis, we propose a privacy preserving identification model for HAS. In this model, first, a secure key sharing and credential issuance protocol is presented. In this protocol Idemix's Verifiable Encryption scheme is implemented. Verifiable encryption schemes ensure the authenticity of the Issuer and the User, while keeping the communication fully secure. During this protocol, the master key for the mutual verification and authentication protocol is shared and the credentials for the User are issued. Then, we explain the mutual verification and authentication protocol, which also employs the HAS Management System (HMS) as a trusted party to obfuscate the communication between the Vendors and Innovative Home Gateway (IHG)'s. We preserve the privacy of the homeowners by masking the brands, types and id's of the appliances inside the household. HMS carries on two different conversations, one with Vendor and the other with IHG, replacing the UUID's in the topics. The topics and messages published on these are unlinkable, thus masking the identity of the IoT devices connected to the IHG. The performance tests are performed using 4 different scenarios. Moreover, security analysis for both key sharing and credential issuance protocol and for the mutual verification and authentication protocol are given, and they are both proven to be secure according to OFMC and ATSE specifications. In conclusion, the system is scalable for actual implementation, and provides security and privacy as proposed.

ÖZET

EV OTOMASYON SİSTEMLERİNDE GİZLİLİK KORUYUCU TANIMLAMA

ŞEVVAL ŞİMŞEK

Bilgisayar Mühendisliği Yüksek Lisans Tezi, Temmuz 2021

Tez Danışmanı: Prof. Dr. Albert Levi

Anahtar Kelimeler: Nesnelerin İnterneti, Ev Otomasyon Sistemleri, Mahremiyet, Kimlik Doğrulama

Ev otomasyon sistemleri (HAS - Home Automation System), bir evde bulunan ve İnternet üzerinden erişilebilen birbirine bağlı bir dizi cihazdan oluşan sistemlerdir. Ev sahipleri, akıllı ev aletlerini HAS'ı kullanarak izleyebilir ve kontrol edebilir, ancak sistem düzgün yapılandırılmamışsa, izinsiz giriş yapanlar sistemdeki bir veya daha fazla cihaza erişim sağlayabilir. Bu tezde, HAS için gizliliği koruyan bir tanımlama modeli önerilmektedir. Bu modelde öncelikle güvenli bir anahtar paylaşımı ve kimlik bilgisi verme protokolü sunulmaktadır. Bu protokolde Idemix'in Doğrulanabilir Şifreleme şeması uygulanmaktadır. Doğrulanabilir şifreleme şemaları, iletişimi tamamen güvenli tutarken, kimlik atama otoritesinin ve kullanıcının kimliğinin doğrulanmasını sağlar. Bu protokol sırasında, karşılıklı kimlik doğrulama protokolünün ana anahtarı paylaşılır ve kullanıcı için kimlik bilgileri verilir. Ardından, Sağlayıcı Firmalar ve Yenilikçi Ev Ağ Geçidi (IHG - Innovative Home Gateway)'ler arasındaki iletişimi gizlemek için güvenilir bir taraf olarak HAS Yönetim Sistemini (HMS - HAS Management System)'i kullanan karşılıklı kimlik doğrulama protokolü açıklanmaktadır. Ev içindeki aletlerin marka, tip ve kimliklerini maskeleyerek ev sahiplerinin mahremiyeti korunur. HMS, konulardaki UUID'leri değiştirerek biri Sağlayıcı Firma ve diğeri IHG ile olmak üzere iki farklı görüşme gerçekleştirir. Bunlarda yayınlanan konular ve mesajlar birbiri ile bağlantılı değildir, bu şekilde IHG'ye bağlı IoT cihazlarının kimlikleri gizlenir. Tezde ayrıca 4 farklı senaryo için yapılan testlerin performans sonuçları sunulmaktadır. Tezin sonunda hem anahtar paylaşımı hem de kimlik bilgisi düzenleme protokolü ile karşılıklı kimlik doğrulama protokolü için güvenlik analizleri verilmiş ve her ikisinin de OFMC ve ATSE şartnamelerine göre güvenli olduğu kanıtlanmıştır. Sonuç olarak, sistemin gerçek uygulamalar için

ölçeklenebilir olduđu ve önerildiđi gibi güvenlik ve gizlilik sağladıđı biçimsel olarak gösterilmiştir.

ACKNOWLEDGEMENTS

Throughout the writing of this thesis I have received a great amount of support and assistance from many people.

I would like to first express my gratitude to my thesis supervisor, Prof. Albert Levi, for his support through my Master's thesis as well as my academic life. His valuable feedbacks and guidance throughout my research improved my knowledge in many ways.

Also, I would like to extend my sincere thanks to the thesis jury members Assoc. Prof. Cemal Yılmaz and Asst. Prof. Kübra Kalkan Çakmakçı, for their time and valuable feedbacks.

In addition, I would like to send many thanks to my friend and project partner Simge Demir for her support. It was a blessing to work with someone so talented and kind throughout my Master's degree.

This work has been partially supported by TÜBİTAK (Scientific and Technological Research Council of Turkey) under grant 117E017 and I would like to thank TÜBİTAK members for their support to our project.

Last but not least, I am sending special thanks to my family, especially to my dear husband M. Enes, for his continued support during my academic life for the last two years, and for believing in me and my goals more than I ever did. This journey has become much easier with you by my side.

TABLE OF CONTENTS

LIST OF TABLES	ix
LIST OF FIGURES	x
1. INTRODUCTION	1
2. SYSTEM BASICS	5
2.1. Literature Review	5
2.2. System Parts	8
2.2.1. IoT Devices	8
2.2.2. Innovative Home Gateway (IHG)	8
2.2.3. HAS Management System (HMS).....	9
2.2.4. Vendors and Homeowners	9
2.3. Idemix	10
2.4. MQTT	15
2.5. Attacker Model	15
3. SECURE KEY SHARING AND CREDENTIAL ISSUANCE	17
3.1. Zero Knowledge Proofs	17
3.2. Verifiable Encryption Scheme	18
3.3. Key Sharing and Credential Issuance	19
4. PRIVACY PRESERVING MUTUAL AUTH. PROTOCOL	21
4.1. Protocol Steps	21
4.1.1. Update Protocol.....	22
4.1.2. Error Report Protocol.....	26
5. PERFORMANCE EVALUATION	29
5.1. Setup and Implementation.....	29
5.2. Performance Evaluation Results	32
5.2.1. Successful Update Scenario	32

5.2.2. Failed Update Scenario.....	34
5.2.3. Error Report Scenario.....	37
5.3. Discussions.....	39
6. SECURITY ANALYSIS.....	43
6.1. Security Analysis of the Credential Issuance Protocol.....	44
6.2. Security Analysis of the Mutual Authentication Protocol.....	45
6.3. Discussion.....	49
7. CONCLUSION.....	50
BIBLIOGRAPHY.....	51

LIST OF TABLES

Table 2.1. Symbols used in protocol description	6
Table 5.1. IoT device features of the testbed	30
Table 5.2. IHG features of the testbed	30
Table 5.3. HMS/MQTT server features of the testbed	31
Table 5.4. Vendor features of the testbed	31
Table 5.5. Average computational results for successful update	33
Table 5.6. Average latencies for successful update w.r.t different file sizes .	34
Table 5.7. Average computational results for failed update	35
Table 5.8. Avg. latencies for successful update w.r.t. different file sizes....	36
Table 5.9. Average results for mixed successful and failed update.....	37
Table 5.10. Average calculation results for error report	38

LIST OF FIGURES

Figure 2.1. Overview of the Home Automation System architecture [1]....	10
Figure 2.2. Sample credential issued to an IoT device	13
Figure 2.3. Sample ProofSpec of an IoT device	14
Figure 3.1. Key Sharing and Credential Issuance protocol	20
Figure 4.1. Outline of the Mutual Authentication Process (Downstream) .	25
Figure 4.2. Outline of the Mutual Authentication Process (Upstream)	28
Figure 5.1. Successful update performance results	33
Figure 5.2. Failed update performance results	35
Figure 5.3. Average end to end latency of receiving error reports	38
Figure 5.4. Error Report test performance comparison	40
Figure 5.5. Successful Update test performance comparison	41
Figure 5.6. Failed Update test performance comparison	41
Figure 5.7. Mixed Update test performance comparison	42
Figure 6.1. Initialization protocol simulation using SPAN tool.....	43
Figure 6.2. Initialization protocol intruder simulation using SPAN tool ...	44
Figure 6.3. SPAN simulation of the proposed protocol.....	46
Figure 6.4. SPAN Intruder simulation for the proposed protocol	47
Figure 6.5. Key sharing protocol safety analysis	48
Figure 6.6. Mutual verification and authentication protocol safety analysis	48

1. INTRODUCTION

Internet of Things (IoT) devices are small but capable devices that collect, store and share data periodically. IoT devices are “things” or physical devices that are equipped with sensors, software and hardware appropriate for their usage purposes. IoT devices are used in a variety of fields, from smart homes to medical care and even military applications. These devices can communicate with other devices and each other, over a network, which can be WiFi, Bluetooth or many other protocols of communication. When these protocols are not carefully designed and implemented, these smart devices can disclose information about the user, or completely give away the control to an adversary [2]. IoT devices have low processing powers and network bandwidth, which makes it difficult to implement complex security mechanisms, hence IoT devices are more vulnerable to manipulation and attacks [3]. Since one of the fields that IoT is most commonly used is smart homes and home automation systems, the privacy of the homeowners becomes of the utmost importance.

A Home Automation System (HAS) is a set of connected devices in a household, often used to control and monitor home features such as air conditioning, lighting or security systems. The devices included in HAS continuously collect data that can be vulnerable, or should be kept private such as the frequency of the usage, updates that it receives and reports that it sends back to the provider. All of these data are considered private, and they should be transferred in a secure and private manner. Any information that leaks from a communication either between the user and the IoT device, or between the provider and the IoT device, threatens the confidentiality of the transmitted information. For example, a third party learning that the homeowner has a specific type of device, can make the household a target for a known vulnerability of that specific device.

In this thesis, we present a privacy preserving secure identification and authentication protocol for HAS. The proposed system originates from the HAS architecture of Batalla and Gonciarz [4], which was developed for the FUSE (Full-Managed Secure Home Automation Systems) project, developed in cooperation by the Turkish and Polish teams. We used this scheme as the base of our approach, including the IHG

structure and HAS Management system (HMS) structure. In the privacy preserving identification scheme we propose in this thesis, IHG is the gateway to manage the communication between IoT devices in the home network and outside users, through HMS. Every message incoming from the outside users must be relayed to the IHG from HMS. This way we ensure that only authorized users reach the IHG, and even then if they are not authenticated they cannot receive or send messages.

The preliminary versions of this study was presented in [1] and in [5]. The first part of the project is presented in the thesis by S. Demir [5], where the privacy aware system is introduced by implementing fakeProofs. This way, the actual update communication becomes indistinguishable from the failed update case, in terms of structure and length of the messages. However, this system does not include encryption, secure key sharing or HMS as a topic switching middleware. The messages are sent over the topic determined by the parties in plaintext, and the topic includes private information such as device brand and type. Here in this thesis, we extend the previous work in several aspects, and for a better understanding the entire work is presented, but the new contributions are discussed in more detail. In the previous model, the entities were the same as the current model, but the extent of their roles changed and broadened. In the proposed privacy preserving model, we change the following aspects:

- 1.1 All of the communications between all of the entities are now encrypted. During the issuance process, the conversation is no longer in plain text. We used Idemix verifiable encryption scheme to ensure the parties are legitimate and the credentials are completely secret.
- 1.2 During the privacy preserving mutual authentication process, and at the end during update and error reports scenarios, all of the conversations between entities are encrypted with a symmetric key that is shared along with the credentials.
- 1.3 Secure key sharing is ensured, using the verifiable encryption scheme [6]. Trusted secure party (HMS, or Issuer) distributes the master keys to every IoT device and Vendor.
- 1.4 Privacy preserving addition to the preliminary version is introduced. In the preliminary works [1],[5], privacy awareness was provided by using fakeProof's, in order to ensure that the usual update scenario and rejected update scenario are indistinguishable from each other for anyone listening to the channel. On top of this privacy preserving mechanism, we add a middleware layer between the Vendors and IoT devices, which is the HMS. HMS breaks the link of the

two communications (between IoT and HMS, and between Vendor and HMS) and the two cannot be linked in any way, since the topics are changed and no information is related between these two communications.

According to these enhancements, the roles of the entities are updated, too. HMS acted as the Issuer and distributed the credentials to the IoT devices and Vendors. HMS also had the role of the MQTT [7] broker, and managed the communication between Vendors and IoT Devices, without changing anything in the messages. On top of these roles, HMS now works as a trusted party that communicates with the entities directly, where it decrypts the incoming message using the sender's key, modifies the topic, and encrypts the new message with the receiver's key. The two messages differ in two ways, first the topics are different since the UUID's [8] are replaced, and secondly the message body is encrypted with a different key. Hence, there exists no connection between the message that HMS receives and the message that HMS relays.

Performance of the proposed model is measured thoroughly for different scenarios. As mentioned before, HMS manages a two way traffic, and both incoming and outgoing messages are encrypted. Since IoT devices have low processing power and small bandwidth, the protocol must be as lightweight as possible, in order not to interfere with the regular duties of the IoT devices. Likewise, Vendors may have more powerful systems on their end but they handle multiple communications at once. When applied to real life smart homes, the number of communications handled at once can be very big. Vendors can limit this number; however, if the end-to-end latency is reduced as much as possible, they can work more efficiently. Even though HMS does not operate the same as IoT devices and Vendors, it processes vast amount of incoming messages, decrypts these with the sender's key, matches with the corresponding topic and relays the message encrypted with the receiver's key. We conducted a number of test cases, with different number of IoT devices, and different data sizes to infer the scalability of the system to real life smart home systems. The detailed results can be found in Chapter 5.

Lastly, we provide a security analysis of the proposed protocol, using the SPAN [9] tool for Avispa [10] security analyzer. First, we test our key sharing and credential issuance protocol, with the goal of authentication of Issuer and IoT devices, and secrecy of the master key. Our key sharing and credential issuance protocol is evaluated to be secure according to OFMC [11] and ATSE [12] specifications. We also tested our privacy preserving mutual verification and authentication protocol, with the goal of secrecy of the message between the Vendor and IoT device. This protocol is also evaluated to be secure according to OFMC and ATSE specifications.

The detailed security analysis and intruder simulations can be found in Chapter 6.

Our results show that the HAS structure developed and enhanced in this thesis is secure, privacy preserving and scalable for real life applications. Our test results with 11 devices in the testbed show that when the number of devices actively receiving updates or sending error messages at the same time, the end-to-end latency increases. This increase in the latency is feasible and acceptable. We also show that encrypting the conversation, and enhancing privacy preservation by adding HMS to switch topics does not increase the end-to-end latency dramatically, hence it is reasonable to carry out the protocol in a fully encrypted manner.

2. SYSTEM BASICS

In this section, first we summarize the recent related works published focusing on the privacy preservation of the HAS. Then, we present the architecture of the proposed privacy-preserving scheme, along with the system parts that are adapted from Idemix [13] to be used in the proposed model. Also, details of the MQTT [7] protocol, which is the lightweight communication protocol that we integrate to our system. Table 2.1 shows the definition of the symbols used in the protocol description. Lastly, a brief attacker model is presented to better explain the domain's borders, as well as the attacker's capabilities in this setup.

In this thesis, we adapt our previous works [1],[5], and we add an efficient privacy preserving protocol. The privacy protocol is developed assuming that HAS Management System (HMS) works as a secure trusted party between the Innovative Home Gateway (IHG) and the Vendor. The members of the communication are described below, under the title System Parts.

2.1 Literature Review

Most of the studies concerning privacy in HAS are surveys or analyses, rather than proposals of privacy preserving embedded systems or privacy preserving solutions to already implemented systems. However, some of the studies in the last 10 years focus on somewhat relevant topics to the focus of this thesis.

In the study by Gharakheili [17], a network level protection for smart home appliances is presented. In this approach, an external entity, called the Security as a Service (SaaS) Provider manages the access control rules. Privacy awareness is provided by blocking the outgoing daily logs, which may contain sensitive information. However, this approach does not take into consideration the communication

Table 2.1 Symbols used in protocol description

<i>Symbol</i>	<i>Description</i>	
$IoT D$		IoT device
V		Vendor
n_i		Nonce [14] with identifier i
U		Cryptographic attribute structure
S, R_i		Part of Issuer's public key
N		Pseudonym of IoT D
A		Ordered set of attributes
$(m_i)_{i \in A}$		Attributes in A
g, h		Public common group parameters
m_1		Master secret
r, v'		Random integers
n		RSA modulus for CL-signatures
P_i		i-th proof generated
σ_{CL}		CL-signature [15]
$IoT D_{A,C}$	A	List of devices in the HAS to get the update
	C	List of connected devices to IHG
$type_{IoT D, PS}$	$IoT D$	Device type of the IoT D
	PS	Device type specified in ProofSpec
$VEPK_i$		i-th public key
$VEPrK_i$		i-th private key
$IDPS_{IoT D}$		Id of the agreed ProofSpec
CH_i		i-th challenge generated
MSG		Encrypted message
gK_{HID}		Group key that belongs to specific Home ID
E_k		Encryption by using k as key
$UUID_{i-P}$		i-th Universally unique identifier [8] generated by P
$C_{IoT D, V}$	$IoT D$	<i>Certificate of IoT D</i>
	V	<i>Certificate of V</i>
$PS_{IoT D, V}$	$IoT D$	<i>proofSpec of IoT D</i>
	V	<i>proofSpec of V</i>
$s_{IoT D, V}$	$IoT D$	<i>Shared secret of IoT D</i>
	V	<i>Shared secret of V</i>
$Proof_{IoT D, V}$	$IoT D$	Generated Proof by IoT D
	V	Generated Proof by V
$fakeProof_{IoT D, V}$	$IoT D$	Generated fakeProof by IoT D
	V	Generated fakeProof by V
$mac_{IoT D}$		Mac address of IoT D
$Hash(.)$		Hash function
$HMAC(.)$		Keyed-Hashing for Message Authentication [16]
$HMAC_{CH_i, MSG}$	CH_i	Generated HMAC using CH_i
	MSG	Generated HMAC using MSG

between the IoT device and the homeowner, and the identity of the IoT devices are not masked.

A recent study by Zavalysyn et al.[18] introduces a privacy-aware smart hub for home environments. Their approach on privacy of the users include restricting apps' usage of the sensitive data collected by smart home devices. They focus on the privacy aspect when the data reaches the app, or the user, rather than what happens on the network level. Their study presents a useful smart hub, however this study does not provide a solution for network level privacy.

Another study by Puri et al. [19] focuses on the privacy of the data collected from smart home appliances such as washer, fan etc. These devices log their activity in terms of energy consumption and this information is sent to the cloud computing layer. They add a fog computing layer to permute this data to anonymize it, assuming that no device activity means the homeowner is not at home. However, this method does not consider the communication between the IoT layer and fog layer, and the method is not implemented or tested to measure the system's security and performance.

Another study by Gope et al. [20] proposes a privacy-preserving two factor authentication scheme for IoT devices to communicate anonymously with a server. To provide two factor authentication, authors integrate physical unclonable functions (PUFs) in addition to shared secret keys as another authentication factor. In order to preserve devices' privacy, they use one time aliases, which is sufficient for their system. However, this approach will fail to mask IoT devices' identities in a setup like ours, where the Vendors communicate with specific brands and types of devices.

When the state of the art approaches, some of which are explained above are considered; there exists a serious lack of privacy preservation in the network layer. Some of the studies only rely on encryption techniques to ensure the privacy of the data, however, in a system like our proposed model, it is not sufficient to only rely on encryption, or one time aliases. To further assure the privacy of HAS, we present privacy preserving identification for HAS.

On top of our contributions to the preliminary versions of the system, our contribution to the literature is that we propose a network layer privacy solution, rather than only providing data privacy in terms of what is revealed to the receiver. By using the Idemix anonymous credential system [13] we also provide data privacy. But rather, in this thesis we focus on the information a third party can deduce by listening to the network, and obfuscate this data in the HMS layer.

2.2 System Parts

HAS consists of five main entities, these are; 1) IoT Devices, 2) IHG, 3) HMS, 4) Vendors, and 5) Homeowners. These entities and their roles are explained in detail below.

2.2.1 IoT Devices

IoT Devices are the smart devices in the home network, such as smart light bulbs and smart TV's. These devices use the network for two purposes, either to report an error, or to receive an update that is sent from the Vendor. These devices connect to the internet only via IHG. IHG is the access point, and the gateway that the IoT devices use to connect the outer network. All of these devices are able to communicate with each other inside the home network, but they can only receive the messages from outside if IHG relays the message to them.

2.2.2 Innovative Home Gateway (IHG)

IHG is a device, which is directly connected to the router via Ethernet connection. We used a Raspberry Pi 4 [21] device to set up the access point for testing purposes, but IHG can be installed to any set-top box. IHG starts working as soon as it is powered on. The main role of the IHG is to listen, it listens to the incoming and outgoing messages and makes sure that the messages are of correct format. If the message format is not correct, it is ignored and not relayed to any of the IoT devices. The IoT devices recognize and immediately connect to this access point when they are turned on, and they cannot connect to any other devices. It is assumed that the IoT devices have the credentials to connect to the IHG access point.

2.2.3 HAS Management System (HMS)

HMS is the server we use for private and secure communication. HMS acts as a trusted third party that all of the Vendors and IHG's recognise and trust. As the trusted middleware, HMS has the following roles:

- HMS is responsible for the secure and private distribution of the master keys that are used in the symmetric encryption scheme we use.
- HMS also acts as the Issuer, which is the responsible party for credential distribution. Devices use these credentials to authenticate themselves to each other and generate proofs during the zero knowledge [22] authentication phase.
- HMS also acts as the middleman between the Vendors and IHG's during the regular communication, where the Vendors send an update to the IoT devices and where the IoT devices send error reports to the Vendors. The reason why we needed HMS as a middleman will be explained in detail in Chapter 5.

2.2.4 Vendors and Homeowners

Vendors and Homeowners are the entities that communicate with the IoT Devices for a particular reason in the system. For instance, Vendors can send periodic updates to the IoT devices and receive error reports from the IoT devices and process these. Homeowners can monitor and control an IoT device remotely such as turning on the air conditioner, lamp, etc. The outline of the system is given in Figure 2.1.

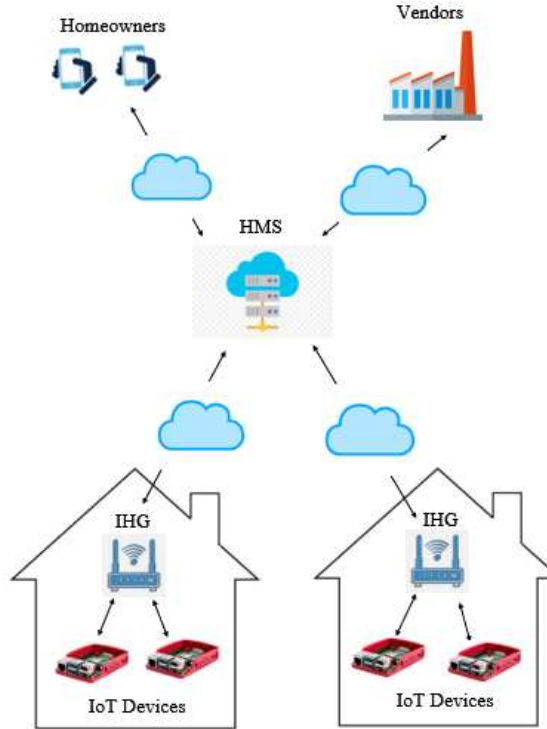


Figure 2.1 Overview of the Home Automation System architecture [1]

2.3 Idemix

In the proposed model, we adopted the Idemix [13] credential system for identification of the entities in the communication. Credential systems are widely used in security applications. Credentials are typically used while requesting access to a system, often during a secured authentication process, to establish a claimed role or specify the attributes about itself. Before the authentication phase of the communication, typically as soon as a new device is connected to the HAS, it requests a credential from the Issuer, which is in our case the HMS server. Same process is done for the Vendors, i.e. when a new Vendor connects to the system it has to first obtain its credentials. This credential issuance process is explained in detail in Chapter 3, which is a part of secure key sharing protocol.

Both of the parties (Vendors and IoT Devices) need credentials, which they use for every communication. This can cause problems, especially in terms of linkability, where one certificate is used many times for different communication. In order to avoid this linkability problem, D. Chaum [23] came up with the anonymous cre-

dential system architecture. In this anonymous credential system, users are known by their pseudonyms. The pseudonyms cannot be linked, but are formed in such a way that a user can prove to another party a statement about his identity. Issuers cannot link the pseudonyms that one user uses for different communications with each other. This way, the user achieves to keep itself anonymous even to the Issuer.

Idemix [13] is the anonymous credential system developed by IBM, that we integrate to our protocols. In this system, there are three entities, the User, the Issuer and the Verifier. The user registers to the Issuer to retrieve the credential for the set of attributes that it needs to prove. The Issuer has the authorization to issue these credentials to the User. The Verifier verifies the credentials of the user, and its specific attributes. After obtaining the credential from the Issuer, User needs this credential to generate the zero-knowledge proof during verification. Using zero knowledge protocol, the user convinces the Verifier that it has the credential issued and signed by the Issuer that contains the corresponding set of attributes. User also proves that it knows the master secret shared along with the credential. This way, unlinkability of different uses of the credential is ensured.

Following are the definitions of the system parts utilized in our model, integrated from Idemix:

Proof is a generated proof of statement, which is typically used in zero knowledge proofs. Basic understanding of zero knowledge proofs is that one party proves that it knows a secret information without revealing it. The proving party answers a challenge, in a way that it would be impossible to answer the challenge correctly without knowing the secret information. Challenge is a random number big enough to not collide with previous random numbers generated, and each party generates a challenge for the other party to use while generating the proofs.

Algorithm 1 demonstrates steps for Proof generation for an *IoT*D, adopted from Idemix. First it checks whether the certificate of the IoT device $C_{IoT D}$ exists, and loads the inputs of $C_{IoT D}$. Then the second check is done to ensure that *IoT*D has all the attributes specified in the ProofSpec of the IoT device, $PS_{IoT D}$. To build $PS_{IoT D}$, *buildProof()* method from the Idemix library [24] is used. The method

employs the CL-signature scheme [15], while validating the certificates of the Prover.

Algorithm 1: Proof Generation Algorithm

Result: $Proof_{IoT D}$ or $fakeProof_{IoT D}$

if $C_{IoT D}$ not exists **then**

 | Get $C_{IoT D}$;

end

Load $C_{IoT D}$ inputs ;

if $PS_{IoT D} \subseteq C_{IoT D}$ **then**

 | Generate $Proof_{IoT D}$;

else

 | Build a dummy document as $fakeProof_{IoT D}$;

end

Credential is the name of the document that the Issuer delivers to the User, which contains the attributes of the user and proves that the user has the mentioned attributes. An example of the attributes specified in the credential is given below in Figure 2.2. As seen in the figure, credential consists of attributes and the signature specific to that user. The attributes include actorType, brand, type, model and homeID values for the device. Moreover, the issuer's public key location is specified in the credential, which is publicly accessible on a server. The signature assigned to the IoT device is received in the credential and stored for later use.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Credential xmlns="http://www.zurich.ibm.com/security/idemix"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.zurich.ibm.com/security/idemix
  ../xsd/Credential.xsd">
  <References>
    <IssuerPublicKey>http://10.36.3.164:8080/files/issuerData/ipk.xml
    </IssuerPublicKey>
    <CredentialStructure>http://10.36.3.164:8080/files/issuerData/
    CredStruct_IoTDevice.xml</CredentialStructure>
  </References>
  <Attributes>
    <Attribute name="peActorType">
      <Value>73</Value>
      <EnumValue>actorType;IoTDevice</EnumValue>
    </Attribute>
    <Attribute name="peBrand">
      <Value>5</Value>
      <EnumValue>brand;Siemens</EnumValue>
    </Attribute>
    <Attribute name="peType">
      <Value>29</Value>
      <EnumValue>type;Television</EnumValue>
    </Attribute>
    <Attribute name="model">
      <Value>-246728037529372992801469033815015107540827695554735485636<
    </Attribute>
    <Attribute name="homeID">
      <Value>5</Value>
    </Attribute>
  </Attributes>
  <Signature>
    <A>1289988802528086975821483544410909496061957659000695219651561573!
    3614329280971510173141429645110230403250533198197302011327864436639!
    <e>2593447230550620599070254914806975719382778895151523062497285831<
    500301</e>
    <v>2987989783187570960523917052376167337795030033600072562451699625!
    8295234999438330806155433314373566578680986295026285851137550760627!
    8036099561992899666198819624986364196427851165027496887431333154536<
    v>
  </Signature>
  <Features/>
</Credential>

```

Figure 2.2 Sample credential issued to an IoT device

ProofSpec (Proof Specification Document) is a file containing the attributes of the Prover that it needs to prove to the Verifier. All of the attributes in the ProofSpec must also appear in the credential of the Prover, but these attributes are just a subset of the attributes of the credential. Prover only reveals the necessary attributes for the specific communication, and the rest of the credential remains secret. Before the verification phase, Prover and Verifier should agree on the specifications of the

ProofSpec. In Figure 2.3, a sample proof specification document is given. ProofSpec consists of declaration and specification sections. Under the declaration section, the attributes of the proof are defined. Under the specification section, the attributes of the corresponding credential are initialized, and it can be seen that only three of the attributes are revealed below (brand, type and actorType) whereas the other two attributes (model and homeID) remain hidden.

```

<?xml version="1.0" encoding="UTF-8"?>
<ProofSpecification xmlns="http://www.zurich.ibm.com/security/idemix"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.zurich.ibm.com/security/idemix
../xsd/ProofSpecification.xsd">
  <Declaration>
    <AttributeId name="id1" proofMode="unrevealed" type="string" />
    <AttributeId name="id2" proofMode="unrevealed" type="int" />
    <AttributeId name="id3" proofMode="unrevealed" type="enum" />
    <AttributeId name="id4" proofMode="unrevealed" type="enum" />
    <AttributeId name="id5" proofMode="unrevealed" type="enum" />
  </Declaration>
  <Specification>
    <Credentials>
      <Credential issuerPublicKey=
        "http://10.36.3.170:8080/files/issuerData/ipk.xml"
        credStruct="http://10.36.3.170:8080/files/issuerData/
        CredStruct_IoTDevice.xml" name="IoTDevCred">
        <Attribute name="model">id1</Attribute>
        <Attribute name="homeID">id2</Attribute>
        <Attribute name="peBrand">id3</Attribute>
        <Attribute name="peType">id4</Attribute>
        <Attribute name="peActorType">id5</Attribute>
      </Credential>
    </Credentials>
    <EnumAttributes>
      <EnumAttribute attributeId="id3" operator="and">
        <EnumValue attributeName="brand">Siemens</EnumValue>
      </EnumAttribute>
      <EnumAttribute attributeId="id4" operator="and">
        <EnumValue attributeName="type">Television</EnumValue>
      </EnumAttribute>
      <EnumAttribute attributeId="id5" operator="and">
        <EnumValue attributeName="actorType">IoTDevice</EnumValue>
      </EnumAttribute>
    </EnumAttributes>
    <Inequalities />
    <Commitments />
    <Representations />
    <Pseudonyms />
    <VerifiableEncryptions />
    <Messages />
  </Specification>
</ProofSpecification>

```

Figure 2.3 Sample ProofSpec of an IoT device

2.4 MQTT

MQTT (MQ Telemetry Transport) [7] is a lightweight messaging protocol that minimizes the network bandwidth and device resource requirements. MQTT has a publish-subscribe mechanism implemented especially for the systems with low processing, such as IoT devices. In this publish-subscribe mechanism, all parties register to the intermediary under a specific topic. When a new message is published to this specific topic, all the parties that have subscribed to this topic receive the message automatically. The intermediary is responsible from this distribution. In our model, we determined a policy in terms of the structure of the topics one can subscribe to. The structure of the topic should be in the following format:

$$\textit{ProcessName}/\textit{UUID1}/\textit{TypeofDevice}/\textit{UUID2}$$

An example topic that we use during the error report scenario is:

$$\textit{report}/19348..57/\textit{Siemens}/\textit{TV}/7484592...34$$

UUID [8] is the universally unique identifier, which is a 128-bit label used for information in computer systems. When generated properly, these UUID's are unique, and this is why we use them for identifying the topics. *UUID1* is placed in the topic by the sender, and *UUID2* is placed in the topic by the receiver. This process is also explained later in Chapter 5. *ProcessName* is either report or update, which helps IHG and HMS understand the means of communication to take proper actions. *TypeofDevice* is a string indicating the brand and type of device i.e. "Samsung/TV". *TypeofDevice* is used to determine which devices in the Home Automation System network will receive the update message, or what kind of a device is sending the error report to the Vendor.

2.5 Attacker Model

Although the intent of the attacker may vary, we explain the intent as obtaining the message, or getting authenticated by either of the end users. The attacker may want to simply learn the messages, alter them, or damage the system. The attacker's capabilities are as follows:

- They can remotely listen to the channel by subscribing to the related topics,
- They can measure the end-to-end latency of the communications between parties and perform a side channel attack,
- They can alter the messages, either by flipping bits or by completely replacing the message,
- They can imitate any of the parties, performing a man in the middle attack,
- They can overwhelm the entities by directing false traffic to them,
- If they have access to the HAS network, i.e. the access point name and password, they can gain access to IoT devices via SSH connection. This requires either physical access to the router or the attacker being in the WiFi connection range.

3. SECURE KEY SHARING AND CREDENTIAL ISSUANCE

In this chapter, we present the secure key sharing and credential issuance protocol used in the proposed privacy preserving identification scheme. This protocol is run as an initialization, which is a preparation for the actual communication. This phase is implemented based on the Idemix [13] anonymous credential system. It starts with credential issuance since both parties should obtain credentials to authenticate themselves to the other party. Initialization protocol is carried out in an encrypted manner, where the public/private key pairs are generated using the Verifiable Encryption Scheme of Idemix [6].

3.1 Zero Knowledge Proofs

Zero knowledge [22] is a concept in cryptography where one party proves that it possesses a certain secret information (key, credential etc.) without revealing the information itself. They provide small pieces of unlinkable information that can accumulate to show that the validity of an assertion is overwhelmingly probable. Even though there is a small probability that the Prover does not know the secret information, but is able to generate, or guess, the response to a challenge correctly, this probability is neglected. An example of zero knowledge proof is interactive two-party protocols [25]. In these protocols, the Verifier asks the Prover one or more questions. If the Prover knows the secret information, it can answer all the questions correctly. This question is in the form of a challenge number, and the Prover is expected to generate a proof using this challenge number and the secret information. This proof is sent to the Verifier over a secure channel, and the Verifier, who either knows the secret information or knows the answer to the challenge, accepts the proof if the proof is validated.

All of the zero-knowledge proofs in the Idemix library use the common three move Zero Knowledge protocol, and is non-interactive [24]. The protocol is non-interactive in the way that parties agree on some common string or group parameters to be used during the process.

3.2 Verifiable Encryption Scheme

A verifiable encryption scheme is a protocol that allows the Prover to convince the Verifier that a ciphertext c is an encryption of a plaintext p under the public key PK and label L , where PK and L are known by both the Prover and the Verifier [26]. This verifiable encryption scheme provides a computational security and applies to the special soundness and zero knowledge properties of Zero Knowledge Protocols [27]. Originally in the Verifiable Encryption scheme [6], it is assumed that the public/private key pairs are generated by a trusted party using the appropriate key generation algorithms, and in our implementation of this scheme, we assume that the public/private key pairs are generated by the manufacturer of the IoT devices, and pre-embedded to the devices' memories. Same conditions apply for the key pair or the HMS server.

In the proposed protocol, the Verifiable Encryption Scheme [6] is used in the secure key sharing and credential issuance protocol. We assume that the Issuer (HMS) knows the public key of the IoT device that the credentials are addressed to, and likewise, the IoT device knows the public key of the Issuer. In a regular RSA public key encryption [28], a message is encrypted with the receiver's public key and the receiver can decrypt the message with its private key. But in this case, there's no insurance that the message is encrypted by the correct sender, namely anyone can encrypt a message. For our protocol, both parties need to make sure that the other party is legitimate, and for this reason they need to verify the identity of each other. In Verifiable Encryption Protocol [6], the Encryptor is at the same time the Prover, and likewise in the Verifiable Decryption Protocol, the Decryptor is at the same time the Verifier. The details of this verifiable encryption scheme will not be explained further as it is not vital in terms of the understanding of the proposed protocol.

3.3 Key Sharing and Credential Issuance

As an initialization, the IoT Devices and Vendors must obtain their master secret and credentials that they will use for the rest of the regular communication during error report and update scenarios. This key sharing and credential issuance protocol is implemented based on the Idemix [13] anonymous credential system. The outline of the key sharing and credential issuance process is given below in Figure 3.1.

Protocol starts with offline generation of the public/private key pair and sharing the public keys with the other party. Initially, Issuer generates $VEPK_1$ and $VEPrK_1$ public and private keys, and the user generates $VEPK_2$, $VEPrK_2$ public-private key pair. Then both parties share their public keys with each other i.e. when HMS is going to issue credentials to an IoT device, HMS shares its public key with the IoT device, and IoT Device shares its public key with HMS. Issuer and IoT device use these keys to encrypt their messages during the protocol.

In detail, Issuer generates a nonce n_1 , encrypts the message using $VEPK_2$ and sends it to User. User gets the message, decrypts it using $VEPrK_2$ and gets n_1 . *IoTD* computes a structure U based on attributes A . Proof generated by *IoTD* is to prove the knowledge of master secret m_1 associated with the pseudonym, N , of the User. Additionally, random t -values as g^t are computed and a challenge CH_1 is created by calculating the $Hash(n_1|N|t)$ of n_1 , N and t . A response in the form of $s := r + CH_1 m_1$ is calculated based on this challenge. Proof P_1 and nonce n_2 are generated and sent to the Issuer encrypted with $VEPK_1$. Also structure U is sent in an encrypted way. When Issuer gets the messages, it decrypts them using $VEPrK_1$. The last round is generating a signature for the credential of User. First, Issuer verifies P_1 and if verification does not fail, it generates a CL-signature [15] σ_{CL} attached to the attribute list $(m_i)_{i \in A}$. Issuer also generates P_2 to send with σ_{CL} . All P_2 , σ_{CL} and $(m_i)_{i \in A}$ are encrypted with $VEPK_2$. Together with the message, a 32-byte integer is randomly generated as the shared secret s_{IoTD} . At the end of the protocol, issued credential σ_{CL} and randomly generated s_{IoTD} are received by the User. This way, shared secret s_{IoTD} is distributed securely to be used as a symmetric key in mutual verification and authentication protocol.



HMS(Issuer)



IoT Device

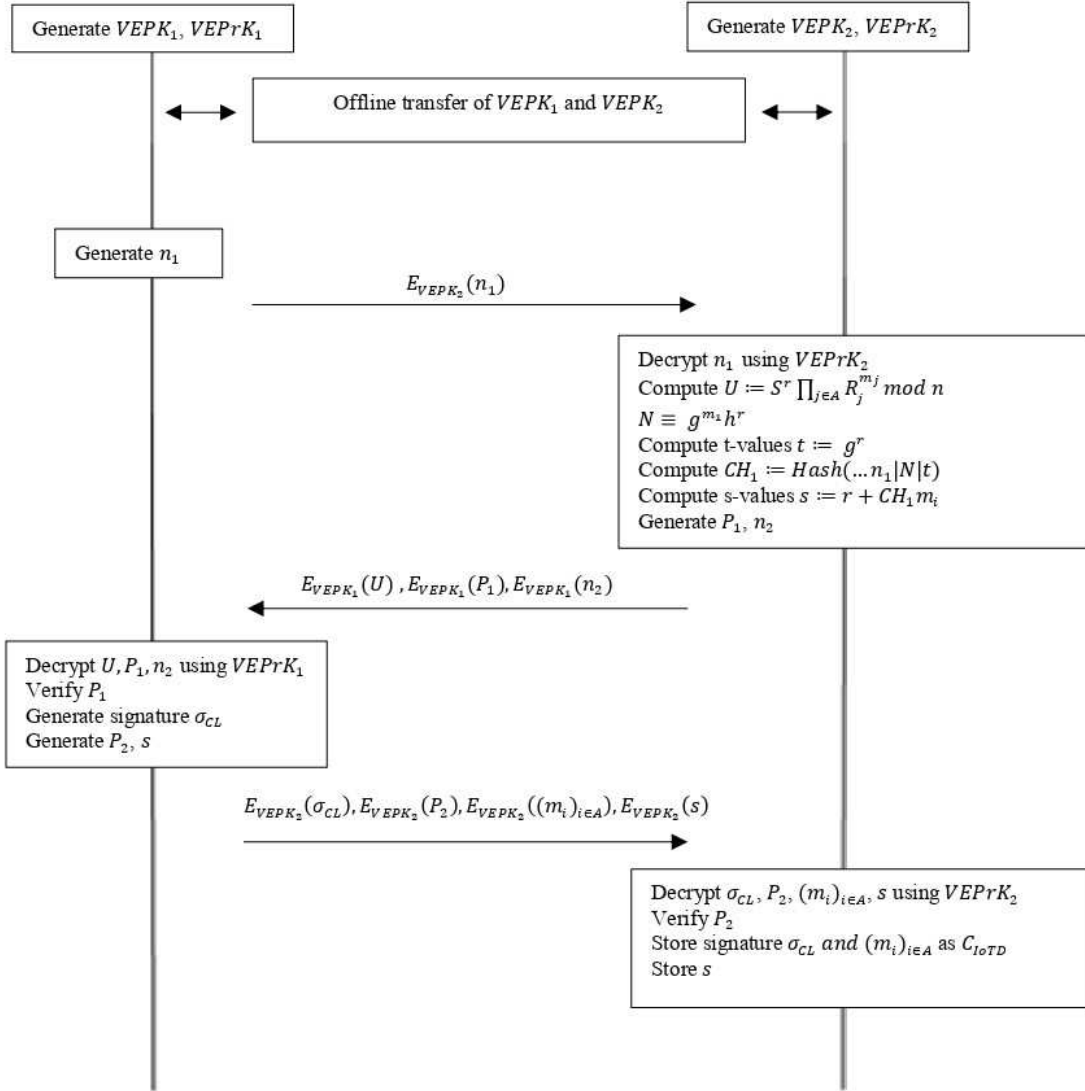


Figure 3.1 Key Sharing and Credential Issuance protocol

4. PRIVACY PRESERVING MUTUAL AUTHENTICATION PROTOCOL

In this chapter, we present the privacy preserving mutual verification and authentication protocol. We explain the update and error report scenarios in detail as well as the privacy preserving properties of the protocol. The preliminary version of the privacy preserving mechanism was already implemented in our previous works [1],[5], which is the fakeProof generation. The rough outline of this mechanism can be seen in Algorithm 1 in Chapter 2. In this thesis, we improve the privacy of the preliminary HAS architecture by introducing the privacy-preserving identification and authentication protocol, where HMS manages data flow between Vendor and IHG by communicating with them over MQTT [7] server. The privacy preserving mechanism that we present in this thesis uses HMS as the trusted server, and assures the untraceability of the communications between the Vendors and IHG's. HMS is responsible for switching MQTT [7] topics between Vendor and IHG in order to obfuscate communication.

4.1 Protocol Steps

The privacy preserving mutual verification and authentication protocol is implemented for the upstream and for the downstream traffic scenarios. The upstream traffic is where an outside user, which is in our case the Vendor, sends software updates to the IoT devices subscribed to a specific topic. The downstream traffic is where an IoT device sends error reports to its Vendor. The steps of the protocol are explained below, for the upstream traffic, for the case where Vendor sends updates to IoT devices.

In Algorithm 2, the proposed privacy preserving mechanism's details are given. This algorithm explains the role of HMS during the update protocol, which provides pri-

vacy while switching keys and topics between the incoming and outgoing messages. HMS keeps the topic and key pairs in hash maps so that the pairs are found quickly, to minimize the delay on HMS as much as possible.

Algorithm 2: Privacy Preserving Mechanism

```

if MSG is the first message from Vendor then
    Decrypt MSG and typeIoT on the topic;
    Encrypt MSG and typeIoT with gKHID;
    Save previous UUID1 in the topic ;
    Replace topic with random UUID2 and encrypted typeIoT ;
end
if MSG is from IHG then
    if MSG is the first message then
        Extend topic by generating random UUID2;
        Put topic pairs to hashMap ;
    end
    Decrypt MSG with sIoT ;
    Get topic pair from hashMap ;
    Encrypt MSG with sV;
    Send encrypted MSG to Vendor;
end
if MSG is from Vendor then
    Decrypt MSG with sV;
    Get topic pair from hashMap ;
    Encrypt MSG with sIoT;
    Send encrypted MSG to IHG;
end

```

4.1.1 Update Protocol

The update protocol is used when a Vendor sends software updates to the devices of a specific type. This protocol has two outcomes, where the IoT device is of the addressed type; hence, it authenticates itself to the Vendor and receives the update, or the IoT device is not of the addressed type and provides a fakeProof instead of Proof and does not receive the update at the end of the process. The successful update scenario is explained below under three steps.

Step 1 At the beginning of the protocol, Vendor, V , chooses the appropriate proofSpec, PS_{IoT_D} , including all the attributes that IoT device, IoT_D , should provide. First, appropriate type of IoT_D specified in proofSpec, $type_{PS}$, is encrypted with shared secret key of Vendor, s_V , to be added to the topic. Then, id of the proofSpec, $IDPS_{IoT_D}$, is also encrypted with s_V and published over the specific MQTT topic $HMS/update/UUID_{1-V}/E_{s_V}(type_{PS})$ where $UUID_{1-V}$ is randomly generated universally unique identifier [8]. HMS receives encrypted $IDPS_{IoT_D}$, $E_{s_V}(IDPS_{IoT_D})$, and decrypts it by using the stored s_V together with $E_{s_V}(type_{PS})$ in the topic to get the $type_{PS}$. Then, HMS encrypts $type_{PS}$ and the $IDPS_{IoT_D}$ with the group key belonging to specific Home ID, gK_{HID} , in order to send to the IHG. Lastly, HMS generates random $UUID_{1-HMS}$ to swap with $UUID_{1-V}$ and sends $E_{gK_{HID}}(IDPS_{IoT_D})$ over the topic $update/UUID_{1-HMS}/E_{gK_{HID}}(type_{PS})$. After getting the message $E_{gK_{HID}}(IDPS_{IoT_D})$, for every IoT device in IoT_D_C which is a list of connected devices, IHG checks if they are appropriate to receive the update. If $type_{PS}$ matches with the device type of IoT_D , $type_{IoT_D}$, then the device is appended to the appropriate devices list IoT_D_A and IHG communicates only with IoT_D in IoT_D_A .

Having received and decrypted the $IDPS_{IoT_D}$ message, IoT_D informs IHG that the verification is required by sending an encrypted *VERIFY* command attached with its mac address (mac_{IoT_D}), which is a physical identifier of the device. For every IoT_D in IoT_D_A , IHG generates a nonce as a unique identifier $UUID_{2-IHG}$ to create a special topic for the corresponding IoT_D . Rest of the messages are published on that selected special topic. Thus, IHG publishes $E_{s_{IoT_D}}(VERIFY)$ on topic $update/UUID_{1-HMS}/E_{gK_{HID}}(type_{PS})/UUID_{2-IHG}/mac_{IoT_D}$. As indicated, IHG is the gateway in the system; from this point on its responsibility is to relay the messages to the other party.

Step 2 After receiving $E_{s_{IoT_D}}(VERIFY)$ command from IHG, HMS decrypts the message with the corresponding s_{IoT_D} and encrypts it with s_V . The topic is also modified at HMS's side so that an intruder cannot track the topics and find out which IoT_D is the target. HMS removes mac_{IoT_D} from the topic and gets the shared secret s_{IoT_D} for the specific mac address from its database. Then, HMS generates random $UUID_{2-HMS}$ and replaces it with the topic. Also, topic pairs are stored in a hash map. The pair consists of one topic that is listening messages from IHG and the other one that is listening messages from Vendor. Then, HMS sends $E_{s_V}(VERIFY)$ on topic $update/UUID_{1-V}/E_{s_V}(type_{PS})/UUID_{2-HMS}$. After this point, HMS's responsibility is getting the message, decrypting it and encrypting with the other parties' shared secret key. Lastly, it sends the message over the specific topic which is stored in the hash map.

When V receives $VERIFY$ message, it generates a challenge, CH_1 . After encrypting CH_1 with $s_{IoT D}$, V sends it to HMS as $E_{s_V}(CH_1)$ and CH_1 is relayed to IHG over the topic $update/UUID_1-HMS/E_{g_{K_{HID}}}(type_{PS})/UUID_2-IHG$. This way, $E_{s_{IoT D}}(CH_1)$ is received by $IoT D$.

In order to create $Proof_{IoT D}$ to prove its identity, $IoT D$ uses received CH_1 , $PS_{IoT D}$ and the credential, $CI_{IoT D}$. If $PS_{IoT D}$ is not a subset of $C_{IoT D}$, then $fakeProof_{IoT D}$ is created. Also, $IoT D$ creates CH_2 to send it to V . In order to preserve integrity, $HMAC$ [12] of CH_2 , $HMAC_{CH_2, Proof_{IoT D}}$, is calculated using $Proof_{IoT D}$ as key, and sent together with CH_2 and $Proof_{IoT D}$. All the messages are concatenated and encrypted as $E_{s_{IoT D}}(Proof_{IoT D}||CH_2||HMAC_{CH_2, Proof_{IoT D}})$. HMS receives the message and relays to V as $E_{s_V}(Proof_{IoT D}||CH_2||HMAC_{CH_2, Proof_{IoT D}})$.

Step 3 V verifies the received $HMAC_{CH_2, Proof_{IoT D}}$ to check the validity of the message and $Proof_{IoT D}$ to authenticate $IoT D$. After successful verification of $HMAC_{CH_2, Proof_{IoT D}}$, V tries to verify $Proof_{IoT D}$ to check whether $Proof_{IoT D}$ is a $fakeProof_{IoT D}$ or not. If $Proof_{IoT D}$ can be verified, then V generates $Proof_V$. In other case, it generates $fakeProof_V$ in order to continue with the protocol. Also, for integrity check V generates $HMAC_{MSG, Proof_V}$ and sends it together with MSG and created $Proof_V$. HMS gets the message $E_{s_V}(Proof_V||MSG||HMAC_{MSG, Proof_V})$, decrypts it with s_V and sends $E_{s_{IoT D}}(Proof_V||MSG||HMAC_{MSG, Proof_V})$ on the specific topic to IHG.

When $IoT D$ receives the message, it checks $HMAC_{MSG, Proof_V}$. After it is verified, it also verifies received $Proof_V$. This way, mutual verification and authentication is ensured.

The update scenario outline can be seen in Figure 4.1. Encrypted messages are shown above the arrows and MQTT topics, on which the messages are published, are shown under the arrows.

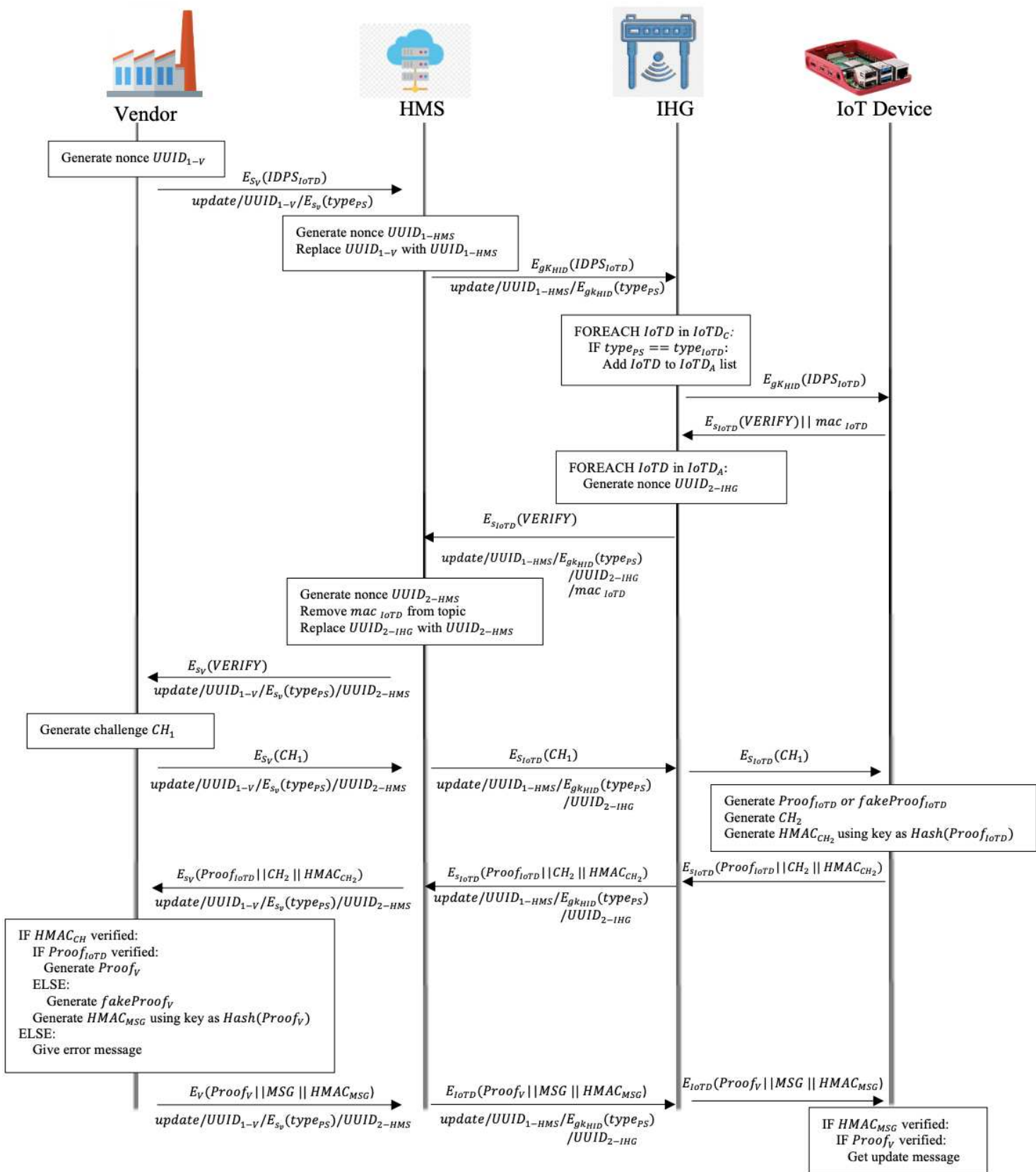


Figure 4.1 Outline of the Mutual Authentication Process (Downstream)

4.1.2 Error Report Protocol

The error report protocol is used then an IoT device sends error reports to its Vendor. Since the addressed Vendor is known, this scenario always results in Vendor successfully receiving the report message. Hence, fakeProofs are not used during error report scenario. The error report scenario outline can be seen in Figure 4.2. Encrypted messages are shown above the arrows and MQTT topics, on which the messages are published, are shown under the arrows. The protocol steps are explained below in detail.

Before the protocol begins, Vendor subscribes to the topic $report/+ /type_{IoT D}/+$ on which the IoT devices of the type $type_{IoT D}$ publish their error reports on. $+$ in the topic means all values and is replaced with a UUID later.

Step 1 At the beginning of the protocol, IoT Device, $IoT D$ chooses the appropriate proofSpec, PS_V , including all the attributes that Vendor, V , should provide. Then, id of the proofSpec, $IDPS_V$, is encrypted with $s_{IoT D}$, and sent to IHG over a TCP connection, along with a random UUID that is generated by the sender $IoT D$. IHG, after receiving $IDPS_V$, subscribes to the topic $HMS/report/UUID_{1-IoTD}/E_{s_{IoT D}}(type_{IoT D})/+$ where $UUID_{1-IoTD}$ is randomly generated universally unique identifier. IHG publishes $E_{s_{IoT D}}(IDPS_{IoT D})$ over the same topic. HMS receives $E_{s_{IoT D}}(IDPS_{IoT D})$ and decrypts it with $s_{IoT D}$ which is stored along with the mac address of the IoT device and re-encrypts it using s_V . HMS also decrypts $E_{s_{IoT D}}(type_{IoT D})$ and re-encrypts it using s_V . Lastly, HMS generates a new UUID, $UUID_{1-HMS}$ and replaces $UUID_{1-IoTD}$ with this new UUID. HMS publishes $E_{s_V}(IDPS_{IoT D})$ over the topic $HMS/report/UUID_{1-HMS}/E_{s_V}(type_{PS})/+$.

Step 2 Vendor receives this $E_{s_V}(IDPS_{IoT D})$ message and decrypts it with its key, s_V and verifies that it has the corresponding ProofSpec, then sends back $E_{s_V}(VERIFY)$ using the topic $HMS/report/UUID_{1-HMS}/E_{s_V}(type_{PS})/VENDOR$. Vendor also subscribes to topic $HMS/report/UUID_{1-HMS}/E_{s_V}(type_{PS})/IHG$ to receive the rest of the messages. HMS receives $E_{s_V}(VERIFY)$ coming from the Vendor, and decrypts it using s_V . Then, HMS encrypts the verify command with $s_{IoT D}$ and sends $E_{s_{IoT D}}(VERIFY)$ over the new topic. The new topic is $HMS/report/UUID_{1-IoTD}/E_{s_{IoT D}}(type_{PS})/UUID_{2-HMS}$ where the first UUID is replaced with IoT D's UUID and the second UUID is replaced with a new randomly generated UUID. IHG receives this message, and relays it to the devices with the matching brand and type, over a TCP connection socket. $IoT D$ receives

this message, decrypts it using $s_{IoT D}$, and if the incoming message is “VERIFY”, it starts the challenge response protocol. $IoT D$ generates a big integer, which is called a challenge. This number is denoted with CH_1 . $IoT D$ encrypts CH_1 and sends $E_{s_{IoT D}}(CH_1)$ to IHG, and IHG publishes this message on the topic $HMS/report/UUID_1-IoTD/E_{s_{IoT D}}(type_{PS})/UUID_2-HMS$. Now HMS receives this message, decrypts it with $s_{IoT D}$ and encrypts it with s_V . HMS publishes $E_{s_V}(CH_1)$ over $HMS/report/UUID_1-HMS/E_{s_V}(type_{PS})/UUID_2-V/IHG$ topic. Vendor receives $E_{s_V}(CH_1)$ and decrypts it using s_V , and generates $Proof_V$. Vendor also chooses a random challenge CH_2 and generates $HMAC_{CH_2, Proof_V}$, which is the HMAC of the CH_2 using $Proof_V$ as the key. Vendor sends $E_{s_V}(Proof_V||CH_2||HMAC_{CH_2, Proof_V})$ over $HMS/report/UUID_1-HMS/E_{s_V}(type_{PS})/UUID_2-V/VENDOR$ topic. HMS receives this message, decrypts and re-encrypts it and publishes $E_{s_{IoT D}}(Proof_V||CH_2||HMAC_{CH_2, Proof_V})$ over the topic $HMS/report/UUID_1-IoTD/E_{s_{IoT D}}(type_{PS})/UUID_2-HMS/IHG$. IHG receives and relays the message to corresponding $IoT D$.

Step 3 $IoT D$ decrypts this message, gets $Proof_V||CH_2||HMAC_{CH_2, Proof_V}$, and verifies the proof. Then it verifies the correctness of HMAC, by generating it again. Lastly, $IoT D$ generates $Proof_{IoT D}$ for CH_2 and it also generates $HMAC_{MSG, Proof_{IoT D}}$ where MSG is the error report message. $IoT D$ sends $E_{s_{IoT D}}(Proof_{IoT D}||MSG||HMAC_{MSG, Proof_{IoT D}})$ to IHG, who then publishes this message on the topic $HMS/report/UUID_1-IoTD/E_{s_{IoT D}}(type_{PS})/UUID_2-HMS/IHG$. HMS receives this message, decrypts and publishes $E_{s_V}(Proof_{IoT D}||MSG||HMAC_{MSG, Proof_{IoT D}})$ on the topic $HMS/report/UUID_1-HMS/E_{s_V}(type_{PS})/UUID_2-V/VENDOR$. Lastly, Vendor receives this message, validates the $Proof_{IoT D}$ and checks the $HMAC_{MSG, Proof_{IoT D}}$. If the proof is correct, then Vendor receives the error report message MSG .

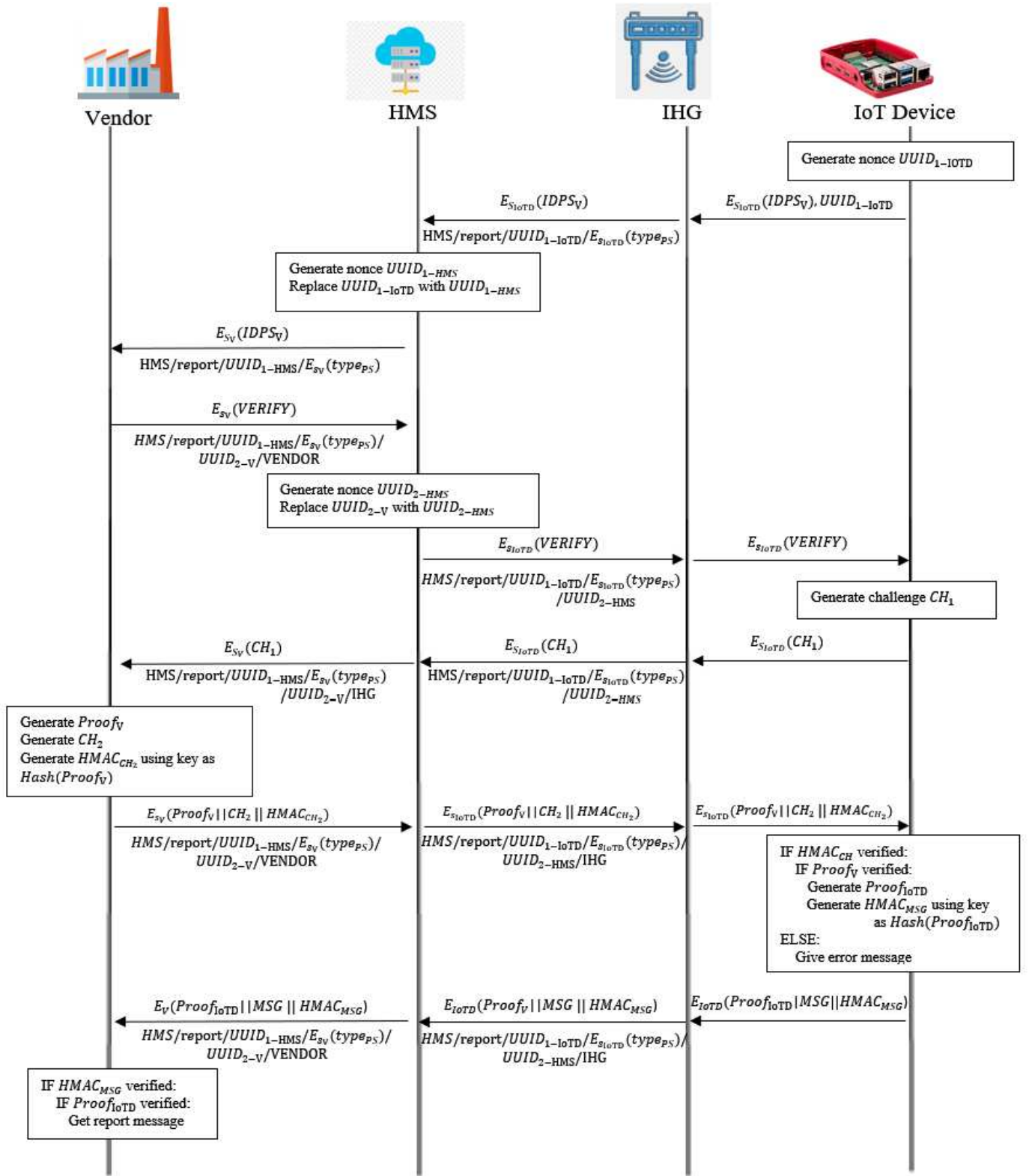


Figure 4.2 Outline of the Mutual Authentication Process (Upstream)

5. PERFORMANCE EVALUATION

In this chapter, we evaluate the performance of the proposed protocol for the upstream and downstream traffics. First, we explain the setup and implementation details of our testbed, and give detailed features of all the devices and software that are used in the HAS simulation. Then, we give detailed results of the performance tests for the upstream and downstream traffic simulations. Lastly, we evaluate and discuss the results of the performance tests.

5.1 Setup and Implementation

For the implementation and testing of our proposed system, we used Raspberry Pi 3 and Raspberry Pi 4 boards [21] as IoT devices, and we chose to use these devices for several reasons. Firstly, most of the research and development process of IoT platforms are supported with single board computers, such as Raspberry Pi. Secondly, they are small in size, and affordable. And lastly, they are very easy to customize and powerful enough to run the software without significant delay. There are a total of 11 IoT devices in our testbed, 5 of which are Raspberry 3 devices, and the remaining 6 are Raspberry 4 devices. The features of all the devices are given in Table 5.1. Furthermore, we used one Raspberry Pi device as a set-top box for IHG. We mentioned in system parts that any device that can be customized as an access point can be used as the IHG. IHG is both the gateway, and the access point that IoT devices use to connect to the Internet. We set up the Raspberry Pi 4 device as an access point and uploaded the IHG software. Raspberry Pi connects to the home/school IoT network via an Ethernet cable and uses this IoT network to connect the IoT devices to the Internet. The features of the IHG are given in Table 5.2.

Table 5.1 IoT device features of the testbed

IoT Device Model	Feature Name	Device Feature
Raspberry Pi 3	Operating System	Raspbian
	RAM	1GB LPDDR2
	GPU	BroadCom VideoCore IV
	CPU	4x ARM Cortex-A53, 1.2/1.4 GHz
Raspberry Pi 4	Operating System	Raspbian
	RAM	2GB LPDDR4-3200
	GPU	Broadcom BCM2711
	CPU	4x ARM Cortex-A72, 1.5 GHz

Table 5.2 IHG features of the testbed

Feature Name	Device Feature
Device Name	Raspberry Pi 4
Operating System	Raspbian
RAM	4GB LPDDR4-3200
GPU	Broadcom 2711
CPU	4x ARM Cortex-A72, 1.5 GHz

Proof Specification documents are served from a publicly reachable server to make it feasible for the parties to reach. Server's IP address is known to every entity in the system. In addition, HMS and the MQTT server run on a laptop computer with Windows 10 operating system and Vendor runs on a laptop computer with MacOS Mojave operating system. The detailed features of the Windows device is given in Table 5.3 and the detailed features of the MacOS device is given in Table 5.4.

All of the software applications for IHG, HMS, IoT device and Vendor are developed using Java Programming Language version 11. MQTT and Idemix libraries are included in the software. All of the IoT devices are connected to the IHG via WLAN to simulate the smart home environment.

Table 5.3 HMS/MQTT server features of the testbed

Feature Name	Device Feature
Device Name	Lenovo Thinkpad
Operating System	Windows 10
Processor	Intel Core i5
RAM	8 GB
CPU	1.6 GHz

Table 5.4 Vendor features of the testbed

Feature Name	Device Feature
Device Name	Macbook Pro
Operating System	macOS Majove 10.14.6
Processor	Intel Core i7
CPU	3.3 GHz
RAM	16 GB LPDDR3 2133 MHz

5.2 Performance Evaluation Results

For the experiments, the testbed is set up and all of the test cases have been run via repeating the communication fifty times to calculate the average end-to-end latency. Tests are performed for the two-way communication model, where IoT sends error reports to the Vendor and where the Vendor sends updates to the IoT devices. For all of the test cases, we set the payload data size to 10 KB for uniformity and comparability of the test results. However, we also tested the protocols with different payload sizes with a fixed number of IoT devices to measure the effect of the payload size to end to end latency. The end-to-end latencies are calculated starting from the beginning of the request of update/report until the end of mutual verification and authentication protocol. Since the key sharing and credential issuance protocol executes only once for each device, the latency does not have effect on the performance of the overall system, so we do not include this protocol to the test cases.

5.2.1 Successful Update Scenario

In this section, we present the performance evaluation results for downstream traffic which is the delivery of successful updates to IoT devices. The breakdown of average computational delay (in milliseconds) is given in Table 5.5.

As seen in the table, average computational delay is approximately 1543 milliseconds. Overall end to end latency for one device receiving a successful update is calculated as 1637 milliseconds, meaning that 94 milliseconds are spent for data communication over the network. We also conducted experiments at which multiple IoT devices have undergone simultaneous update operations. Figure 5.1 shows the end-to-end latency results of 1-to-11 IoT devices which successfully receive the update at the same time. In the timings of this figure, both communication and batch computational delays are included. As seen in Figure 5.1, average time to get updates is in between 1.6 - 3.9 seconds. The increase in latency with respect to the increase in number of IoT devices is with approx. 0.22 second per device slope, as seen by the linear trend line, which indicates the scalability of the proposed protocol.

Table 5.5 Average computational results for successful update

Calculation	Time (ms)	Device
creating challenge	0.58	Vendor
creating proof	761.72	IoT Device
calculating HMAC	3.22	IoT Device
verifying HMAC	2.64	Vendor
verifying proof	2.68	Vendor
creating proof	16.16	Vendor
calculating HMAC	0.34	Vendor
verifying HMAC	2.78	IoT Device
verifying proof	419.82	IoT Device
total encryption/decryption	7.64	Vendor
total encryption/decryption	306.76	IoT Device
Total time (ms)	1543.34	

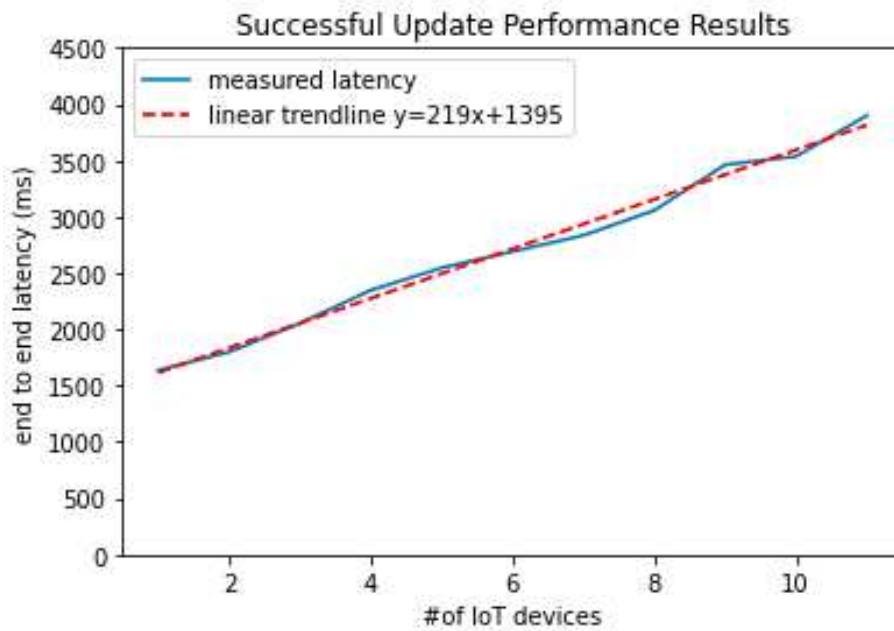


Figure 5.1 Successful update performance results

In the protocol implementation, a 10KB file (payload) is sent as an update and downloaded on IoT devices' side. We also tested for different sizes of update files to discuss the feasibility of the protocol. Table 5.6 gives the results of end to end latency calculated when five IoT devices are receiving successful updates with different file sizes. As can be seen here, when file size increases from 1 KB to 100 KB, end to end latency increases from approximately 2.5 seconds to 4 seconds. The difference between overall latency values only depends on the computational delay of file encryption and decryption processes, and expectedly the increase is linear.

Table 5.6 Average latencies for successful update w.r.t different file sizes

File Size (KB)	End to end latency (ms)
1	2486
10	2546
50	3233
100	3961

5.2.2 Failed Update Scenario

In this part, we present the experimental results for delivery of failed updates which correspond to downstream traffic. Failed update case provides privacy-awareness in the protocol. Even if the update is not addressed to the IoT device, the protocol is still completed to hide the real identity of the device from outsiders. This way, third parties cannot differentiate the actual target IoT devices of the update. Average computational delay breakdown of failed update scenarios for one device is given in Table 5.7. Table 5.7 shows that average computational delay is 694.42 milliseconds and on top of that communication delay is measured as 104.50 milliseconds that yields overall end-to-end delay of 799 milliseconds for one device receiving a failed update.

Table 5.7 Average computational results for failed update

Calculation	Time(ms)	Device
creating challenge	0.58	Vendor
creating proof	25.32	IoT Device
calculating HMAC	2.86	IoT Device
verifying HMAC	2.84	Vendor
verifying proof	10.86	Vendor
creating proof	5.32	Vendor
calculating HMAC	0.68	Vendor
verifying HMAC	2.9	IoT Device
verifying proof	6.92	IoT Device
total encryption/decryption	12.32	Vendor
total encryption/decryption	29.83	IoT Device
Total time (ms)	694.42	

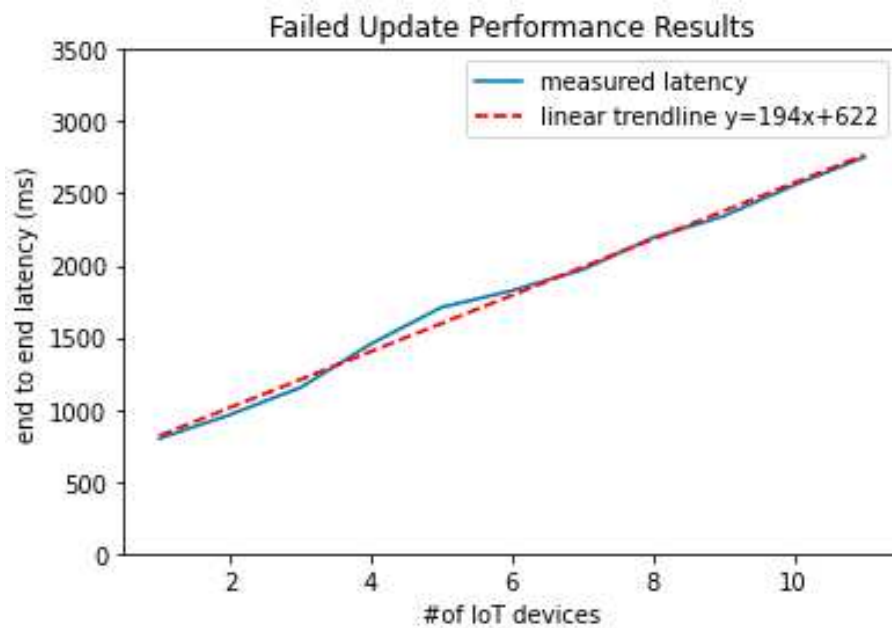


Figure 5.2 Failed update performance results

Figure 5.2 shows the end to end latency of failed updates with multiple IoT devices. As seen in Figure 5.2, it took to receive failed updates between 0.8 - 2.8 seconds with linear increase. The slope of the increase is approximately 0.19 second per device, as suggested by the linear trend line. Thus, we can conclude that also for failed update scenarios, the protocol shows good scalability.

As in the successful update scenario, we tested the protocol with different file sizes. Table 5.8 shows end to end latency results based on different file sizes. It is shown in the table that when file size increases from 1KB to 100KB, end to end latency increases from approximately 1.5 seconds to 3.3 seconds. As in the successful update scenario, the difference between end to end latency values depends on the file encryption and decryption processes and results in a linear increase.

Table 5.8 Avg. latencies for successful update w.r.t. different file sizes

File Size (KB)	End to end latency (ms)
1	1493
10	1709
50	2457
100	3281

Moreover, experiments are conducted for different scenarios with mixed and simultaneous failed and successful update cases. In Table 5.9, the average results for end-to-end latency in such mixed scenarios of eleven IoT Devices are shown. As Table 5.7 demonstrates, failed updates in the case that also successful updates are sent by the Vendor do not make a big difference in terms of end to end latency. It is seen that getting one successful update and ten failed updates has higher overall latency than the case where all the eleven IoT devices receive failed updates. This is because, even one successful delivery of an update results in more computational complexity. On the other hand, receiving eleven successful updates has higher overall latency than the case where ten successful and one failed updates are received because of the increased computational complexity.

It can be concluded that since the mutual verification and authentication protocol is completed in all cases, and there is at least one device receiving a successful update, latencies are close to each other. Still, while the number of devices that receive successful updates are increased, end to end latency is also increased due to higher computational delay.

Table 5.9 Average results for mixed successful and failed update

Successful/Failed Update	Time(ms)
1 Successful/ 10 Failed	2916
2 Successful/ 9 Failed	3103
3 Successful/ 8 Failed	3273
4 Successful/ 7 Failed	3369
5 Successful/ 6 Failed	3442
6 Successful/ 5 Failed	3594
7 Successful/ 4 Failed	3657
8 Successful/ 3 Failed	3718
9 Successful/ 2 Failed	3734
10 Successful/ 1 Failed	3795

5.2.3 Error Report Scenario

In this section, we present the experimental results for upstream traffic which is the delivery of error reports from IoT devices to Vendor. In Table 5.10, the breakdown of average computational delay for one device is given.

Table 5.10 shows that average computational cost for one error report delivery is 1359.48 milliseconds. The time calculated for communication delay is 197 milliseconds which yields an average end to end latency of 1557 milliseconds.

Additionally, Figure 5.3 indicates the results of end to end latency for 1-to-11 errors reports sent simultaneously to the Vendor. As shown in the figure, latency values increase linearly with a slope of approximately 0.08 second per device.

Table 5.10 Average calculation results for error report

Calculation	Time(ms)	Device
creating challenge	1.36	IoT Device
creating proof	0.04	Vendor
calculating HMAC	0.24	Vendor
verifying HMAC	20.92	IoT Device
verifying proof	431.62	IoT Device
creating proof	727.34	IoT Device
calculating HMAC	4.56	IoT Device
verifying HMAC	0.36	Vendor
verifying proof	21.44	Vendor
total encryption/decryption	12.98	Vendor
total encryption/decryption	138.62	IoT Device
Total time (ms)	1359.48	

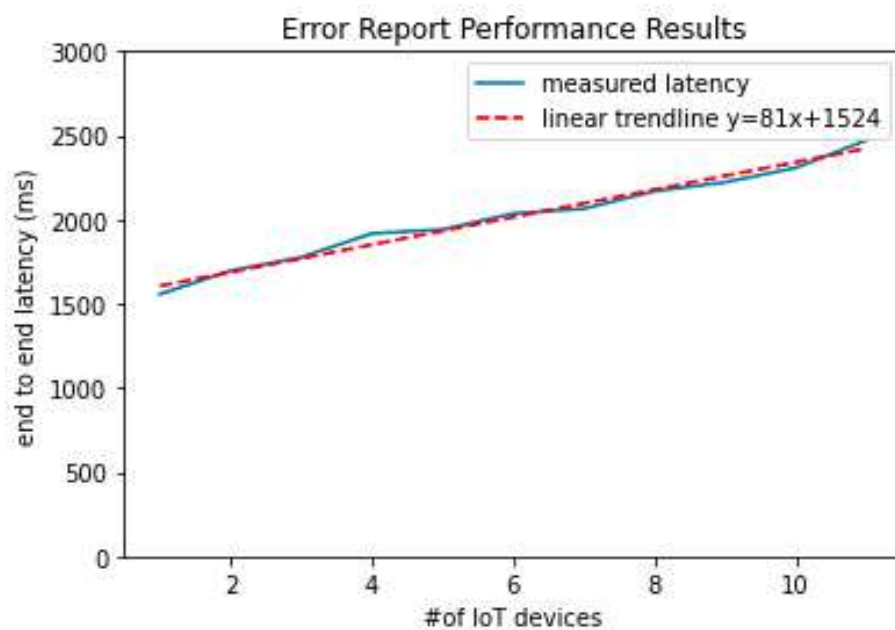


Figure 5.3 Average end to end latency of receiving error reports

5.3 Discussions

In this section, we discuss the experimental results in a comparative way. Average end to end latency in successful update scenario is higher than the error report scenario since in the successful update scenario, decryption of the final message including the Proof and 10KB file is performed at IoT device's side with low processing power and that results in more computational complexity. Thus, the slope of increase per device in the successful update scenarios is higher than the error report scenario. The results indicate that when fake proof is generated, the update scenario is much faster. The reason is, generating fake proof is almost twice as fast as generating accurate proof. Furthermore, it reduces the calculation time for proof verification. The slope of increase per device in both failed update and successful update applications are the same confirming that similar applications deliver consistent results.

Also, in Table 5.6 and Table 5.8, it can be seen that the payload data size affects the end to end latency in both update and error report scenarios, however, this increase is higher for the update scenario. This shows us that the low processing power of the IoT devices combined with big payload data size increases the end to end latency more significantly. On the other hand, big data sizes (such as 50KB and 100KB) will be most likely used only in update scenarios where the Vendor sends software update to IoT devices. We expect error reports to not exceed 10KB, therefore, the increase will not be crucial in this case. In Figure 5.1, Figure 5.2 and Figure 5.3, the increase in the graphs are linear as expected due to increased computational complexity at Vendor's side. Still, the slope of the increase is quite low showing that the proposed model is scalable to provide privacy-preserving identification and authentication for Home Automation Systems.

The breakdowns of the average computational delays for the update scenarios (Table 5.5 and Table 5.7) and for the error report scenario (Table 5.10) show that the most time consuming operations are proof creation and total encryption/decryption durations. It is also seen that the IoT device's operations take longer time, due to low processing power. This results in the increase of the overall latencies for both scenarios. However, IoT device performs more encryption/decryption operations during the successful update scenario, which results in a 200ms difference between the two scenarios.

Lastly, the comparison of the test results presented in this thesis with the preliminary test results that were presented in [5] are given in the figures above. In Figure

5.4, it is seen that error report scenario did not get affected from the encryption overhead much, and the two trend lines are parallel. This is a result of fast encryption/decryption operation on Vendor's side and very little work done in the IoT device's side. As mentioned before, the successful update and the failed update results are affected from the encryption overhead, which can be seen in Figure 5.5 and Figure 5.6 respectively. This means when the number of IoT devices performing encryption operation increase, the overall latency increases with a bigger slope, yet still remaining on the linear trend line. This is similarly caused by the processing power of IoT devices, since in update scenarios IoT devices process much more messages and perform more decryption operations. However, for the mixed update scenario, the total number of devices communicating with HMS stay the same. This results in a trend line almost parallel to the preliminary results' trend line, which can be seen in Figure 5.7. The only difference between the two lines is caused by the average encryption process overhead and small relaying duration.

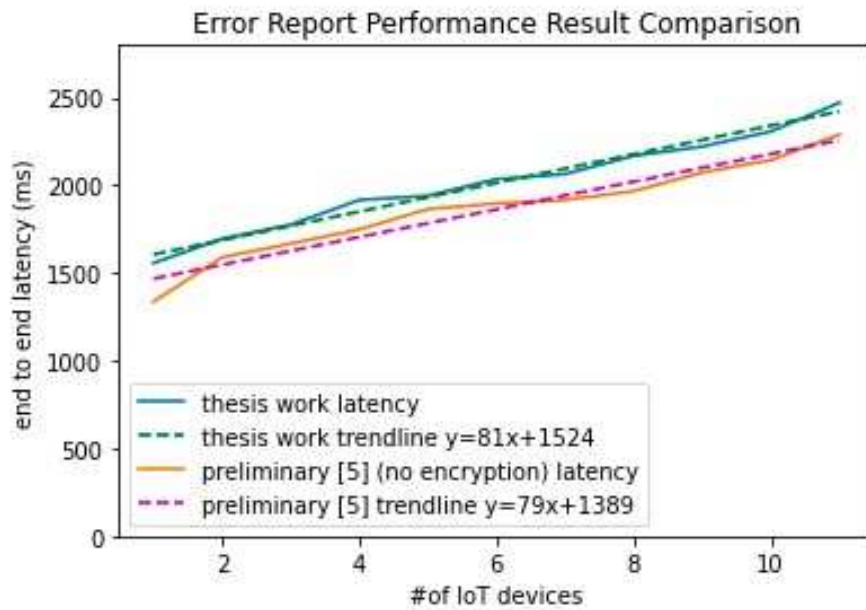


Figure 5.4 Error Report test performance comparison

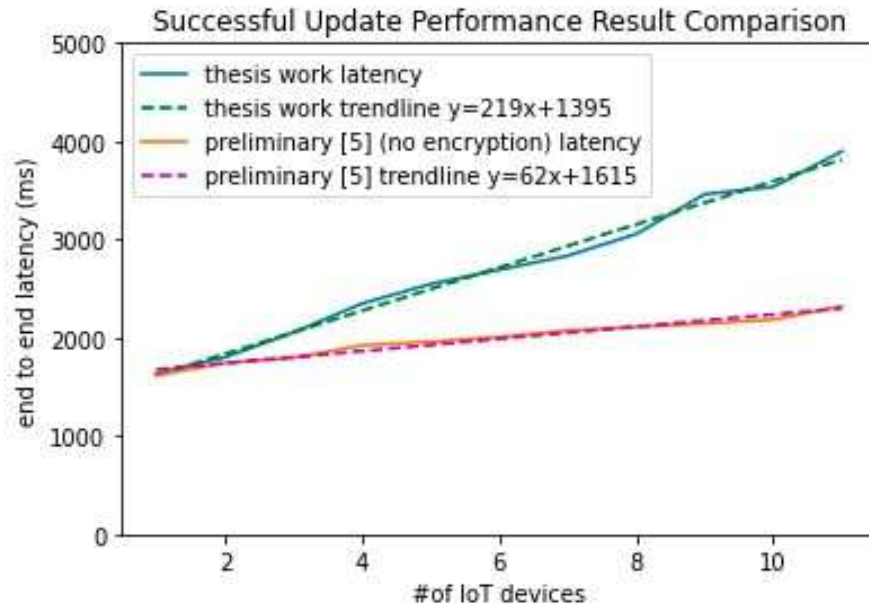


Figure 5.5 Successful Update test performance comparison

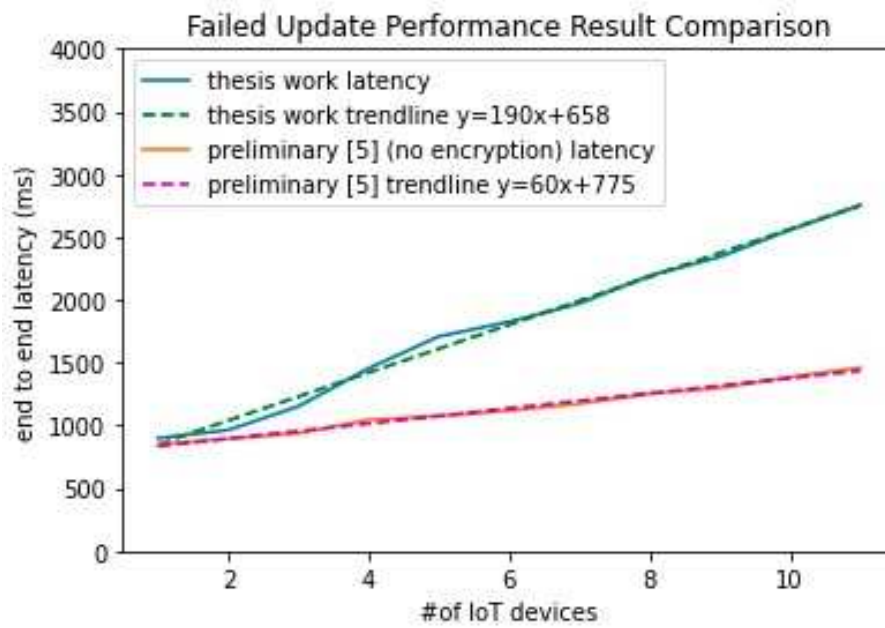


Figure 5.6 Failed Update test performance comparison

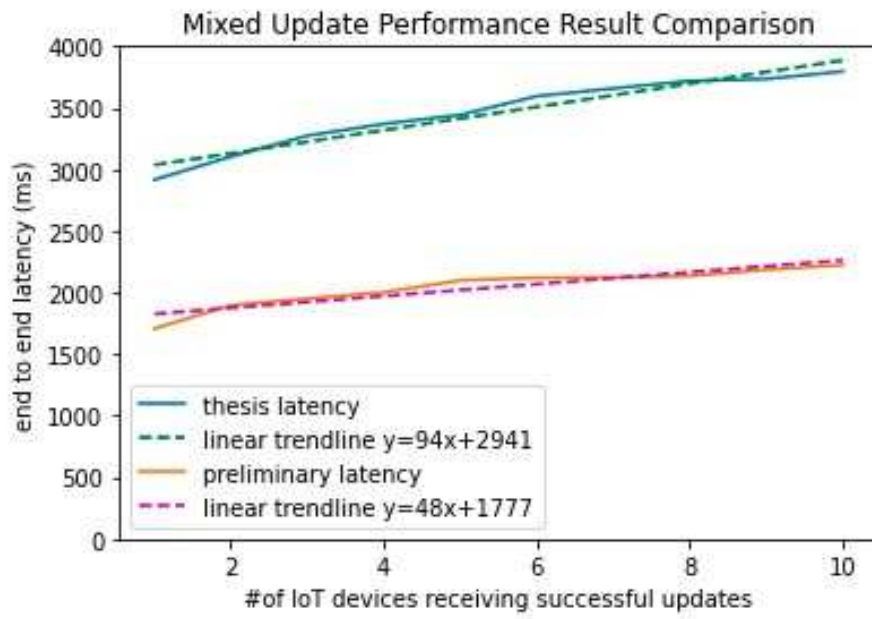


Figure 5.7 Mixed Update test performance comparison

6. SECURITY ANALYSIS

For the security validation and analysis of our protocols, we used SPAN [9], which is a security protocol animator for Avispa [10]. SPAN translates the provided specification and interactively builds a message sequence chart of the protocol execution. Then, it automatically builds attacks on the provided specification using the intruder mode. Using SPAN tool, we analyzed and validated the security of the initialization and the mutual verification and authentication phases of the proposed protocol. The simulation of initialization protocol can be found below in Figure 6.1. The protocol simulation is written in CAS+ [29] language, hence the appropriate syntax is used to define the protocol. The details of this syntax, which is different from the syntax that we used in our protocol descriptions throughout the thesis due to technical reasons, can be found in the CAS+ manual [29]. Also, both of the CAS+ specifications for the initialization and mutual verification and authentication protocols can be found under our GitHub repository [30].

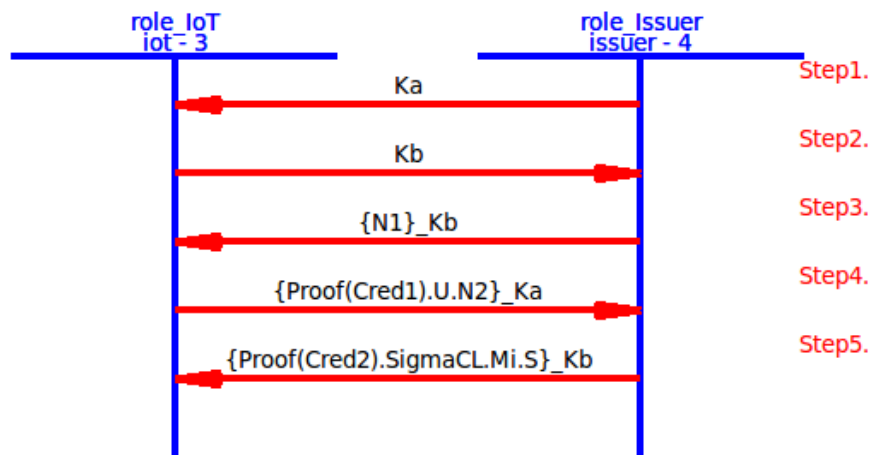


Figure 6.1 Initialization protocol simulation using SPAN tool

6.1 Security Analysis of the Credential Issuance Protocol

The intruder simulation of the initialization phase is tested for authentication of the parties using Cred1 and Cred2 (credentials of the parties) and the validation of the Proofs generated using these credentials. This intruder simulation can be seen in Figure 6.2, where the intruder tries to pose as IoT and Vendor at the same time, performing a man in the middle attack. In the simulation where the intruder tries to be authenticated by the Issuer, the intruder cannot generate $Proof(Cred1)$ since they do not know the credentials of IoT. Also they cannot trick IoT into thinking that they are the issuer since they do not know the credentials of the Issuer; and thus cannot generate $Proof(Cred2)$ either. This attack attempt fails as the intruder does not gain any knowledge about the contents of the messages. Also the intruder cannot alter any message during the communication, since they cannot decrypt the messages. During the attack, both parties understand that they are not communicating to each other so they fail the authentication process. The attack analysis on OFMC [31] and ATSE [32] modes prove that this initialization phase is safe.

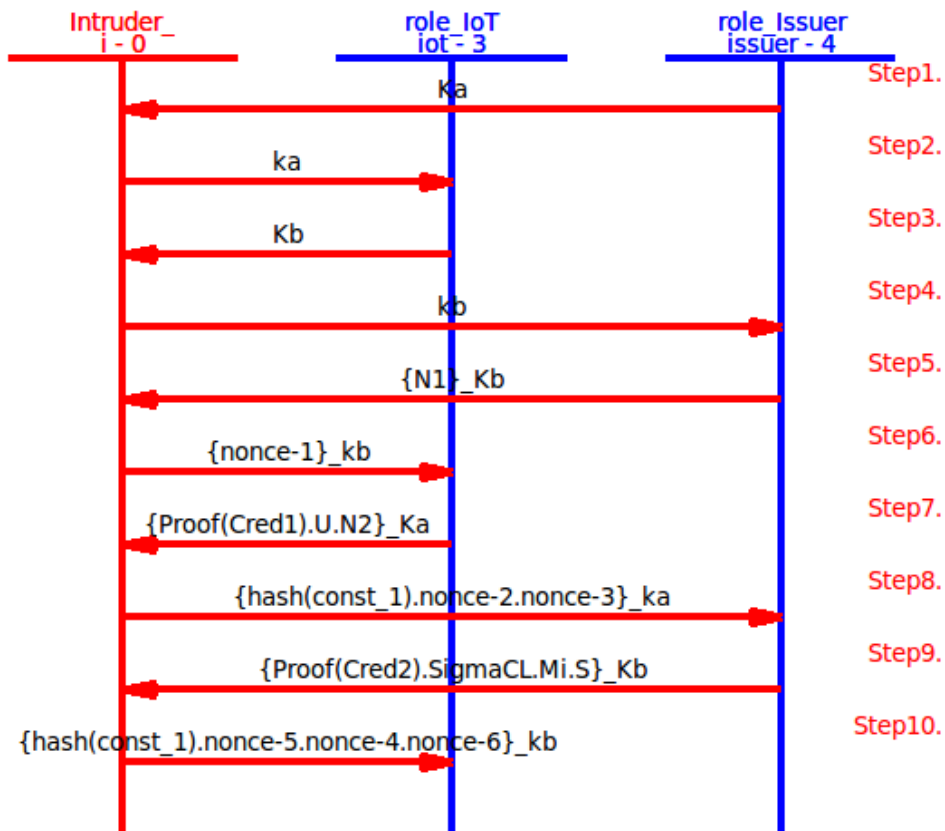


Figure 6.2 Initialization protocol intruder simulation using SPAN tool

6.2 Security Analysis of the Mutual Authentication Protocol

The communication simulation of the mutual verification and authentication protocol is shown in Figure 6.3. This simulation only shows the normal communication between the parties, where there are no adversaries involved. We also performed a number of simulations where the intruder can try to pose as Vendor, HMS or IHG. In either of these three cases the intruder cannot decrypt any of the messages, since the symmetric keys (Kih, Kvh, GrpKy) are distributed securely using the initialization protocol. The goal is to verify that the vendor and IoT are both legitimate by verifying the proofs, and most importantly, secrecy of the message. All of the messages except for the ones between IHG and IoT are communicated using the usual Dolev-Yao channel [33], which is not read or write protected. Since the IoT network inside of the house is secure and IoT devices can only communicate with IHG, this channel is read and write protected, and intruders cannot pose as an IoT device. One example of an intruder trying to pose as the Vendor during the mutual verification and authentication protocol is given below in Figure 6.4.

As it can be seen in the simulation, the intruder receives all of the messages addressed to the Vendor. After receiving the message, they can try to change bits of the message and they relay this message to the Vendor. When the Vendor sends a message to the HMS, the intruder receives this message and relays it to HMS. In both ways of communication, intruders may receive these messages but they cannot gain any knowledge about the contents of the message. The symmetric key encryption ensures that the messages are only available to the key holders. As mentioned before, intruders may try to change some bits of the messages, which are represented by the word “nonce-xx” in Figure 6.4. This attempt also fails since after every decryption operation, the users verify the proofs and/or other information in the messages. As a result, no matter how the intruder tries to pose as one of the parties of the communication, they cannot authenticate as any of the users, or they cannot reach the messages.

Figures 6.5 and 6.6 show that both the initialization protocol and the mutual verification and authentication protocol are safe according to the OFMC [31] specification. The safety of protocols are also analyzed and verified according to the ATSE [32] specification, which is not shown in the figure. Our results can be replicated easily using the SPAN tool with the codes provided in the GitHub repository [30].

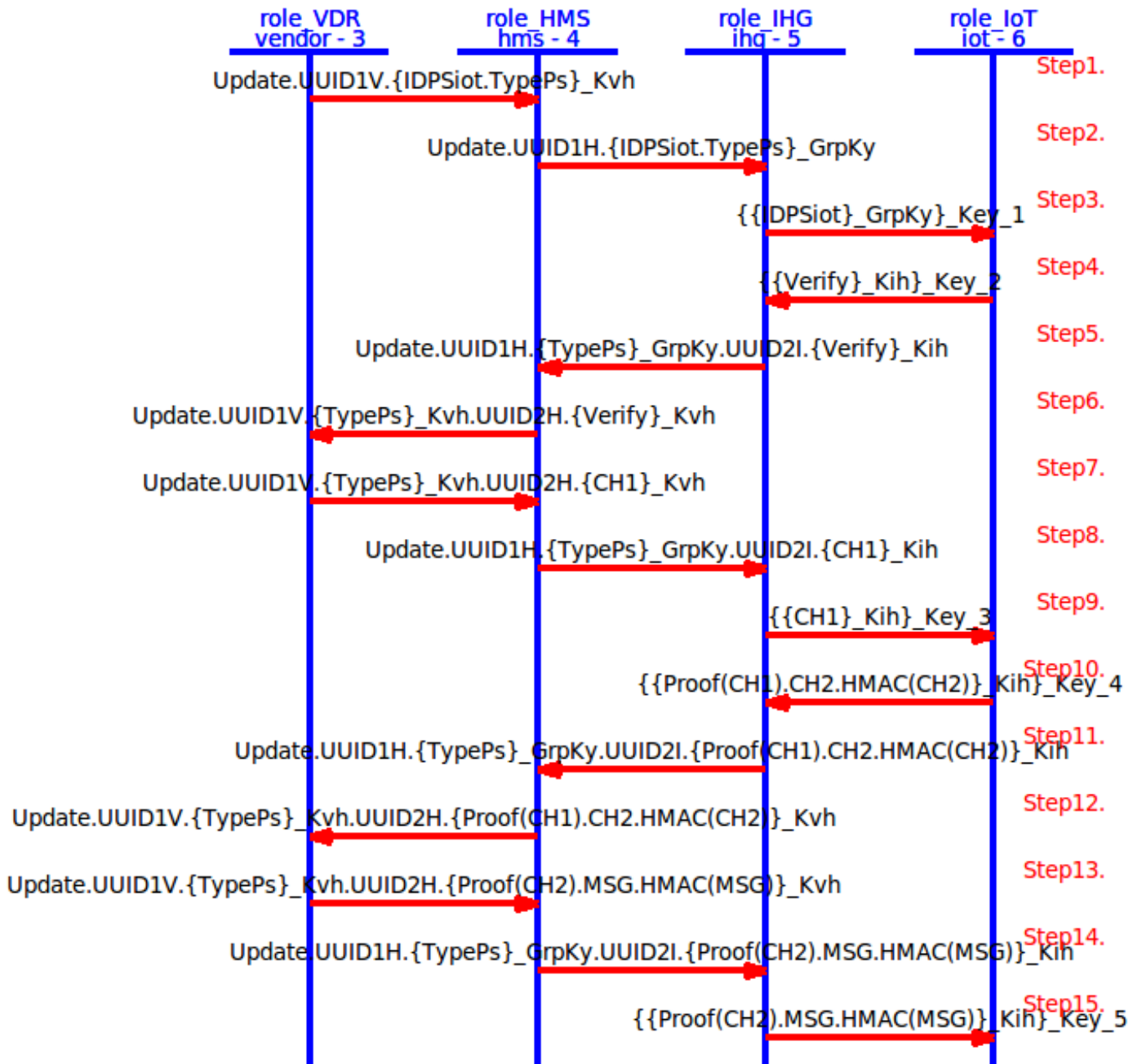


Figure 6.3 Mutual verification and authentication protocol simulation using SPAN tool

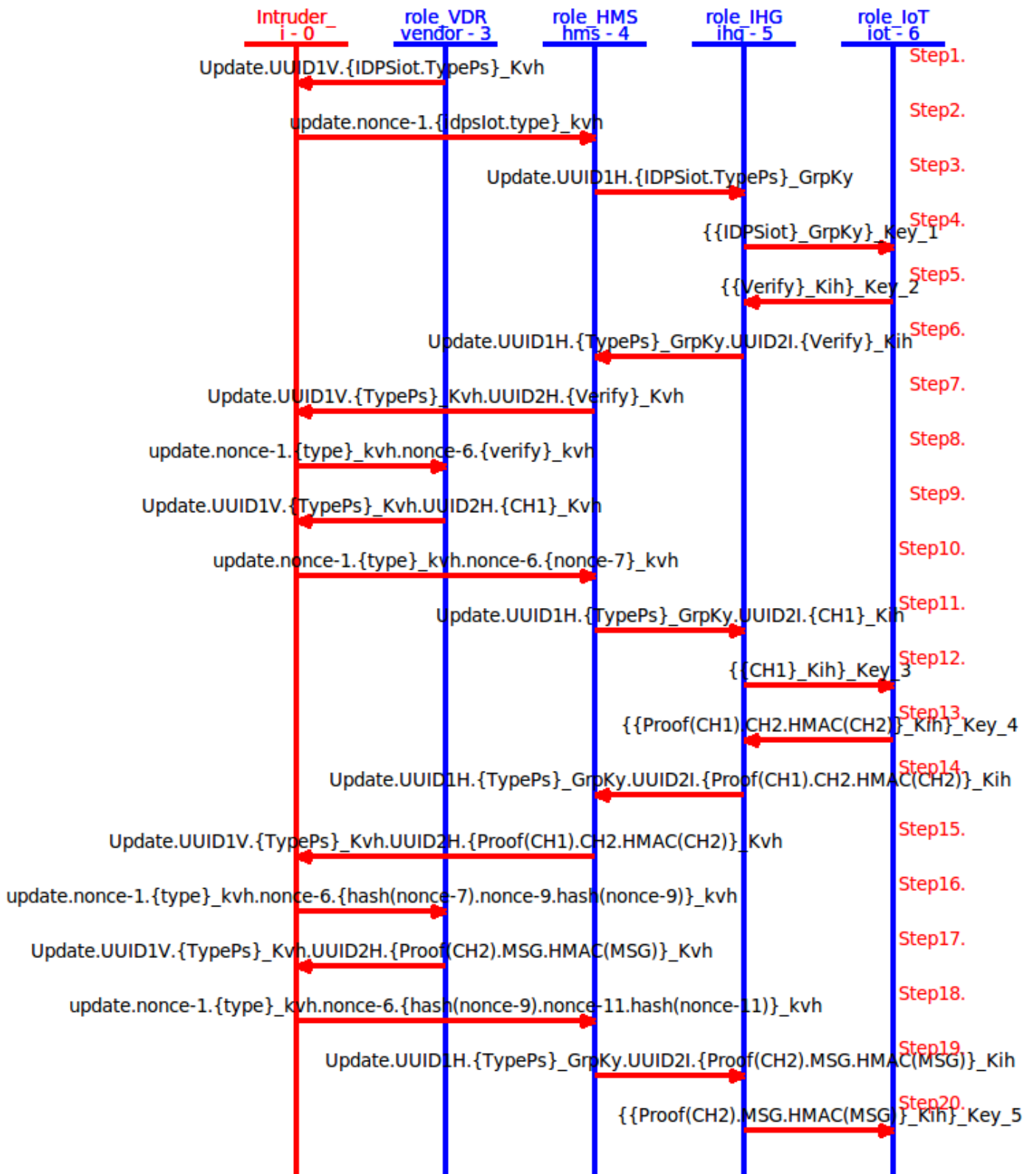
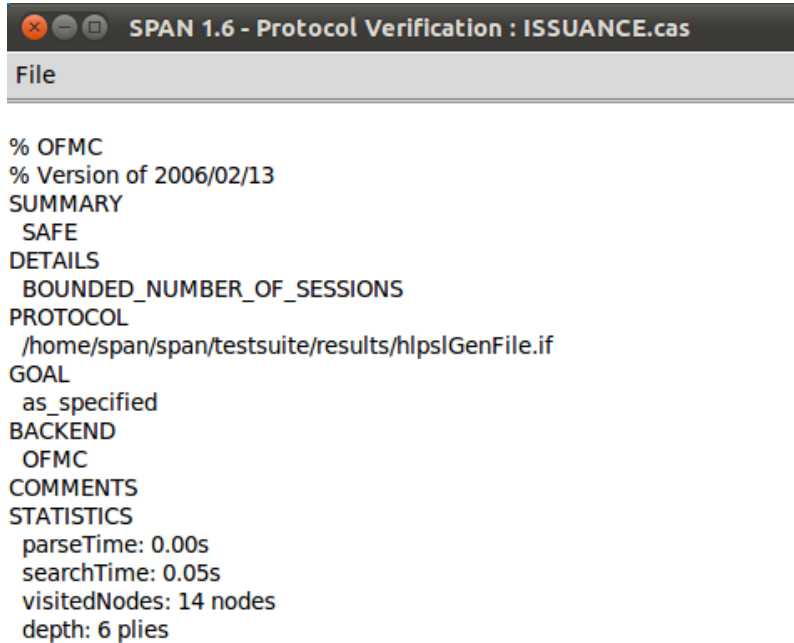
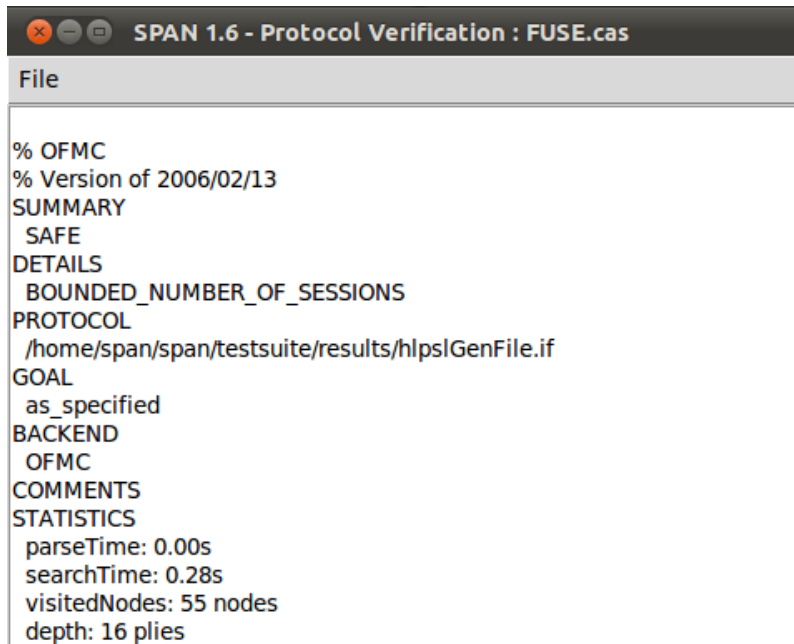


Figure 6.4 Mutual verification and authentication protocol intruder simulation using SPAN tool



```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /home/span/span/testsuite/results/hlpslGenFile.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.05s
  visitedNodes: 14 nodes
  depth: 6 plies
```

Figure 6.5 Key sharing protocol safety analysis



```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /home/span/span/testsuite/results/hlpslGenFile.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.28s
  visitedNodes: 55 nodes
  depth: 16 plies
```

Figure 6.6 Mutual verification and authentication protocol safety analysis

6.3 Discussion

In our model, the attacker cannot learn or alter the messages. This is explained in the below sections further. Since we use a publish-subscribe mechanism via MQTT, one needs to guess or possess the UUID's used in the topic, as well as the encryption keys that are used to encrypt the messages. The master keys used in the model stay the same for all messages, but the IV always gets updated after each session. This results in the topic changing after each session, including the encrypted device brand and type. We are confident in that sense that first it is unfeasible for the attacker to guess the topic and listen to the conversations.

However, one vulnerability of our system is that the fake Proofs take less time to be generated than the actual Proofs. This results in overall less end-to-end latency when all of the devices are failing i.e. none of the devices in the household are actually receiving an update. We assume that the attacker does not know how many devices are present in a HAS, but if the attacker has that information, it would be possible to eliminate the brand and type information. For example if the Samsung Vendor is sending updates to TV's, and none of the devices in the household is a Samsung TV, then the overall latency will be significantly lower than the case where there's a Samsung TV receiving the update. By doing so in multiple sessions, attacker can narrow down the devices' information in the long term, causing a privacy leak.

7. CONCLUSION

In this thesis, we proposed a privacy preserving identification protocol for Home Automation Systems (HAS). The proposed protocol employs Vendors, IoT Devices, an Innovative Home Gateway (IHG) and a HAS Management System (HMS). IoT devices only connect and converse with the IHG's. IoT devices are protected from outside users, and only users that have been authorized by IHG can send messages to IoT devices. We use Idemix anonymous credential system [13] for authentication of the entities in the system. We also use the Verifiable Encryption scheme [6] of Idemix, to carry out the key sharing and credential issuance process. Issuer, which is the trusted party in our system also known as the HMS, grants credentials to the users so that they can authenticate themselves to other parties. At the end of this issuance protocol, the master keys that are used for symmetric encryption are shared with the User. HMS stores the master keys of each entity along with their identity so that it can work as an obfuscation mechanism during the mutual verification and authentication protocol.

Mutual verification and authentication protocol is carried out for two types of communications: update and error report. During the update scenario, Vendor initiates the protocol and during the error report scenario IoT device initiates the protocol. In both scenarios, both parties authenticate themselves and verify the other party's identity mutually. Also during both scenarios, both of the parties only communicate with HMS, and HMS handles the topic switching and message relaying. HMS changes UUID's [8], in the topic, and decrypts the incoming message with the sender's key and encrypts it again with the receiver's key.

We conducted experiments for all of the scenarios, which are error report, update, failed update, where fakeProof's [1][5] are used and mixed cases for successful and failed updates, in addition to other privacy preserving features of the protocol. All of the results are promising and show scalable latency values. Moreover, we provide a security analysis of the system including the credential issuance phase and mutual authentication phase, using the SPAN [9] tool. Our system is proven to be secure according to OFMC [31] and ATSE [32] specifications.

BIBLIOGRAPHY

- [1] S. Gur, S. Demir, S. Simsek, and A. Levi, “Secure and privacy-aware gateway for home automation systems,” in *13th International Conference on Security of Information and Networks*, pp. 1–10, 2020.
- [2] T. Xu, J. B. Wendt, and M. Potkonjak, “Security of iot systems: Design challenges and opportunities,” in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 417–423, 2014.
- [3] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, “Tot: Internet of threats? a survey of practical security vulnerabilities in real iot devices,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019.
- [4] J. M. Batalla and F. Gonciarz, “Deployment of smart home management system at the edge: mechanisms and protocols,” *Neural Computing and Applications*, vol. 31, no. 5, pp. 1301–1315, 2019.
- [5] S. Demir, “Idemix based anonymization for home automation systems,” Master’s thesis, Sabancı University, Istanbul, Turkey, 2021.
- [6] J. Camenisch and V. Shoup, “Practical verifiable encryption and decryption of discrete logarithms,” in *Advances in Cryptology - CRYPTO 2003* (D. Boneh, ed.), (Berlin, Heidelberg), pp. 126–144, Springer Berlin Heidelberg, 2003.
- [7] R. A. Light, “Mosquito: server and client implementation of the mqtt protocol,” *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.
- [8] P. J. Leach, R. Salz, and M. H. Mealling, “A universally unique identifier (uuid) urn namespace.” RFC 4122, 2005.
- [9] Y. Glouche, T. Genet, O. Heen, and O. Courtaf, “A security protocol animator tool for avispa,” in *In ARTIST-2 workshop*, 2006.
- [10] 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*, pp. 281–285, Springer, 2005.
- [11] S. Mödersheim and L. Vigano, “The open-source fixed-point model checker for symbolic analysis of security protocols,” in *Foundations of Security Analysis and Design V*, pp. 166–194, Springer, 2009.
- [12] M. Turuani, “The cl-atse protocol analyser,” in *International Conference on Rewriting Techniques and Applications*, pp. 277–286, Springer, 2006.
- [13] J. Camenisch and E. Van Herreweghen, “Design and implementation of the idemix anonymous credential system,” in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 21–30, 2002.

- [14] P. Rogaway, “Nonce-based symmetric encryption,” in *Fast Software Encryption* (B. Roy and W. Meier, eds.), (Berlin, Heidelberg), pp. 348–358, Springer Berlin Heidelberg, 2004.
- [15] J. Camenisch and A. Lysyanskaya, “A signature scheme with efficient protocols,” in *Security in Communication Networks* (S. Cimato, G. Persiano, and C. Galdi, eds.), (Berlin, Heidelberg), pp. 268–289, Springer Berlin Heidelberg, 2003.
- [16] H. Krawczyk, M. Bellare, and R. Canetti, “Rfc2104: Hmac: Keyed-hashing for message authentication,” 1997.
- [17] S. Notra, M. Siddiqi, H. Habibi Gharakheili, V. Sivaraman, and R. Boreli, “An experimental study of security and privacy risks with emerging household appliances,” 10 2014.
- [18] I. Zavalysyn, N. O. Duarte, and N. Santos, “Homepad: A privacy-aware smart hub for home environments,” in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 58–73, 2018.
- [19] V. Puri, P. Kaur, and S. Sachdeva, “Data anonymization for privacy protection in fog-enhanced smart homes,” in *2020 6th International Conference on Signal Processing and Communication (ICSC)*, pp. 201–205, 2020.
- [20] P. Gope and B. Sikdar, “Lightweight and privacy-preserving two-factor authentication scheme for iot devices,” *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 580–589, 2019.
- [21] “Teach, learn, and make with raspberry pi.” Raspberry Pi, Retrieved July 5, 2021, from <https://www.raspberrypi.org>.
- [22] M. Blum, P. Feldman, and S. Micali, “Non-interactive zero-knowledge and its applications,” in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC ’88, (New York, NY, USA), p. 103–112, ACM, 1988.
- [23] D. Chaum, “Showing credentials without identification transferring signatures between unconditionally unlinkable pseudonyms,” in *Advances in Cryptology — AUSCRYPT ’90* (J. Seberry and J. Pieprzyk, eds.), (Berlin, Heidelberg), pp. 245–264, Springer Berlin Heidelberg, 1990.
- [24] S. Team and R. Switzerland, “Specification of the identity mixer cryptographic library version 2.3.0,” IBM Research, Zurich, 2010.
- [25] S. Goldwasser, S. Micali, and C. Rackoff, “The knowledge complexity of interactive proof-systems,” in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC ’85, (New York, NY, USA), p. 291–304, Association for Computing Machinery, 1985.
- [26] K. Sako, “Verifiable encryption,” in *Encyclopedia of Cryptography and Security* (H. C. A. van Tilborg and S. Jajodia, eds.), (Boston, MA), pp. 1356–1357, Springer US, 2011.

- [27] O. Goldreich and Y. Oren, “Definitions and properties of zero-knowledge proof systems,” *Journal of Cryptology*, vol. 7, no. 1, pp. 1–32, 1994.
- [28] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, p. 120–126, Feb. 1978.
- [29] R. Saillard and T. Genet, “Cas+ manual,” 2011. Retrieved on 5 July 2021 from http://people.irisa.fr/Thomas.Genet/span/CAS_manual.pdf.
- [30] S. Simsek, “Fusespan, github repository,” 2021. <https://github.com/sevvalboylu/FuseSPAN>.
- [31] S. Mödersheim and L. Viganò, “The open-source fixed-point model checker for symbolic analysis of security protocols,” in *Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures*, pp. 166–194, Springer Berlin Heidelberg, 2009.
- [32] M. Turuani, “The CL-Atse Protocol Analyser,” in *17th International Conference on Term Rewriting and Applications - RTA 2006* (F. Pfenning, ed.), vol. 4098 of *Lecture Notes in Computer Science*, (Seattle, WA/USA), pp. 277–286, Springer, Aug. 2006.
- [33] D. Dolev and A. Yao, “On the security of public key protocols,” *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.