# AUTHENTICATION PROTOCOLS FOR SMART HOMES

by

**Maninder Singh Raniyal**

B.Tech in Computer Science, Punjab Technical University, India, 2010

A Thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2018

## Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

# AUTHENTICATION PROTOCOLS FOR SMART HOMES



Master of Science
in Computer Science
Ryerson University

## Abstract

One of the IoT's greatest opportunity and application still lies ahead in the form of smart home. In this ubiquitous/automated environment, due to the most likely heterogeneity of objects, communication, topology, security protocols, and the computationally limited nature of IoT objects, conventional authentication schemes may not comply with IoT security requirements since they are considered impractical, weak, or outdated. This thesis proposes: (1) The design of a two-factor device-to-device (D2D) Mutual Authentication Scheme for Smart Homes using OTP over Infrared Channel (referred to as D2DA-OTP-IC scheme); (2) The design of two proxy-password protected OTP-based schemes for smart homes, namely, the Password Protected Inter-device OTP-based Authentication scheme over Infrared Channel and the Password Protected Inter-device OTP-based Authentication scheme using public key infrastructure; and (3) The design of a RSA-based two-factor user Authentication scheme for Smart Home using Smart Card.

# Acknowledgment

Foremost, I would like to thank Dr. Isaac Woungang for his incomparable guidance and continuous support. It is a wonderful and extraordinary experience to work with Dr. Isaac Woungang. His guidance was very helpful throughout my research. I am also thankful to the Department of Computer Science at Ryerson University for their support toward the accomplishment of my degree, special thanks to our program administrator, Norm Pinder, for his instantaneous support throughout my degree.

Wholeheartedly, I thank my family members: father, mother, my both sisters and brothers-in-law, and my brother and sister-in-law. Their support was very helpful and motivating, I feel without their support it was difficult to achieve my goal.

I am also thankful to my friends for their support at the Ryerson University, they keep me motivated to succeed in my goal.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| IR | Infra-red |
| IC | IR channel |
| OTP | One Time Password |
| D2D | Device to Device |
| DH | Diffie-Hellman |
| RSA | Rivest-Shamir-Adleman public-key cryptosystems |
| PKI | Public Key Infrastructure |
| D2DA-OTP-IC | D2D authentication schemes for smart homes using OTP over IC |
| PPIDA-IC | Password Protected Inter-device Authentication scheme over IC |
| PPIDA-PKI | Password Protected Inter-device Authentication scheme using PKI |
| RSA-ASH-SC | RSA-based Authentication scheme for Smart Home using Smart Card |
| HLPSL | High Level Protocol Specification Language |
| AVISPA | Automated Validation of Internet Security Protocols and Applications |
| SPAN | Security Protocol ANimator for AVISPA |
| FIPS | Federal Information Processing Standard |
| HG | Home Gateway |
| BLE | Bluetooth LE |
| LED | Light-Emitting-Diodes |
| VLC | Visible Light Communication |
| DoS | Denial-of-Service |
| NFC | Near Field Communications |
| MIMA | man-in-the-middle attack |
| BAN | Burrows-Abadi-Needham logic |
| SSH | Secure shell |
| CA | Certificate Authority |
| NIST | National Institute of Standards and Technology |

| | |
|---|---|
| OFMC | Onthefly ModelChecker |
| CLAtSe | Constraint Logicbased Attack Searcher |
| TA4SP | Tree Automata for the Analysis of Security Protocols |
| DY | Dolev Yao attacker model |
| DAC | Discretionary Access Control |
| ACL | Access Control Lists |
| LIRC | Linux Infrared Remote Control |
| AES | Advanced Encryption Standard |
| SATMC | SAT-based Model Checker |
| PAM | Pluggable Authentication Module |
| OS | Operating System |

# Chapter 1

# Introduction

This Chapter introduces the motivation and context of the research carried out in this thesis. The research problems and our solution approaches are highlighted, along with our contributions.

## 1.1 Motivation and Research Problems

Internet of Things (IoT) [1] has emerged as a driver for many applications in the area of smart homes, smart cities, to name a few. In IoT, devices (also referred to as objects) are often grouped into clusters, therefore, in order to enable secure communications across them, it is required that these objects be able to first authenticate with each other using cloud platforms or IoT-based communication protocols such as the constrained application protocol or the message queue telemetry transport [2], to name a few. However, most of these protocols do not possess inbuilt security mechanisms or rely on limited inbuilt single-factor authentication security mechanisms [2]. In addition, focusing on a particular class of authentication methods for IoT devices in the cloud, namely one-time password (OTP), it has been reported [3] that most existing OTP schemes are not applicable in the smart home context because their designs would have to be substantially adjusted to fulfill the IoT requirements, in particular smart home requirements. In such an environment, there are

essentially two types of authentication schemes: user authentication and device-to-device (D2D) authentication. In user-authentication schemes, some type of secrecy (known only by the user) or a unique biometric information of the user, is utilized for authentication purpose [4].

On the other hand, the design of D2D authentication schemes is challenging since devices are usually heterogeneous, and thus, they are expected to operate under various different protocols as well as being subject to various different resource constraints such as energy restriction and security constraints. In addition, most smart home systems make use of wireless technologies (such as ZigBee, Wi-Fi, to name a few) for communication purpose; and if an attacker can hold the related network packages, various types of attacks such as replay attack, man-in-the-middle, eavesdropping, masquerading, message modification, denial-of-service, reflection, impersonation, to name a few, can be launched [5] to compromise the security of the system. These facts have motivated the need for designing D2D multi-factor authentication schemes for IoT devices, in particular, smart home devices. This thesis proposes the design of two-level device-to-device authentication schemes for smart homes using OTP over an infrared channel (our so-called D2DA-OTP-IC protocol), along with a proof-of-concept in the form of a hardware solution [7]); and two password-proxy-based protocols (so-called PPIDA-IC and PPIDA-PKI schemes), which are enhancements to the D2DA-OTP-IC protocol [7].

Although smart homes filled with connected objects will make our lives more convenient by enabling the users to remotely control these objects in communication using, for instance, a voice command or simple push buttons, it is expected that this benefit will also entail serious security concerns from a remote user authentication perspective [6]. Focusing on password-based remote user authentication using smart card, which has been widely used as method (due its user friendliness and easy implementation) to verify the legitimacy of remote users over an insecure network, ensuring a proper access rights to system resources by users and validating the user's identity are major challenges, which justify a clear demand for

new layered security approaches for remote user authentication using smart card that can be resistant to various types of attacks including stolen verifier password, forgery attack, server spoofing attack, replay attack, password guessing attack, to name a few. In this respect, this thesis proposes a RSA-based two-factor user Authentication scheme for Smart Home using Smart Card (so-called RSA-ASH-SC) protocol.

## 1.2 Approaches

The design of our proposed D2DA-OTP-IC protocol [7] comprises two level. In the first level, the key exchange is performed using the DH key exchange protocol and a public key cryptography in order to validate the identity of devices. The second level relies on the use of infrared communication to distribute the OTPs among devices for authentication purpose. In the D2DA-OTP-IC protocol [7], storing the secret information may not be considered safe [8]. Therefore, to fulfill this requirement while complying with the Federal Information Processing Standard (FIPS) 4.7.5 standard [8], the design of the D2DA-OTP-IC scheme is modified to protect the secret and public keys, yielding the PPIDA-IC and PPIDA-PKI schemes, where these keys are protected using passwords and a centralized server is used to provide proxy-password services to the smart home devices. The design of our proposed RSA-ASH-SC relies on the use of a variant of RSA [10], where the message is encrypted and the anonymity about the device/user is achieved and maintained using a one-time-token.

The High-Level Protocol Specification Language (HLPSL) is used to model the proposed D2DA-OTP-IC, PPIDA-IC, and PPIDA-PKI schemes, and the Security Protocol Animator for (SPAN)/AVISPA (Automated Validation of Internet Security Protocol and Applications) tool is used for the formal analysis and verification of the efficiency of D2DA-OTP-IC, PPIDA-IC, and PPIDA-PKI schemes, in terms of achieving the goals of secrecy of secret keys and D2D mutual authentication, and integrity. A security analysis of the proposed

RSA-ASH-SC scheme is also conducted, showing that it is about 50% faster than the Om and Kumari scheme [10] and 15 times faster than few other RSA-based schemes [10] in terms of RSA decryption speed when the RSA key length is 2048. The RSA-ASH-SC scheme is also shown to maintain the anonymity of the user using a one-time token.

## 1.3 Thesis Contributions

The contributions of this thesis are as follows:

- Design of a two-factor device-to-device (D2D) mutual Authentication scheme for Smart Homes using OTP over Infrared Channel (D2DA-OTP-IC scheme), along with a proof-of-concept in the form of a hardware solution.
- Design of two proxy-password protected OTP-based schemes (PPIDA-IC and PPIDA-PKI) for smart homes, which are enhancements of the D2DA-OTP-IC scheme.
- Design of a RSA-based two-factor user Authentication scheme for Smart Home using Smart Card (RSA-ASH-SC).
- Formal analysis and verification of the security properties of the above D2DA-OTP-IC, PPIDA-IC and PPIDA-PKI protocols using the SPAN/AVISPA tool.
- Security analysis of the RSA-ASH-SC protocol.

## 1.4 Thesis Outline

The thesis is organized as follows:

- **Chapter 1** introduces the subject, motivation, and contributions of our research.
- **Chapter 2** presents some background information and related work pertaining to the scope of this thesis.

- **Chapter 3** describes the designs and modelling of the proposed D2DA-OTP-IC, PPIDA-IC and PPIDA-PKI protocols, as well as the design of the RSA-ASH-SC protocol.
- **Chapter 4** is devoted to the simulations and security analysis of the proposed schemes.
- **Chapter 5** concludes the thesis and highlights some of the future work that can be carried out further.

# Chapter 2

# Background and Related Work

This Chapter introduces the concept of authentication in IoT, in particular, smart homes. The security requirements for smart homes are highlighted, along with the types of authentication schemes for smart home networks. Some security metrics and tools for evaluating and analyzing authentication schemes are discussed, and the ones used in this thesis are highlighted. In addition, the class of security attacks on authentication protocols, which prevail in the context of our study is presented. Finally, some of the recent related work on authentication protocols for smart home environments and remote user authentication schemes for smart home using smart card are discussed.

## 2.1 Background

### 2.1.1 Need for Multi-Factor Authentication in Smart Home Environment

Authentication can be defined as a secure process or cryptographic mechanism to correctly identify the communicating parties over an insecure channel in order to prevent any illegal access to the system resources [2]. In conventional authentication schemes, a password is often associated with the user's identity in such process or cryptographic mechanism. The

client and the server have no limitations regarding the resources. All devices are able to run complex algorithms such as encryption, and they have no memory limitations. Putting this in the context of IoT or smart home environment, authentication refers to the process of validating a device's identity in communication while ensuring the reliability of the origin of the communication, keeping in mind the constraints associated with the device (such as limited processing power, battery, bandwidth, CPU, memory, network capabilities, to name a few).

Nowadays, with the rise of emerging trends of the Internet of Things (IoT) paradigm [11], which are considered as drivers for many applications in the area of smart homes, smart cities, to name a few, IoT devices (referred to as things' or objects' [11]) are typically grouped into clusters. Therefore, in order to design an efficient authentication scheme, it is necessary that a secured and integrated communications be primarily established between objects, in such a way that these objects can be able to authenticate each other afterwards, for instance, via a cloud platform using IoT communication protocols such as the Constrained Application Protocol, the Message Queue Telemetry Transport, to name a few [12]. However, most of these protocols do not have inbuilt security mechanisms, or they rely on limited inbuilt single-factor authentication security mechanisms [13]. In addition, our literature survey has revealed that most existing conventional OTP-based authentication schemes [3] have not been designed specifically for IoT [4]. This has motivated the need for multi-factor authentication schemes for IoT, in particular, smart home environments, which involve combining multiple factors such as smart card, biometrics, one-time-password (OTP), to name a few.

### 2.1.2 Smart Home Network Architecture in IoT

In a smart home environment (such as the one as depicted in Fig. 2.1), physical objects such as doors, temperature, alarms, alerts, appliances, fridge, to name a few, are equipped with the ability to operate over the Internet, for monitoring, collecting and sharing their

resources (in particular their data) [14].

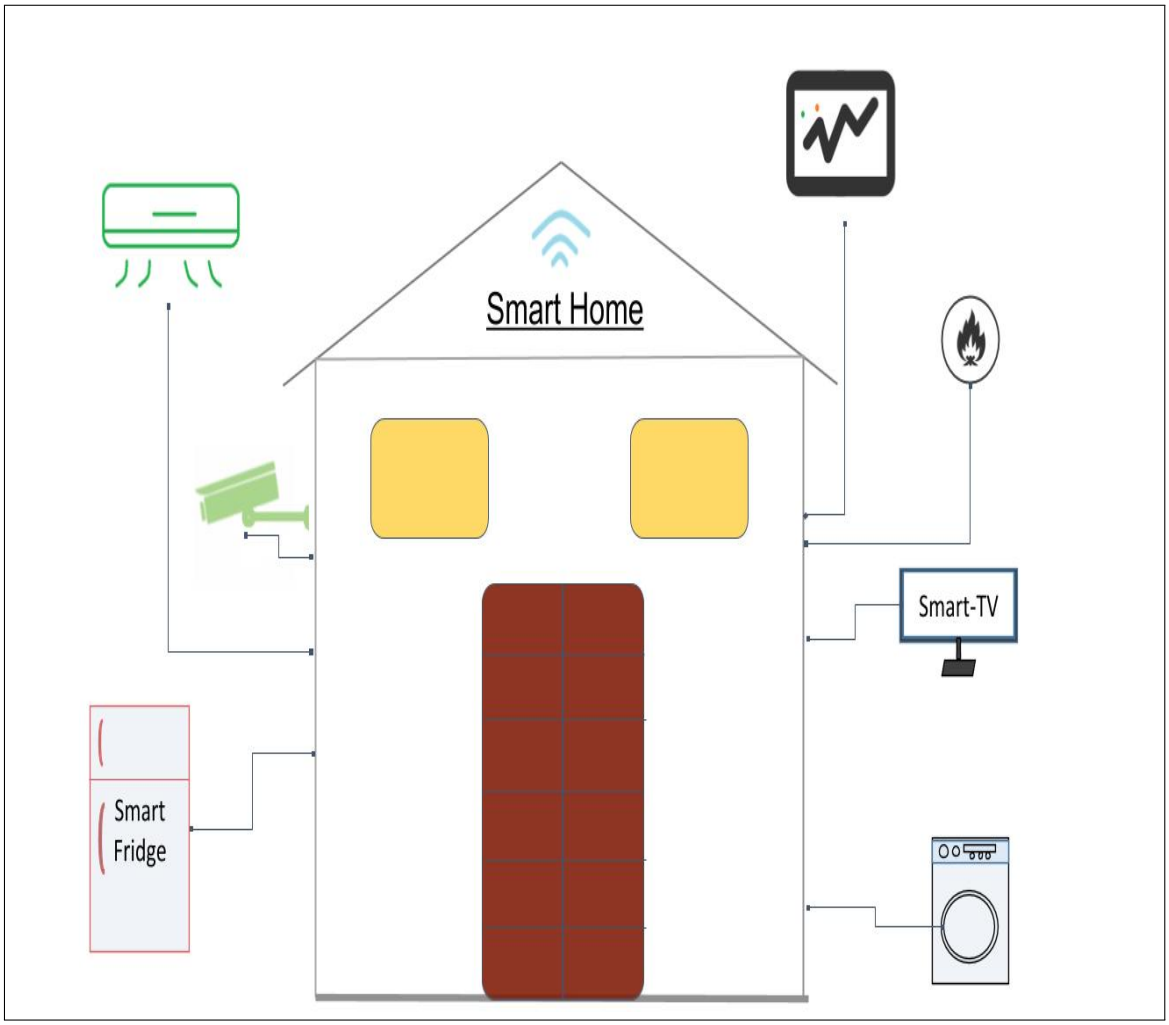


Figure 2.1: Example of a smart home.

As shown in Fig. 2.1, devices in a smart home are usually connected to each other through a Home Gateway (HG), an interface between the home network and the Internet. In order to control the smart home system, the HG has a routing functionality and is usually connected to the user interface using a mobile software, a well-mounted solution, to name a few. Some smart home implementations also include the use of a home server, which is responsible for authenticating the devices. A typical smart home network architecture in IoT is depicted in Fig. 2.2

Considering the architecture in Fig. 2.2, communication technologies in a smart home

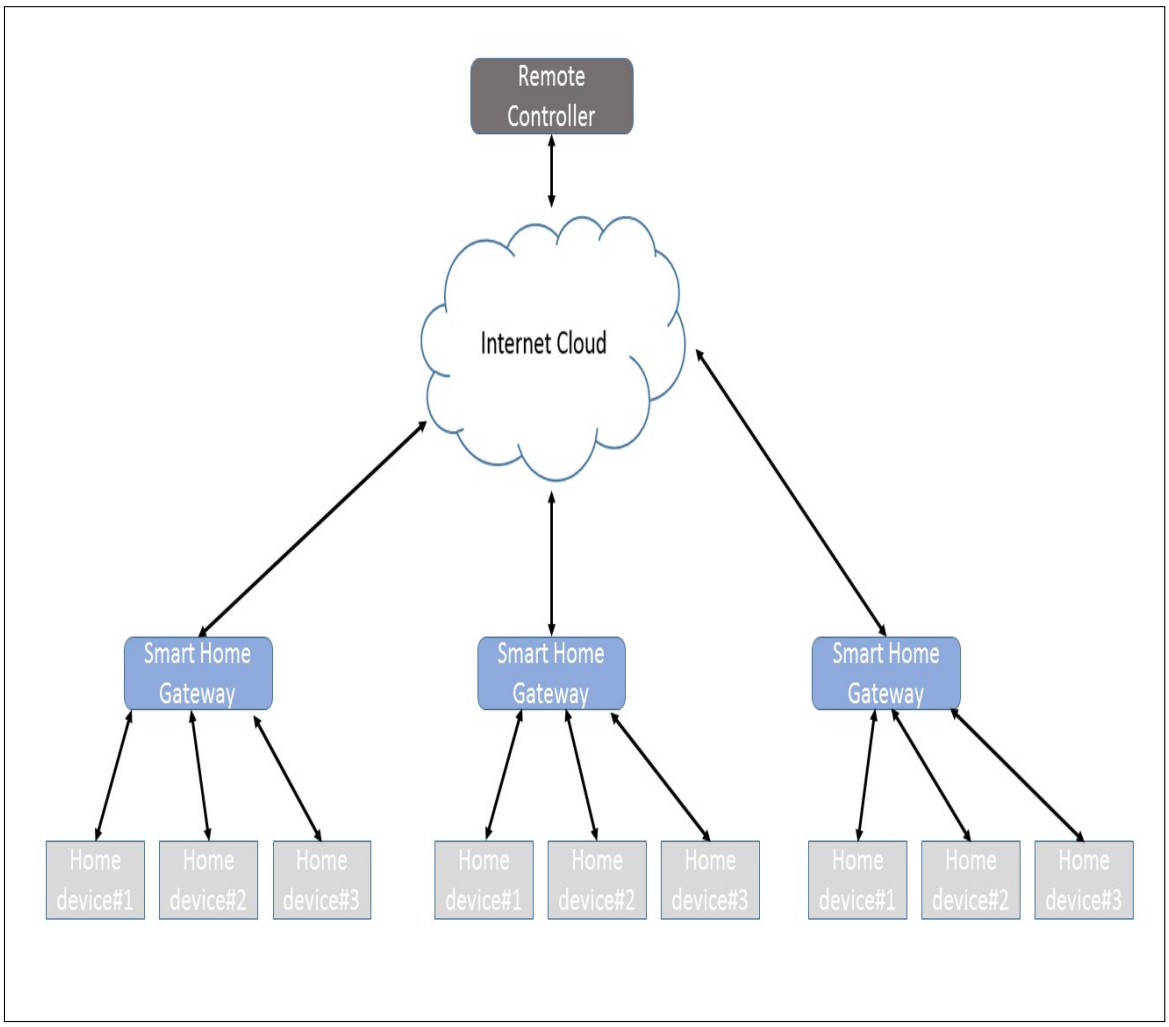


Figure 2.2: Typical Smart Home Network Architecture in IoT.

environment include radio waves which are electromagnetic waves with frequencies between 3 kHz to 300 GHz - and light waves communication where light is used as communication medium. The former is widely accepted and has the ability to penetrate through the wall or objects, making them suitable for smart homes while the latter is rarely utilized and usually cannot penetrate through walls or objects, limiting the communication space. Common radio wave technologies include X10, Ethernet, RS-485, 6LoWPAN, Bluetooth LE (BLE), ZigBee, Li-Wi, and Z-Wave. On the other hand, Li-Wi is a well-known example of light waves communication [15] where communication is achieved by turning the light ON or OFF. Indeed,

the light-emitting diodes (LEDs), in particular Visible Light Communication (VLC) can be used as a mean to achieve light communication because of their high performance to turn the light ON or OFF. However, one of the problems encountered with the use of VLC is that it requires the light to be constantly ON for the communication to happen. In a smart home environment, this requirement might not be achieved since in the night, the lights are likely to be turned OFF. Other types of light communication include dark visible light communication - which uses VLC for communication purpose - and infrared communication which uses Infrared light spectrum for the same purpose [16]. In the proof-of-concept of our proposed D2DA-OTP-IC protocol [7] in the form of hardware solution, we have used the infrared-light for inter-device communication purpose. However, any other light communication medium could have been used as well.

Finally, considering the architecture in Fig. 2.2 in the prospect of designing an authentication protocol for a smart home network, some minimal security requirements should be met, which include: confidentiality, availability, integrity, and mutual authentication.

- ***Confidentiality***: The smart home devices should maintain privacy of the data exchanged between devices/user. Only authorized user/device should have access to sensitive data and the data should be protected from unintended users. Typically, cryptography is used to address the confidentiality issue when dealing with an insecure network.
- ***Availability***: In a smart home, it is desirable that a service be available to the authorized users all the time. Typically, Denial-of-Service (DoS) attacks are used to make a service unavailable/unusable. In this type of attack, the attacker sends many false requests, with the goal to overload the server; in such a way that it can no longer serve the requests from legitimate users normally, or in such a way that the service is completely turned down.
- ***Integrity***: Data integrity is an important aspect of security. It is meant to ensure that

the is not manipulated by an unauthorized user. If an adversary is able to manipulate the data/message, the results of doing so could be devastating to the network.

- ***Mutual authentication (also referred to as two-way authentication)***: It occurs when both parties involved in the authentication validates each other's identity. During this process, an attacker can attempt to spoof the server by sending a false information in many ways, via the use of several types of attacks. Therefore, a mutual authentication mechanism usually comes with its set of attacks that its design can enable to prevent, and for which a security analysis has to be conducted to verify the effectiveness of the said protection.

### 2.1.3 Types of Authentication Schemes for Smart Home Environments

Considering the above-mentioned smart home network architecture in IoT, there are essentially two types of authentication schemes: user authentication and D2D authentication. In user-authentication schemes, some type of secrecy (known only by the user) or a unique biometric information of the user, or a combination of these metrics, is utilized for authentication purpose [4]. On the other hand, the design of D2D authentication schemes is challenging since devices are usually heterogeneous. Thus, they are expected to operate under various different protocols as well as being subject to various different resource constraints such as energy restriction and security constraints. In addition, most smart home networks make use of wireless technologies (such as ZigBee, Wi-Fi) for communication purpose; and if an attacker can hold the related network packages, various types of attacks such as replay attack, man-in-the-middle, eavesdropping, masquerading, message modification, denial-of-service, impersonation, can be launched [5], [17] to compromise the security of the system.

As far as user remote authentication for smart home environments is concerned, our liter-

ature survey has revealed that techniques based on passwords [18] are typically weak and are not considered very secure. To improve this weakness, other techniques have been proposed such as those relying on the user's personal memory (or digital memory) [19], those relying on the use of Kerberos authentication to implement a single-sign-on [20], those relying on mobile OTP or OAuth [21], an open standard authentication and authorization protocol, in which the user receives a password that should be used within short time period after its generation, otherwise the OTP expires; those relying on smart cards [22], those relying on biometrics [23], those involving transactions based on Near Field Communications (NFC) [24], and those relying on RSA [9], [10], to name a few. The novel RSA-ASH-SC scheme proposed in this thesis belongs to this later class of protocols since its design relies on the use of a variant of RSA (so-called Om and Kumari scheme [10]), in which the anonymity of the user/device is achieved by using a one-time-token.

### 2.1.4 Considered Security Attacks

Considering the aforementioned smart home network architecture in IoT, any adopted type of authentication scheme should be designed by taking into account the security attacks that could be prevented using its design features, as well as common security requirements for the smart home automation network. In this thesis, for the later, we have considered: (1) *user anonymity or privacy protection* - this property should be maintained in such a way that an intruder cannot trace the user's identity. For instance, some cryptographic techniques can be applied on the user's identity before sending it over a public channel; (2) *secrecy* - an intruder should not be able to compute a valid session key from previously eavesdropped messages. A potential solution for this is to ensure the freshness of the session key at each computation time; and (3) *No verification table should be necessary* - this refers to storing secret values in a table. For instance, this can be avoided by ensuring that none of these values are stored in any format at the server side. For the former, following the above re-

quirements, the security attacks that have been considered in the authentication schemes proposed in this thesis include: man-in-the-middle attack (MIMA), eavesdropping attack, masquerading/impersonation attack, replay attack, message modification attack, dictionary attack, stolen card attack, password-guessing attack, forgery attack, Denial of Service attack, forward secrecy attack, brute force attack, reflection attack, and multi-protocol attack [5]. To evaluate and/or perform a security analysis of any authentication schemes with respect these attacks (and others), some security metrics can be considered, along with existing tools for analyzing or validating the security features of such schemes. In this thesis, we have used the Security Protocol Animator (SPAN) for Automated Validation of Internet Security Protocol and Applications (AVISPA) tool (referred to a SPAN/AVISPA) [25]. Other tools (not used here) include the Burrows-Abadi-Needham (BAN) logic [26]. Examples of security metrics used in this context include: secrecy of the secret keys, mutual authentication, integrity, types of security attacks (case of the D2DA-OTP-IC, PPIDA-IC and PPIDA-PKI schemes) and computation time, type of security attacks, computational performance, and RSA decryption speedup (case of the proposed RSA-ASH-SC remote user authentication scheme).

## 2.2 Related Work

Several authentication schemes for smart home and IoT as well as remote authentication schemes for smart home network in IoT are discussed.

### 2.2.1 Related Work on Authentication Schemes for Smart Home-/IoT Environments

Authentication schemes for IoT environments have been intensively investigated in the recent years. Representative ones are described as follows.

In [4], Madsen reports on new mechanisms and standards that can be used to authenti-

cate IoT actors in a health care devices architecture. Related challenges and opportunities are also described.

In [13], Shivraj et al. proposed an end-to-end authentication scheme for IoT based on Lamport's OTP. In their scheme, the OTP generation is modelled as a 4 steps process, namely: (1) the Setup phase - where some cryptographic parameters are involved in the PKI generation to optimize some computations over elliptic curve; (2) the Extract phase - where public and private keys are assigned to registered IoT applications and devices; (3) the Generate phase - where both the requests and data are transferred between IoT devices and a IoT cloud center using the aforementioned keys, and the required information are extracted; and (4) the Validate phase - where the OTP received at the cloud center is checked against that originally sent by the device. It is proved that the hardness of the OTP generation scheme is equivalent to solving the computational Diffie-Hellman problem [51].

In [27], Wilson proposed a D2D mutual authentication protocol for IoT based on asymmetric cryptography providing that a shared secret session key be given. The public key cryptography is meant for data exchange purpose between devices. The scheme is composed of three phases, namely: a pre-deployment phase where the manufacturer prepares and equip the devices with the necessary firmware; a deployment phase where the authentication information and a device virtual identity are collected; and an authentication phase where a public key cryptography scheme is designed based on the aforementioned parameters and is invoked to establish a secure data exchange between two devices after these devices have mutually authenticated themselves to each other.

In [28], Yao et al. proposed a lightweight multicast authentication scheme for small scale IoT applications based on the absorbency property of the original Nyberg's fast one-way accumulator [30]. In their scheme, the MAC is exploited to achieve such design; and in terms of security analysis, seven cardinal properties that are required by multicast authentication for resource-constraint applications are assessed and validated.

In [29], Hernandez-Ramos et al. proposed a set of lightweight authentication and autho-

rization mechanisms which can be used to assure authentication and authorization functionalities on constrained smart objects. These mechanisms are combined with some standard technologies to address different security planes of the life cycle of an IoT device within an introduced Architectural Reference Model-compliant security framework [30]. The suitability of the proposed framework for IoT scenarios is demonstrated through experiments, showing promising results.

In [31], Sharaf-Dabbagh and Saad proposed an authentication framework for IoT objects, which uses the unique fingerprints of these objects to differentiate between normal changes in fingerprints and security attacks. In their scheme, the IoT objet set is divided into multiple hierarchies based on the types and geographic locations of the fingerprinting features. Each partition is then modelled as a set of distributions on which a learning approach is performed to extract the knowledge from the different fingerprints, thereby identifying the IoT objects. From simulations, it is found that their scheme outperforms the conventional authentication schemes by up to 8% in terms of authentication performance.

In [32], Shin et al. proposed an authentication scheme for smart home networks, which uses a group key shared among the devices to encrypt and decrypt all messages from all devices. A digital DNA is also used to differentiate the devices against those pertaining to other smart home networks. The DNA (in the form of a magical number) generated by the authentication server or home server is used to distinguish the devices that belong to the different smart homes. However, if the group key is compromised, so will be the entire system. This scheme has been shown to be vulnerable to replay attacks since they do not use any timestamps or other mechanisms to prevent these types of attacks. In their scheme, the initial registration phase assumes that the communication channel is secure while registering a new device.

In [33], Kumar et. al. here proposed a short token-based authentication scheme which can be used for secure key establishment in smart home environments. In their scheme, the device registration is performed offline, which is a quite cumbersome process. This scheme

is not fully protected against DoS attacks and the communication using it cannot be kept anonymous. Moreover, devices cannot communicate with each other.

In [34], Santoso et al. proposed an elliptic curve cryptography-based authentication scheme for smart homes, where the authentication mechanism involves the use of a mobile device. First, the user gets the identity information (such as device ID), and a secret key for the device, then passes them to the IoT device, which is connected automatically to the mobile device, which in turn shares this information with the gateway. Using this mechanism, the device attempts a connection to the gateway, and if the gateway has the secret key, a second level authentication process starts and a secure connection is established. However, the secret information is stored on the device itself, which is an issue. Their scheme does not keep the network anonymous, and the communication is handled via a Wi-Fi channel, which makes it vulnerable to DoS and replay attacks.

In [35], various different forms of secure shell (SSH)-based authentication schemes are analyzed and an SSH agent forwarding scheme is proposed, where a user can log in to a second hop system without storing the keys into the first hop system, and the private keys never leaves the user's system at any time. In such scheme, it is assumed that the public keys are pre-shared by the remote system in advance. However, this approach does not allow the application in the remote device to authenticate itself to another remote service as there is no private keys for each system/application. The application has no way to forward the challenge to the user's system so that the ssh agent can create and respond to it.

In [36], Park and Kang proposed an inter-device authentication scheme along with a session-key distribution protocol for IoT devices which are only equipped with encryption modules. The proposed approach relies on the assumption that the secret key and encryption function are securely share among devices and saved at the device setting step. Based on this, an authentication requestor and a responder can authenticate each other using an authentication and session-key agreement procedure that enables them to share a different session key for each authentication session without having to exchange new parameters or

without the need to invoke an additional method. The proposed authentication scheme has been shown to protect against man-in-the-middle attacks, replay attacks, and wiretapped secret-key attacks.

In [37], Kang et al. proposed a secure mutual D2D authentication scheme in smart homes, which is based on the use of zero-knowledge proof instead of a secret value between the sensor node and the home gateway. In their scheme, a master key exchange produced in the registering phase is utilized for securing the channel between two devices before a message gets send from one device to another.

In [38], Vaidya et al. proposed a device authentication mechanism for smart home area network jointly with a key establishment method, based on elliptic curve cryptography and a self-certified public key following the smart energy profile standard for home area networks [21]. Their proposed scheme is composed of (i) pre-deployment phase - where every device must securely obtain an implicit certificate from a Certificate Authority (CA) prior to enter an initialization phase, in which long-term private/public keys are computed. Based on this computation, (ii) a two-pass authenticated key agreement protocol between two entities is established to achieve the goal of device authentication.

The D2DA-OTP-IC [7], PPIDA-IC and PPIDA-PKI schemes proposed in this thesis are fundamentally different from the above-discussed schemes by their design features. In addition, unlike the above-discussed schemes, a proof-of-concept of the proposed D2DA-OTP-IC scheme in the form of a hardware solution is provided, which uses Raspberry-Pi, Infrared circuit and the OpenSSH protocol.

### 2.2.2 Related Work on Remote User Authentication Schemes Using Smart Card

Some of the notations used in this subsection are as follows:

$U_i$: $i^{th}$ user

$PW_i$: $i^{th}$ user's password

$ID_i$: $i^{th}$ user ID

$CID_i$: Smart Cards identifier

$h_i$: secret information create by the card issuer at the registration time

$S_i$: user's secret information

$(e, n)$: Public key components of RSA

$d$: Private key components of RSA

$g$: An integer and primitive element for prime factors of the private key of RSA

$f$: Hash function

$T$: Current time stamp

$R_i$: Random number generated by smart card

In the literature, several remote user authentication schemes for smart home based on the use of smart card have been investigated. Representative ones are described as follows.

In [39], Yang and Shieh proposed two smart card based password authentication methods, namely, time-stamp based and nonce-based. In their schemes, the host does not need to keep the password table as well as the verification table. For their timestamp-based protocol, in the registration phase, the following information $(ID_i, CID_i, h_i, S_i, e, g, n)$ is written into the card. Next, whenever the user wants to login, he calculates $X_i \equiv g^{r_i.pw_i}(mod\, n)$ and $Y_i \equiv S_i.h_i^{r_i.f(CID_i.T)}(mod\, n)$, and sends $(ID_i, CID_i, X_i, Y_i, e, g, n, T)$ to the host, which in turn verifies the following information: $ID_i, CID_i$and timestamp, then calculates $Y_i^e = ID_i.X_i^{f(CID_i.T)}$ . If the equation holds true, the user gets authenticated. Chan and Cheng [40] and Sun and Yeh [41] performed a cryptanalysis of this scheme and reported that it is vulnerable to forgery attack.

In [42], Fan et al. also performed a cryptanalysis of the Yang and Shieh scheme [39] and showed that it is vulnerable to impersonation attack. Based on their findings, they proposed an enhanced scheme which can be forged only with a valid $CID_i$, by imposing a restriction

on $ID_i$ so that an attacker cannot freely generate the $ID_i$.

In [43], Yang et al. also proposed an improvement of the timestamp-based authentication scheme proposed in [39]. In that scheme, the registration phase is not modified, but in the login phase, the user calculates $X_i = g^{pw_i.r_i}(mod\, n)$ and $Y_i = S_i.h_i^{r_i.T}(mod\, n)$ , and sends $(ID_i, CID_i, X_i, Y_i, e, g, n, T)$ to the host. In the authentication phase, the host verifies the following information: $ID_i, CID_i$ and timestamp, and calculates $Y_i^e = ID_i^{CID_i}.X_i^T mod\, n$ . If the equation holds true, the user gets authenticated. They also performed a security analysis of this scheme, and verified that it is protected against forgery attack, password-guessing attack, smart-card loss attack and replay attack.

In [44], Shen et al. proposed a modified version of the Yang and Shieh scheme [39] scheme, transforming it to a mutual authentication scheme. They performed an analysis of the scheme in [39] and discovered that no relationship has been established between $ID_i$ and $CID_i$,and without this, an intruder could bypass the remote server verification. In the modified scheme, a relationship between $ID_i$ and $CID_i$at the registration phase is established a as follows: $CID_i = f(ID_i \oplus d)$,so that when the server receives the login request, it checks the validity of $ID_i$ Then, it calculates $CID_i' = f(ID_i \oplus d)$ ,and if $CID_i'$ is equal to $CID_i$, the user gets authenticated.

In [45], Liu et al. proposed a remote user mutual authentication scheme for smart home as a result of a cryptanalysis of the Shen et al. scheme [44]. In their analysis, it was reported that if $ID_i$ is chosen by a legitimate user, an attacker may be able to impersonate that user. In the modified Shen et al. scheme [44], to address this deficiency, instead of sending the $CID_i$, the user sends $f(CID_i)$ to the server, and a random nonce is used to challenge the user for mutual authentication purpose.

In [46], Chien et al. also proposed a remote user mutual authentication scheme for smart home, for which the security is based on the use of one-way hash functions. In their scheme, the server maintains a secret value $x$, and a timestamp is generated and attached to all messages that are exchanged between the user and the server, in order to protect against replay

attack. However, a cryptanalysis of this scheme by Hsu [47] showed that it is vulnerable to parallel session attacks.

In [22], Om and Reddy proposed a RSA-based remote user authentication scheme using smart card. In their scheme, to achieve authentication, no verification table, nor password table is needed. Their scheme uses the standard RSA algorithm for cryptography as follows. In the registration phase, the user $U_i$ submits its $ID_i$ to the system, which in turn calculates $PW_i = ID_i^d mod\, \phi(n)$. It should be noted that a password generated in this way is difficult to remember for a user. Next, the smart card is issued with $(f, ID_i, e, n)$. To login, the user calculates $x = PW_i \oplus T$, $y = ID_i^x (mod\, n)$ and $C = y^e (mod\, n)$. Then, sends the following message $S = (x, ID_i, C, T)$ to the server. Upon receipt, the server validates $ID_i$ and the timestamp, then checks if $C^d (mod\, n) = y$. If that equation holds, the user gets authenticated. A cryptanalysis of this scheme by Om and Kumari [10] reveal that it is does not work properly for any given password.

In [10], Om and Kumara proposed a modified version of the Om and Reddy scheme [22] as follows. The registration phase is similar to that of the Om and Reddy scheme [22], except that additionally, the user calculates $S_i = f(PW_i||ID_i)$ and stores it in the smart card. To login, the user calculates $x = f(f(PW_i||ID_i) \oplus T) mod\, n$, $y = ID_i^x (mod\, n)$ and $C = y^e (mod\, n)$, then sends $(ID_i, C, T)$ to the server. Upon receipt, the server checks for $ID_i$ and the timestamp, then calculates $x = f(S_i \oplus T) mod\, n$, $y = ID_i^x mod\, n$ and $M = C^d mod\, n$. If $M = y$, the server authenticates the user. It should be noted that $S_i$ is a sensitive information which is related to the password $S_i = f(PW_i||ID_i)$; and $S_i$ is also calculated when the user enters the password. If the smart card is lost, this information may lead to a compromised security [48]. Om and Kumari [10] also proposed a user remote authentication scheme based on the RPrime algorithm, a variant of RSA which is considered fast than the standard RSA [49]. In this scheme, at the registration time, the user chooses $ID_i, PW_i$ and a random number $N_i$, then sends $ID_i, f(PW_i \oplus N_i)$ to the server. Next, the server calculates, $CID_i = f(ID_i \oplus d) mod\, \phi(n)$, $S_i = (CID_i \oplus f(PW_i \oplus N_i))^e mod\, n$ and $R_i = f(S_i||N_i)$, where

$S_i$ is the sensitive information referred to in the Om and Reddy scheme [22], then stores the data $(CID_i, R_i, n)$ in the smart card. This scheme was shown to be vulnerable to smart-card loss attack.

Unlike the above-described remote user authentication schemes, our proposed scheme is based on a lightweight version of a RSA variant [10]. the messages are kept encrypted and only one-time-token is used in clear-text. In addition, the anonymity of the device or user is achieved using a one-time-token, and a session key is established for any new session. Finally, our scheme is designed to protect against replay attacks by using a timestamp in each message exchange between the user and the server.

# Chapter 3

# Proposed Authentication Protocols for Smart Homes Network

In this Chapter, the designs and modelling (using the HPSL language [25]) of the two-factor D2D Mutual Authentication Scheme for Smart Homes using OTP over Infrared Channel (so-called D2DA-OTP-IC scheme), the Password Protected Inter-device OTP-based Authentication scheme over Infrared Channel (so-called PPIDA-IC) and the Password Protected Inter-device OTP-based Authentication scheme using Public Key Infrastructure (so-called PPIDA-PKI scheme) are presented, along with the design of the RSA-based two-factor user Authentication scheme for Smart Home using Smart Card (so-called RSA-ASH-SC scheme).

## 3.1 Design and Modelling of the D2DA-OTP-IC Scheme

### 3.1.1 High-Level Architecture of the D2DA-OTP-IC Scheme

The high-level architecture of the proposed scheme is depicted in Fig. 3.1, where three devices are considered: OTP-server, Device 1 (denoted D1), and Device 2 (denoted D2) (of course, more devices can be considered). Without loss of generality, the number of devices

has been restricted to three only for the hardware proof-of-concept purpose. In Fig. 3.1, D2 is initiating the authentication request with D1. After being authenticated using a public-key infrastructure, D1 sends a request for OTPs to the OTP-server. Upon receipt of this request, the OTP-server replies by sending the OTPs to both D1 and D2. It should be noted that the same high-level architecture prevails for the PPIDA-IC and PPIDA-PKI schemes since these have been proposed as enhancements of the D2DA-OTP-IC scheme.

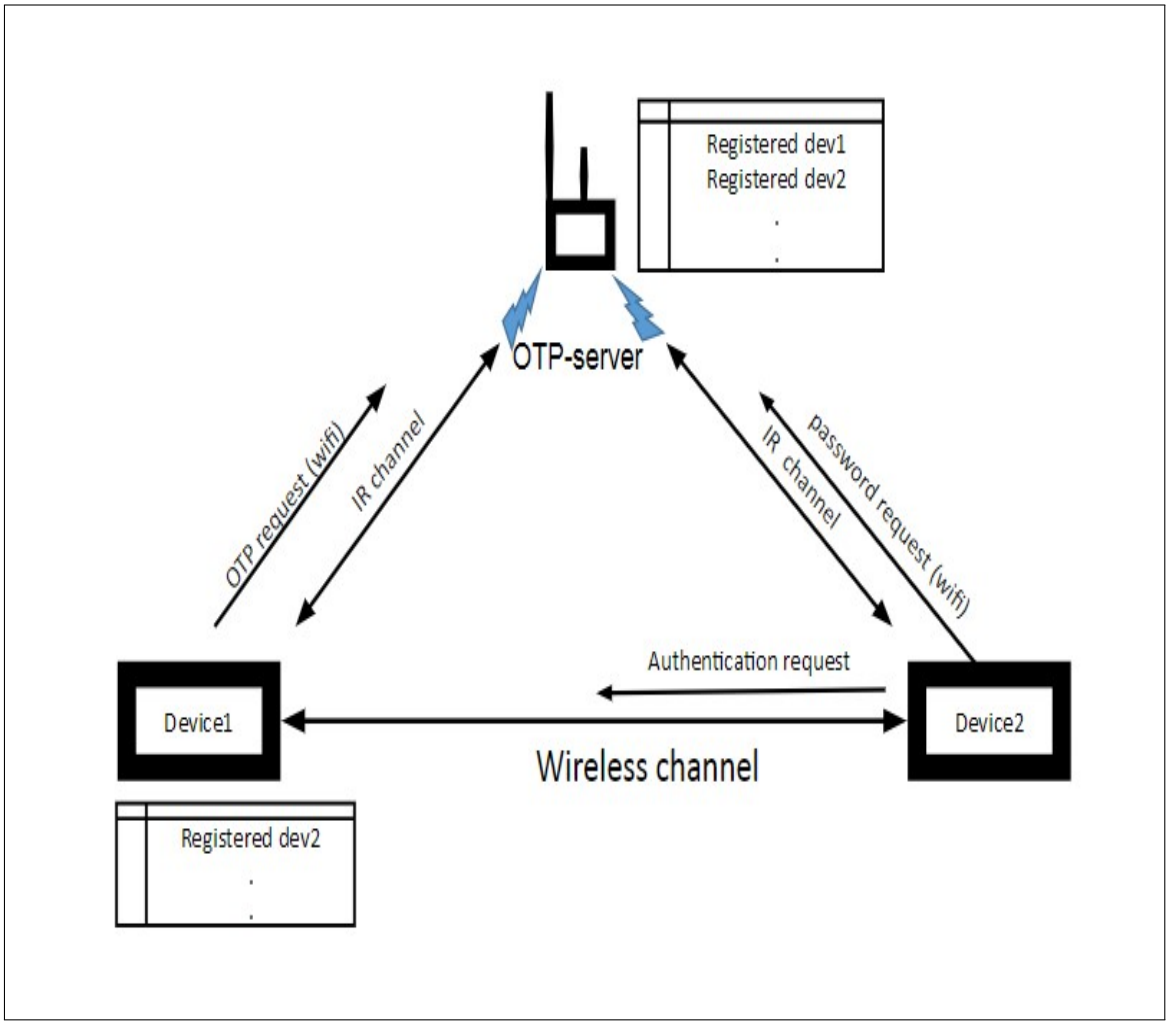


Figure 3.1: High-level architecture of the proposed authentication scheme.

In order to represent the data exchanged in the proposed protocols and the cryptographic processes that are involved, the abstract A-B notations style given in Table 3.1, which is

inherited from [25], will be used (here, A-B is meant for Alice-Bob').

Table 3.1: A-B Style Notations

| **Notation** | **Description** |
|---|---|
| D1 | Client device |
| D2 | Server device |
| OTP-server | OTP server that generates and sends the OTPs |
| T1, T2, T3, T4, T5 | Current timestamp |
| $K_{D1}, K_{D2}, K_O, K_{KPHo}, K_{KPHa}$ | Public keys of devices D1, D2 and O |
| $K'_{D1}, K'_{D2}, K'_O, K'_{KPHo}, K'_{KPHa}$ | Private keys of devices D1, D2 and O |
| P, P1, P2 | Prime number |
| G, G1, G2 | primitive root modulo of P |
| N1, N2, N3, N4 | Random secret numbers |
| SK, SK1, SK2 | Secret key generated using the Diffie-Hellman key exchange protocol [53] |
| $Hash()$ | Hash function |
| $A -> B : M$ | A send message M to B |
| $En\{\}_K$ | Encryption is performed using key K |
| EXP | Exponent function |
| REQ | OTP request number |
| MOD | Modulus operation |
| OTP1, OTP2 | One-Time-Password |
| Token | Random number to uniquely identify a service |
| Password | Password of a service |
| Timestamp | Time stamp |

As per the notations on Table 3.1, the working of the proposed D2DA-OTP-IC scheme (Fig. 3.1) consists of three phases: manufacturing (or pre-deployment) phase, deployment and registration phase, and authentication (or functioning) phase, described as follows.

**Manufacturing Phase:** In this phase, the manufacturer presets the device for father coming phases. First, the manufacturer builds the device along with the infrared light transceivers which will be used for infrared communication. In this process, each device gets a unique ID. Using those IDs, devices will communicate with each other. The manufacturer then uploads the software (i.e. operating system such as Linux or other hardening

tools such as Lynis, CIS-CAT and Tiger) to a device, including the drivers for infrared light communication. The software is then configured in such a way that it is compliant with the NIST/FIPS regulations and standards [8]. Even though our focus is not on the hardening of devices, it is usually a good practice to harden the system in advance in order to protect against zero-day attacks [50]. It is also assumed that the software is capable of updating itself frequently in a secure manner.

**Deployment and Registration Phase:** In this phase, all devices are registered with the OTP-server, as well as with peer devices with whom they need to authenticate in future. To register with the OTP-server, the public key of the device is stored with the OTP-server in a secure manner, along with the device ID. The location of the device with respect to the OTP-server is also stored in the form of degree of angle. It should be emphasized that the device has to face towards the OTP-server in order for the infrared light communication to happen. In addition, the device will save its IP address information and device ID of the OTP-server. For a device, say D1, to communicate with another device, say D2, device D1 will need to pre-register with device D2 the same way that it has registered with the OTP-server.

**Authentication Phase:** When a device, say D1 wants to communicate with another device, say D2, device D1 initiates the request for authentication. The goal is to generate a secret key which will be used for future communication. Here, the DH key exchange protocol [51] is used to generate the secret keys that will be kept secret between D1 and D2. For every new session, a different secret key will be generated for the communication. The authentication steps are then performed as follows.

**Step 1:** First, D1 generates a random number N1 and calculates EXP(G1, N1), where G is a chosen positive integer and EXP denotes the exponentfunction. The output of this operation is used to find the modulus over P using the MOD (modulus) function, where G and P are publically known. D1 then sends the following message to D2, signed with its private key:

$D1 -> D2$ :

$D1.\{D1.T1.G.P.MOD(EXP(G, N1), P).Hash(D1.T1.G.P.MOD(EXP(G, N1), P))\}_{K'_{D1}}$

**Step 2:** D2 first checks if the message is originated from D1using D1's public key, then it checks whether the timestamp is recent or not. If both checks are successful, D2 continues with the request and saves MOD(EXP(G, N1), P) into a variable (say AA); otherwise the request is dropped. Next, D2 calculates EXP(G,N2) and the output of this is used to calculate the modulus over P. Finally, D2 sends the following message to D1, signed with its private key.

$D2 -> D1$ :

$D2.\{D2.T2.MOD(EXP(G, N2), P).Hash(D2.T2.MOD(EXP(G, N2), P))\}_{K'_{D2}}$

Then, it derives the secret key SK by calculating MOD(EXP(AA, N2),P). When D1 receives the message, it first checks the authenticity of the message using D2's public key. Then, it checks whether the timestamp is recent or not. If both checks are successful, D1 continues further with that request and saves MOD(EXP(G, N2), P) into a variable (say BB); otherwise it understands that something has gone wrong or it drops the message. Next, it gets the secret key SK by calculating MOD(EXP(BB, N1), P). At this point, both devices D1 and D2 have successfully shared the secret key SK using the DH key exchange algorithm [51].

**Step 3:** Next, D2 requests the OTP-server to provide the OTPs for both devices D1 and D2. To issue such request, D2 generates a timestamp T3 and sends the following message to OTP-server, signed with its private key.

$D2 -> OTP - server : D2.\{D1.D2.T3.REQ.Hash(D1.D2.T3)\}_{K'_{D2}}$

**Step 4:** Upon receipt of the request, the OTP-server checks if the message is really originated from D2 and it also checks if the timestamp is recent. If both checks are successful, the OTP-server continues with the request and generates two OTPs: OTP1 and OTP2, along with a timestamp T4; otherwise, it drops the request. Next, it sends the following message to D1, encrypted with D1's public key:

OTP-server $-> D1 : \{T4.OTP2.OTP1\}_{K_{D1}}$

**Step 5:** In a similar manner than in Step 4, the OTP-server sends the following message to D2, encrypted with D2's public key:

OTP-server $-> D2 : \{T4.OTP1.OTP2\}_{K_{D2}}$

It should be noted that the OTPs are sent over the infrared communication channel within the home (which is considered as a secure channel); therefore, the encryption of the above message can be optional. At this point, both devices D1 and D2 have received their OTPs.

**Step 6:** D1 sends the hash of OTP2 to D2, encrypted with the secret key SK, i.e.

$D1 -> D2 : \{Hash(OTP2)\}_{SK}$

**Step 7:** Similarly, D2 sends the hash of OTP1 to D1, i.e.

$D2 -> D1 : \{Hash(OTP1)\}_{SK}$

Finally, D1 (resp. D2) checks the received hash Hash(OTP1) (resp. Hash(OTP2)) using M=Hash(OTP1) (resp. M=Hash(OTP2)), where M is the received message after decryption; and if there is a match, D1 and D2 have mutually authenticated each other; otherwise the authentication has failed.

### 3.1.2 3.1.2 Modelling of the D2DA-OTP-IC Scheme

#### 3.1.2.1 Modelling Tool

The SPAN/AVISPA tool [25] is used for the formal evaluation and simulation of the D2DA-OTP-IC protocol. Its high-level design [25] is illustrated in Fig. 3.2.

The use of the SPAN/AVISPA tool requires that the protocols be modelled using HLPSL [25]. This tool considers the protocol written in HLPSL format, then uses its integrated HLPSL2IF translator to convert it into an Intermediate format (IF), which is then passed to a specific backend tool composed of four modules, namely the On the fly Model Checker OFMC), the Constraint Logic based Attack Searcher (CLAtSe), the SAT based Model Checker (SATMC), and the Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP), which are responsible for the analysis of the

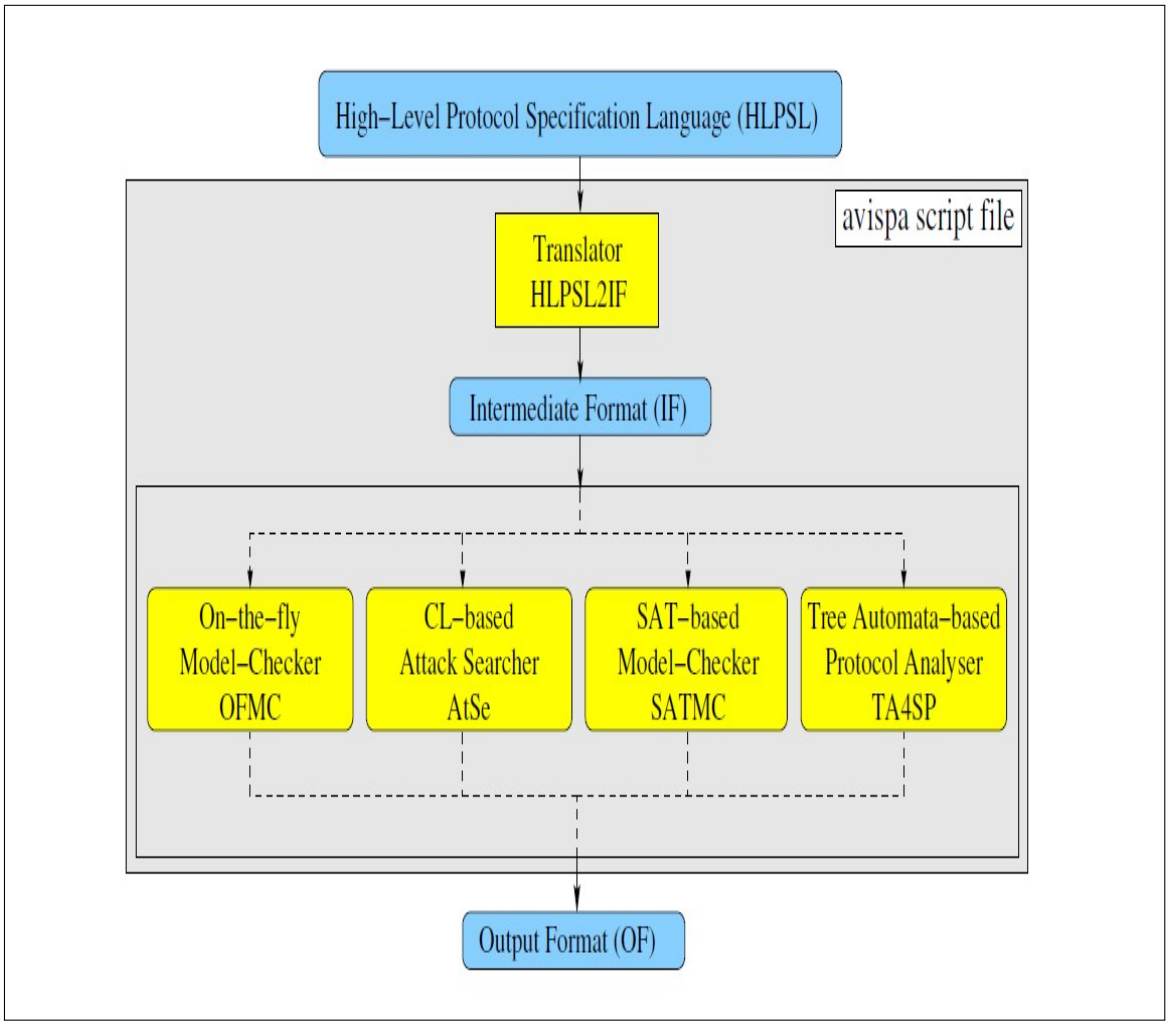


Figure 3.2: High Level Design of AVISPA Tool [25].

protocol as well as the verification of its effectiveness in terms of various different predefined attack types.

This backend tool also checks whether the sessions are bounded or unbounded, then present the results in an output format (OF). The OFMC module is meant for protocol falsification and bounded session verification considering both typed and untyped protocol models as well as integrated symbolic constraint-based techniques which can be used to ensure that no attacks are lost. OFMC also supports the intruder's implementation aiming at guessing weak passwords; in addition, it provides the specification of the algebraic properties of cryptographic operators. When the OFMC model specifies that the protocol is safe, it

means that all the specified goals have been met. On the other hand, the CL-AtSe module also performs the protocol falsification and verification by translating the protocol specification into a set of constraints used to identify the attacks. These back-end modules can be used for analyzing the security threats in a protocol using the implemented Dolev Yao (DY) network attack model itself considered as the default attack model in which an attacker has a full control over the network, thus can perform various different types of hacking tricks to compromise the protocol. When the SPAN/AVISPA tool [25] is run, its generated output is interpreted as safe or unsafe against the DY attacker model.

#### 3.1.2.2 HSPSL Syntax

The abstract notations used to convert the proposed protocols into the HSPL code to be considered as input (as per Fig. 3.2) to the AVISPA tool [25], are given in Table 3.2.

In HLPSL, variables start with capital letters and constants are represented with lower case letters. HLPSL is a role-based language, where all entities involved in a communication such as client, environment, server, session, etc., have roles associated with their message exchange. As an example, security-related properties of a protocol can be defined using HLPSL [25]. In doing so, there are two types of roles: basic roles and composed ones, depending on whether they are defined to represent the actions of one or more agents. To achieve message exchanging between two entities, a session is created between the roles and the environment for all sessions.

**Basic Role:** Considering the Alice-Bob (A-B) notation (given in Table 3.1). A sends an encrypted message M to B, using key K to encrypt the message is represented as:

$A \rightarrow B: \{M\}\_K$

An example of the syntax of a basic role declared using HLPSL is:

*role alice(A,B,S: agent,*

Table 3.2: HLPSL Notations

| **Notation** | **Description** |
|---|---|
| . | associative concatenation (of messages) |
| := | initialization (of local variable) in init-section OR assignment to a local variable |
| , | prime, used for referring to the next (new) value of a variable in a transition |
| /\ | conjunction (logical AND) |
| =\|> | immediate transition |
| { } | set delimiter, e.g. in knowledge declaration |
| {}_ | encryption or signature |
| agent | data-type for agents |
| channel (dy) | data-type for channels. Currently only the Dolev-Yao channel is implemented. |
| def= | indicates the beginning of the body of a role |
| hash_func | data-type for one-way functions |
| exp | exponentiation operator (prefix) |
| init | indicates the initialization of local variables |
| inv | inverse of a key: given a public key returns the private key |
| Intruder knowledge | defines knowledge of the intruder |
| played_by | for basic roles: specifies which agent is playing this role |
| request | used to check strong authentication (together with a witness) |
| witness | used to check authentication (together with the request) |
| secret | used to check secrecy |

*Kas: symmetric_key,*

*SND, RCV : channel (dy))*

*played_by A def=*

*local State: nat, Kab: symmetric_key*

*init State:= 0*

*Transition*

*...*

*end role*

where Alice is a role called with agents A, B, S, symmetric key Kas (resp. Kab) shared between A and S (resp. A and B) and SND, RCV as Dolev-Yao (dy) channels [10].

**Transition:** The behavior of a system in HLPSL is modelled as a state. Each state has variables that are responsible for state transitions. A transition is defined by a trigger event or precondition, along with the associated action that should be performed when the trigger event occurs [10]. An example is as follows:

*State = 0 ∧ RCV({Kab'}_Kas) =|>State':= 2 ∧ SND({Kab'}_Kbs)*

Where step1 is the transition's name. The above HLPSL statements specify that if the value of the state is equal to 0 and a message, which contains some value Kab' encrypted with Kas is received on the RVC channel, then a transition will occur, yielding a new state value (here 2) and that same value of Kab' encrypted (this time) using Kbs will be sent on a SND channel.

**Composed role:** This is a composition of several basic roles in sessions where the knowledge shared between these roles are explicitly known. An example of such role made of three different basic roles [10], where the symbol / indicates that the constituent roles are executed in parallel, is given as:

*role session(A,B,S : agent,*

*Kas, Kbs: symmetric_key)*

*def=*

*local SA, RA, SB, RB SS, RS: channel (dy)*

*Composition*

*alice (A, B, S, Kas, SA, RA)*

*∧ bob (B, A, S, Kbs, SB, RB)*

*∧ server(S, A, B, Kas, Kbs, SS, RS)*

*end role*

**Environment:** The environment used for protocol execution is defined, where i denotes the intruder and 'kis' denotes the intruder's private key. In the environment role, top level global constants are defined. In this top-level role, the intruder's knowledge parameter defining the intruder's initial knowledge is specified. An example of environment role [10] is as follows:

*role environment()*

*def=*

*const a, b, s : agent,*

*kas, kbs, kis : symmetric_key*

*intruder_knowledge* = {*a, b, s, kis*}

*Composition*

*session(a,b,s,kas,kbs)*

$\wedge$*session(a,i,s,kas,kis)*

$\wedge$*session(i,b,s,kis,kbs)*

*end role*

#### 3.1.2.3 Modelling of the D2DA-OTP-IC Scheme Using HSPSL

Following the above HLPSL syntax specifications [25], our proposed D2DA-OTP-IC scheme can be modelled as follows:

First, initially, when a device (client) wants to start communication with another device (server), it sends the request along with all parameters required for DH key exchange. This request is modelled in HLPSL as follows:

*State=0* $\wedge$*RCV(start)* =|>

*State'*:=*1* $\wedge$*Na'*:= *new()*

$\wedge$ *Timestamp'*:= *new()*

$\wedge AA' := MOD(EXP(G,Na'),P)$

$\wedge SND(A.\{A.Timestamp'.AA'.G.P.Hash(AA'.G.P.Timestamp'.A)\}\_inv(KPa))$

where the state variable 'State' is initialized to 0 in the init section.

Second, the client receives a start message via the RCV channel to start the protocol. It first updates the state variable's new value to 1, then it generates a random number, which in turn is used to generate the DH key exchange. It also generates the current timestamp (assuming that all the system's time is in sync). Then, it calculates MOD(EXP(G,Na')), where P and G are known variables, and then it stores the output in a variable, say AA', which is known to the intruder. It then calculates the hash of the message. Next, it sends the message by signing it with its private key. Upon receipt of this message, the server checks whether the timestamp is valid or not, and if that is validated, the server sends back its calculated value for the DH key exchange. This is modelled in HLPSL as follows:

$State=0 \wedge RCV(A.\{A.Timestamp'.AA'.G.P.Hash(AA'.G.P.Timestamp'.A)\}\_inv(KPa))$
$=|>$

$State':=1 \wedge Nb':= new()$

$\wedge BB' := mod(exp(G,Nb'),P)$

$\wedge SND(B.\{B.Timestamp'.BB'.Hash(BB'.Timestamp'.B)\}\_inv(KPb))$

$\wedge Key' := mod(exp(AA',Nb'),P)$

$\wedge secret(Key',secret\_key,\ A,B)$

$\wedge Req':=new()$

$\wedge SND(B.\ \{A.B.Timestamp'.Req'.Hash(A.B.Timestamp')\}\_inv(KPb))$

It should be noted that in this specification, the state variables are local to both the client and the server.

Third, the server generates its own private random number Nb' for the DH key exchange, then in a similar way as above, it calculates BB' and sends the message to the client by sign-

ing it with its private key. At this point, the server can form the secret session key Key'. Here, the secret() function is used to check the secrecy of the session key. The server then requests the OTP-server to provide the OTPs to both the client and itself. At the same time, the client receives the message from the server. It then attempts to validate the timestamp and authenticity of the message. If the validation is successful, the client will acquire the BB' value from the server, and form a session as shown below:

*State=1 ∧RCV(B.{B.Timestamp.BB'.Hash(BB'.Timestamp'.B)}_inv(KPb)) =|>*

*State':=2 ∧Key' := mod(exp(BB',Na'),P)*

*∧secret(Key',secret_key, {A,B})*

Fourth, the OTP server attempts to validate the timestamp and the authenticity of the request. If that is successful, it sends the OTPs to both the client and the server by encrypting them with their respective public keys as shown below:

*State=0 ∧RCV(B.{A.B.Timestamp'.Req'.Hash(A.B.Timestamp')}_inv(KPb)) =|>*

*State':=1 ∧Timestamp' := new()*

*∧SND({Timestamp'.OTP1.OTP2}_KPb)*

*∧SND({Timestamp'.OTP2.OTP1}_KPa)*

*∧secret(OTP1, server_otp1, {A,B,O})*

*∧secret(OTP2, server_otp2, {A,B,O})*

It should be noted that the OTPs are sent on the infrared light communication channel within the house, which is considered secure since the light cannot pass through objects such as walls. In real situations, the OTPs can be sent unencrypted because it is assumed that the intruder cannot get hold on the light communication channel. However, the OTPs have been kept encrypted. Also, it should be noted that the secrecy of the OTPs is ensured by using the secret() function.

Fifth, when both the client and the server have received their OTPs from the OTP-

server, they decrypt the message and validate the timestamp. Next, the client sends OTP2, encrypted with the session key and the server sends OTP1, encrypted with the session key as well. Both the client and the server mutually authenticate each other if the encrypted hash of the received OTP matches with the hash of the OTP which was not sent by both entities. In the client side, to check for strong authentication, which is achieved by means of the request() and witness() functions, the following HLPSL code is used:

*State= 3 /\RCV( {Hash(OTP1')}_Key) =|>*

*State':= 4 /\witness(A,B,server_client_bb , {B.Timestamp'.BB'}_inv(KPb)*

*.Hash(OTP1'))*

*/\request(A,B,client_server_aa ,{A.Timestamp'.AA'}_inv(KPa).Hash(OTP2'))*

Now, at the server side, the server witnesses the message and the OTP hash it has received from the client and request for its message and hashed OTP using the following HLPSL code.

*State= 2 /\RCV({Hash(OTP2)}_Key') =|>*

*State':=3 /\SND({Hash(OTP1)}_Key')*

*/\witness(B,A,client_server_aa,{A.Timestamp'.AA'}_inv(KPa).Hash(OTP2'))*

*/\request(B,A,server_client_bb,{B.Timestamp'.BB'}_inv(KPb).Hash(OTP1'))*

Finally, the goals of keeping OTP1 and OTP2 secret; ensuring the secrecy of the session key; and achieving mutual authentication between the client and server, is coded as follows using HLPSL:

*goal*

*secrecy_of server_otp1*

*secrecy_of server_otp2*

*secrecy_of secret_key*

*authentication_on client_server_key*

*authentication_on server_client_key*

*end goal*

## 3.2 Enhancements of the D2DA-OTP-IC Scheme

In the above proposed D2DA-OTP-IC Scheme [7], storing the secret information may not be considered safe as per the recommendations from [8]. Thus, it is essential that the proposed D2DA-OTP-IC design be adjusted to protect the secret and public keys. One way to achieve this goal is to store the password (or passphrase) used to protect the encrypted key in the system itself in order to comply with the Federal Information Processing Standard (FIPS) 4.7.5 [8], which states that "*Plaintext secret and private keys shall not be accessible from outside the cryptographic module to unauthorized operators. A cryptographic module shall associate a cryptographic key (secret, private, or public) stored within the module with the correct entity (e.g. person, group, or process) to which the key is assigned*". Therefore, in a D2D scenario, a solution to encrypt the secret information and provide the password on demand to the registered applications is required. In this regard, two proxy-password protected OTP-based schemes for smart homes, designed to achieve this requirement, are proposed as enhancements to the D2DA-OTP-IC scheme (so-called PPIDA-IC and PPIDA-PKI schemes).

In both schemes, the keys are protected using passwords and a centralized server provides proxy-password services to the smart home devices. In addition, this server maintains the database of passwords as well as the servers' password-proxy services.

A Linux security module (so-called Secure-Enhanced Linux (SELinux [52]) is also utilized, which provides a mandatory access control (MAC) mechanism implemented at the kernel level. Without this feature, a Linux system can only enforce the Discretionary Access

Control (DAC) and the Access Control Lists (ACLs) methods, and the user and programs can gain access to the files and other resources which are not required for their normal operations. SELinux works on the least-privilege model and it has three modes of operations: Enforcing, Permissive and Disabled [53]. In the enforcing mode, it enforces the policies and only enables the use of operations that are allowed in the policy while blocking the remaining operations.

Using SELinux, the process and resources are confined by means of labels [53]; the superuser (root user) privileges can be restricted or disabled [54], [55], [56]. As reported in [57] [58], the *sudoers* should not have access to change the password for the root and to shutdown, reboot, mount, unmount or install any new software. The only access given to *sudo* users is to execute our developed infrared program, which reads the password over the infrared channel and load the private key. The above-mentioned proof-of-concept of D2D-OTP-IC scheme [7] has been implemented in such a way that the infrared channel can be read using the Linux Infrared Remote Control (LIRC) program [65], but there was no provision of a mechanism to check which user is calling or whether an operation is valid or not. In our proposed PPIDA-IC scheme, the developed infrared program is meant to achieve these operations. Typically, this program is used for accessing the infrared devices in the smart home. It should be noted that usually a non-super user program does not have access to the infrared device, unless a *sudo* command is utilized.

Finally, it should be noted that the HLPSL language [25] is also used to model the proposed PPIDA-IC and PPIDA-PKI schemes; and a security analysis is conducted using the SPAN/AVISPA tool [25] to demonstrate that these schemes can also achieve the goals of secrecy of the secret keys and mutual authentication (please refer to Chapter 4 for the details of the evaluation).

### 3.2.1 Assumptions

For the design of both the PPIDA-IC and PPIDA-PKI schemes, the following assumptions prevail:

- The applications which need authentication runs on a non-root (non-administrative) account so that even if a non-root application is compromised (for instance, due to a vulnerability), the impact of this attack would be less severe than that of an attack in a root account.
- The OTP-server encrypts its database (which contains all the information on the proxy services) with a secret key (for instance, using the Advanced Encryption Standard (AES)), and this secret key is protected using the database's private key.
- This private key is further protected by the SELinux [52] in such a way that only the OTP-server process can access it. Alternatively, this private key can also be protected by a passphrase. In that case, the user will have to provide the passphrase until the OTP-server reboots.
- The infrared channel is considered secure and the infrared light cannot spread outside a home.

### 3.2.2 Design of the PPIDA-IC Scheme

In the PPIDA-IC scheme, the infrared communication channel is used to send the password to the requesting device. The high-level architecture depicted in Fig. 3.1 also supports this scheme. Here, device D2 wants to get authenticated to device D1 with the help of the OTP-server. In addition, two kinds of services are involved, namely password-proxy service for D2's private key and distribution of the OTPs to both D1 and D2 for mutual authentication purpose.

The working of the PPIDA-IC scheme consists of three phases as follows:

**(1) Manufacturing phase:** In this phase, the manufacturer presets the device and equips

it with the infrared light transceivers and drivers for infrared light communication. This software package is configured in compliance with the FIPS NIST standards [8] and the system is hardened for protection against zero-day attacks [50] using tools such as Tiger, Lynis, etc.

**(2) Deployment and registration phase:** All the devices need to be registered in advance with OTP-server. There are two types of registration: OTP-server registration and peer-device registration. If a device, say D1, needs to authenticate to its peer device, say D2, D1 should also register with D2. To register a device D1 with OTP-server, the public key of the device is stored in the OTP-server in a secure manner, along with the device ID. The location of the device with respect to the OTP-server is also stored in the form of degree of angle (it is assumed that it is pre-registered). It should be emphasized that the devices should face towards the OTP-server in order for the infrared light communication to occur. In addition, home devices must save the OTP-server s public-key. Each home-device can run more than one services. All the services should be registered in advance. For each password-proxy service, an associated random token is generated by the OTP-server, which in turn is saved in the OTP-server's database and the home-device. This token is also linked with the password of that password-proxy service (as a way to recognize the requested service in future). For peer-peer home-device registration, the public keys are assumed to be shared among these devices in a secure manner.

**(3) Authentication phase:** This phase comprises the following steps (as shown in Fig. 3.3).

**Step 1:** When a device, say D1, needs to get authenticated, it first requires the use of a password for its private-key. It sends a message to the OTP-server, encrypted with the OTP-server s public key. This message includes a unique token, which is to be used for service identification. To protect against replay attack, a timestamp is appended to the token; and for integrity check, the message is hashed. These operations translate into the following statement:

$$D1 -> O : D1.\{Token.Timestamp.Hash(Token, Timestamp)\}_{K_o}$$

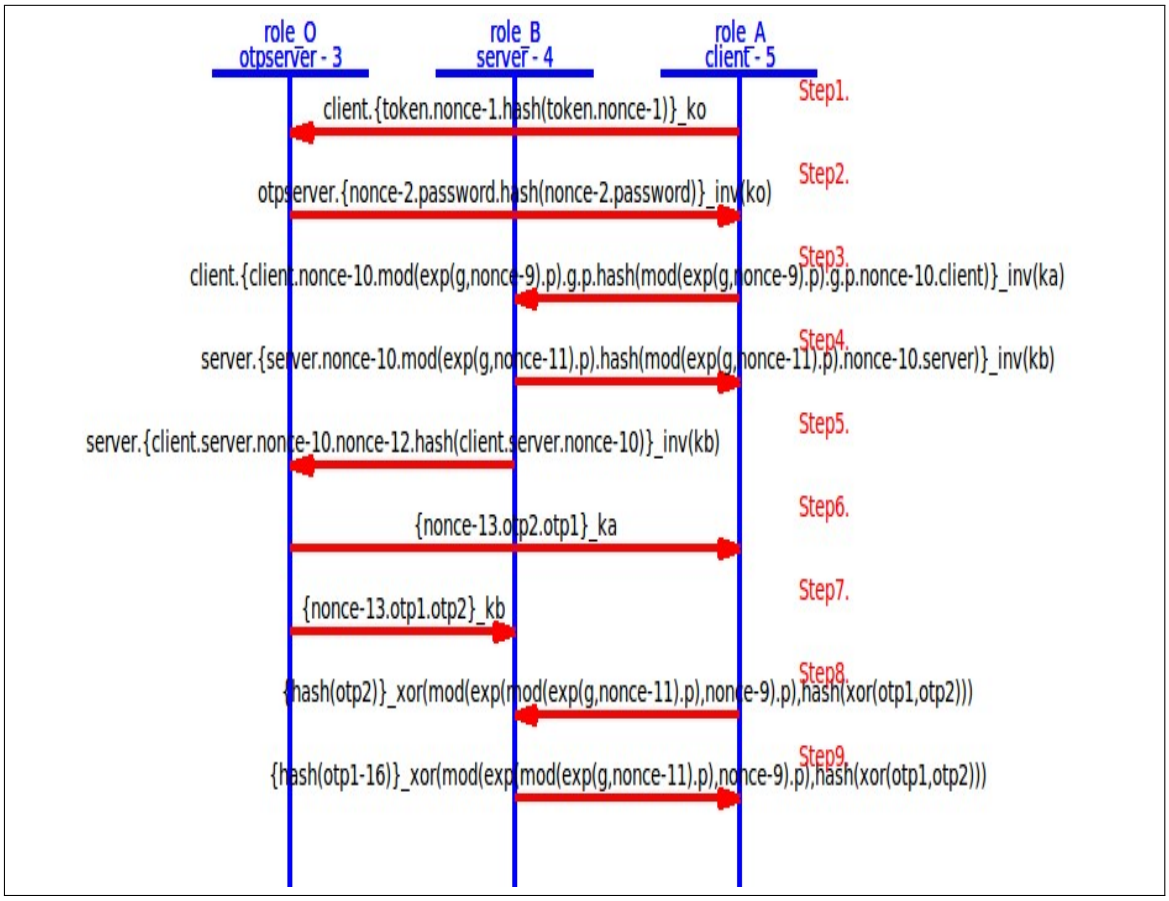


Figure 3.3: PPIDA-IC scheme authentication phase.

**Step 2:** Once the OTP-server has received a request, it validates the timestamp and token. If this validation is successful, it sends the password over the infrared channel to the requesting device.

$$O->D1 : O.\{Timestamp.Password.Hash(Timestamp.Password)\}_{K'_O}$$

where O denotes the OTP-server. Once the password is received, it is used to fetch the encrypted private key.

**Step 3:** D1 generates a random number N1 and calculates EXP(G,N1), where G is a chosen positive integer and EXP is the exponent function. The output of this operation is used to find the modulus over P using the MOD (modulus) function, where P is publicly known. D1 then sends the following message to D2, signed by its private key.

$$D1->D2 :$$

$D1.\{D1.T1.G.P.MOD(EXP(G,N1),P).Hash(D1.T1.G.P.MOD(EXP(G,N1),P))\}_{K'_{D1}}$

**Step 4:** D2 checks if the message is originated from D1 using D1's public key, then it checks whether the timestamp is recent or not. If these checks are successful, D2 continues with the request and saves MOD (EXP(G, N1),P) into a variable (say AA); otherwise the request is dropped. Next, D2 calculates EXP(G, N2) and the output of this is used to calculate the modulus over P. Finally, D2 sends the following message to D1, signed with its private key.

$D2 -> D1$ :

$D2\{D2.T2.MOD(EXP(G,N2),P).Hash(D2.T2.MOD(EXP(G,N2),P))\}_{K'_{D2}}$

Then, it derives the secret key SK by calculating MOD(EXP(AA, N2), P). When D1 receives the message, it first check the authenticity of the message using D2's public key. Then, it checks whether the timestamp is recent or not. If both checks are successful, D1 continues further with that request and saves MOD(EXP(G,N2), P) into a variable (say BB); otherwise it understands that something has gone wrong or it drops the message. Next, it gets the secret key SK by calculating MOD(EXP(BB, N1), P). At this point, both devices have successfully shared the secret key SK using the DH key exchange algorithm [51].

**Step 5:** Next, D2 requests the OTP-server to provide the OTPs for devices D1 and D2. To issue such request, D2 generates a timestamp T3 and sends the following message to OTP-server, signed with its private key.

$D2 ->$OTP-server: $D2.\{D1.D2.T3.REQ.Hash(D1.D2.T3)\}_{K'_{D2}}$

**Step 6:** Upon receipt of the request, the OTP-server checks if the message is really originated from D2 and whether the timestamp appended to it is recent or not. If both checks are successful, the OTP-server continues with the request and generates two OTPs: OTP1 and OTP2, along with a timestamp T4; otherwise, it drops the request. Next, it sends the following message to D1, encrypted with D1's public key.

OTP-server$->D1 : \{T4.OTP2.OTP1\}_{K_{D1}}$

**Step 7:** In a similar manner as in Step 6, the OTP-server sends the following message to D2, encrypted with D2's public key.

OTP-server$->D2 : \{T4.OTP1.OTP2\}_{K_{D2}}$

It should be noted that the OTPs are sent over the infrared communication channel within the home (which is considered as a secure channel); therefore, the encryption of the above message can be optional. At this point, both devices D1 and D2 have received their OTPs. For the OTP-based mutual authentication of D1 and D2 (i.e. second authentication level), the above secret key SK is modified as follows, producing SK'

$SK' := XOR(SK, Hash(XOR(OTP1, OTP2)))$

where XOR and Hash denote the XOR operation and a one-way hash function respectively.

**Step 8:** D1 sends the hash of OTP2 to D2, encrypted with the secret key SK', i.e.

$D1 -> D2 : \{Hash(OTP2)\}_{SK'}$

**Step 9:** Similarly, D2 sends the hash of OTP1 to D1, i.e.

$D2 -> D1 : \{Hash(OTP1)\}_{SK'}$

Finally, D1 (resp. D2) checks the received hash Hash(OTP1) (resp. Hash(OTP2)) using M=Hash(OTP1) (resp. M=Hash(OTP2)), where M is the received message after decryption; and if there is a match, D1 and D2 have mutually authenticated each other; otherwise the authentication has failed.

It should be emphasized that in the above process, the infrared channel program utilized to fetch the password using the infrared channel requires the *sudo/root* access. Thus, an access control program such as the one proposed in [28] can be used to enable the infrared device to read the infrared channel and to allow any user to execute it in order to receive the password. It is the task of the infrared program to check if the requesting user is the owner of the service. This program is meant to read the password and load the private key into a SSH-agent [29], which in turn can enable that user to access it. This way, the private key is kept secure and a non-root user cannot get hold of the password. It should be noted that once the private key is loaded, it can be used by the same user multiple times, until it is revoked.

### 3.2.3 Design of the PPIDA-PKI Scheme

In the PPIDA-PKI scheme, a PKI is used to send the password proxy services to the requesting device. Its high-level architecture is similar to that of PPIDA-IC scheme depicted in Fig. 3.1, but with some differences in its functional requirements (as will be described in the following).

In this scheme, two levels of PKI key-pairs are used, namely, host level authentication and user level authentication (where the application runs), each using a private-public key pair. At the host level authentication phase, the private key is protected by the root user. Afterwards, the root user is disabled if attempted to login (as per the SELinux requirements [52]). In our scheme, SELinux is used to enforce the access policy in such a way that no one can access the private keys of the host, except the cryptographic module. It should be emphasized that all *sudoers* accounts should only have access to the infrared program which in turn reads the infrared channel for OTP exchange purpose. In the user level authentication, the user's private key is encrypted with the password or passphrase, then it is used for authentication purpose.

The working of the PPIDA-PKI scheme consist of three phases as follows:

**(1) Manufacturing phase:** This is similar to the manufacturing phase of the PPIDA-IC scheme.

**(2) Deployment and registration phase:** All the devices need to be registered in advance with the OTP-server. There are two types of registration: home-device registration with OTP-server and peer-to-peer home-device registration. If a device, say D1, needs to authenticate to its peer device, say D2, D1 should also register with D2. To register a device D1 with the OTP-server, a host level public key-pair (kpha) is created, then the associated public key is stored in the OTP-server in a secured manner. It should be noted that Kpha is owned by the root (superuser) on D1. Similarly, the OTP-server creates a host level public key-pair (kpho) in a one-time operation, and the associated public key is also stored with D1 securely.

A user level public key pair (KD1/ka) of D1 is also created and its associated public key is shared with the OTP-server. Next, the OTP-server's public key (ko) is stored with D1; this key is used to receive the password-proxy requests. In addition, the location of the device with respect to the OTP-server is also stored in the form of degree of angle. It is worth pointing out that the devices should be facing towards the OTP-server in order for the infrared light communication to occur.

In addition, the home devices must save the OTP-server s public-key. For each password-proxy service, a random token is generated by the OTP-server, then saved in the OTP-server and the device. This token is also linked with the password of that password-proxy service (as a way to recognize the requested service in future).

For peer-device registration, if a device, say D1, wants to communicate with another device, say D2, then D1 has to pre-register with D2. In peer-to-peer home device registration, the public keys KD1/ka and KD2/kb are shared in advance in (a secure manner) among the devices.

**(3) Authentication phase:** This phase comprises the following steps (shown in Fig. 3.4).

**Step 1:** When a device, say D1 wants to communicate with another device, say D2, it initiates the request for authentication. In the PKI-based host authentication level, the DH key exchange algorithm [27] is used to generate the secret session key that will be kept secret between D1 and D2. For every new session, a different secret key will be generated and used for encrypting all communication between both hosts. Next, D1 sends the following message to initiate the authentication by encrypting the service token, the current timestamps, and their hash using with OTP-server's public key, i.e.

$$D1 -> O :$$

$$D1.\{Token.Timestamp.Hash(Token.Timestamp)\}_{K_O}$$

where O denotes the OTP-server.

**Step 2:** Once the OTP-server has received that request, it generates a random number

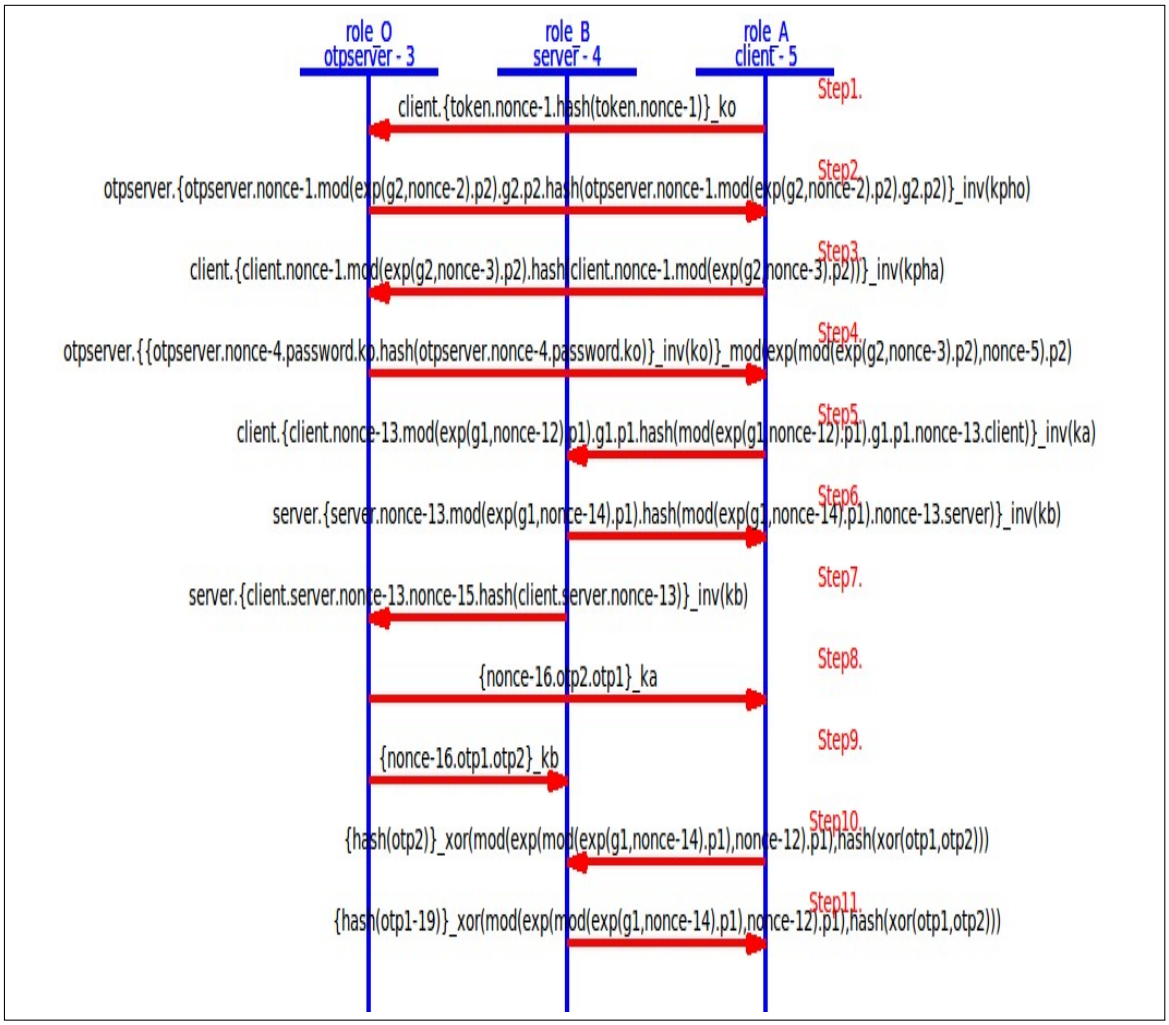


Figure 3.4: PPIDA-PKI scheme authentication phase.

N1 and calculates EXP(G2, N1), where G2 is a chosen positive integer. The output of this operation is used to find the modulus over P2 using the MOD (modulus) function, where P1 is publicly known. O then sends the following message to device D1, signed by its private key PKHo:

$$O->D1:$$

$$O.\{O.T1.G2.P2.MOD(EXP(G2,N1),P2).Hash(O.T1.G2.P2 .MOD(EXP(G2,N1),P2))\}_{K'_{PKo}}$$

**Step 3:** D1 checks if the message is originated from O using O's public key, then it checks whether the timestamp is recent or not. If both checks are successful, D1 continues with

that request; otherwise the request is dropped. If these checks are successful, D1 continues with the request and saves MOD(EXP(G2, N2),P2) into a variable (say AA); otherwise the request is dropped. Next, D1 calculates EXP(G2, N2); and the output of this is used to calculate the modulus over P2. Finally, D1 sends the following message to O signed by its private key PKHa:

$$D1 -> O :$$

$$D1\{D1.T2.MOD(EXP(G2, N2), P2).Hash(D1.T2 .MOD(EXP(G2, N2), P2))\}_{K'_{PKa}}$$

After sending the message to O, D1 obtains the secret key SK1 by calculating MOD(EXP(AA, N2), P2). Once O receives the message, it first checks the authenticity of the message using D1's public key. Then, it checks whether the timestamp is recent or not. If these checks are successful, O continues with the request and saves MOD(EXP(G2, N2),P2) into a variable (say BB); otherwise it understands that something has gone wrong or it drops the message. Next, it gets the secret key SK1 by calculating MOD(EXP(BB, N1), P2). At this point, both devices have successfully shared the secret key SK1 using the DH key exchange algorithm [51].

**Step 4:** Now, the OTP-server looks into database entries for the token and retrieve the password. This password is encrypted with O's private key, then further encrypted with SK1, i.e.

$$O -> D1 : \{O.T3.Password.K_O.Hash(O.T3.Password.K_O)\}_{SK1}$$

Next, D1 validates the message and retrieves the password used to decrypt the private key file $K'_{D1}$

**Step 5 to Step 9:** These steps are similar Step 3 to Step 7 of the PPIDA-IC scheme authentication phase.

**Step 10:** For the OTP-based mutual authentication of D1 and D2 (i.e. second authentication level), the generated secret key SK (different from the above private key SK1) is modified as follows, producing SK'

$SK' := XOR(SK, Hash(XOR(OTP1, OTP2)))$

**Step 11:** Next, D1 sends the hash of OTP2 to D2, encrypted with the secret key SK', i.e.

$D1 -> D2 : \{Hash(OTP2)\}_{SK'}$

Similarly, D2 sends the hash of OTP1 to D1, i.e.

$D2 -> D1 : \{Hash(OTP1)\}_{SK'}$

Finally, D1 (resp. D2) checks the received hash, i.e. Hash(OTP1) (resp. Hash(OTP2)) using M=Hash(OTP1) (resp. M=Hash(OTP2)), where M is the received decrypted message; and if there is a match, D1 and D2 have mutually authenticated each other; otherwise, the authentication has failed.

### 3.2.4 Modelling of the PPIDA-IC Scheme Using HLPSL

As per the above HLPSL syntax specifications [25], our proposed PPIDA-IC scheme can be modelled as follows. It should be noted that the numbering of the steps in this subsection is similar to that of the PPIDA-IC design steps described earlier in Subsection 3.2.2. The steps are as follows.

**Step 1:** Initially, when a device D1 wants to start communication with another device D2, it does not have a password for securing its private key. It then sends the following message to the OTP-server to request one, encrypted with the OTP-server's public key:

*State=0 ∧RCV(start) =\|>State':=1*

*∧Timestamp':=new()*

*∧SND(A.{Token.Timestamp'.Hash(Token.Timestamp')}_KPo)*

where State is a local variable of every agent, which is initialized to 0 at the beginning. D1 receives a start message over the RCV channel to begin the protocol. Then, it updates its state to 1, and then it generates a timestamp, which will be used to protect against replay attacks. Here, Token is a unique number used to identify a service, and this number is assigned at the registration phase. D1 sends the message to the OTP-server over the SND

channel, which includes the token, timestamp and their hash, all encrypted using the OTP-server's public key. This message is transmitted over a radio channel (e.g. using Wi-Fi).

**Step 2:** The OTP-Server receives the message over its RCV channel and updates its state to 1. First, it validates the request by checking the timestamp, D1 identify, hash of the message and the service token. It then creates a current timestamp and fetch the password from its database. Next, it fetches the location of the device relative to its position from the database and points the Infrared trans-receiver toward D1. Then, it sends the following message, encrypted with its private key over the infrared channel to D1.

$$State{=}0 \wedge RCV(A.\{Token.Timestamp'.Hash(Token.Timestamp')\}\_KPo) =|> State':=1$$
$$\wedge Timestamp' := new()$$
$$\wedge SND(O.\{Timestamp'.Password.Hash(Timestamp'.Password)\}\_inv(KPo))$$
$$\wedge secret(Password,sec\_1,\{A,O\})$$

Where the secret() function is used to check for the password secrecy between D1 and the OTP-server. It should be noticed that the infrared channel is considered secure, therefore, the password can also be sent to D1 even in its plaintext form over the infrared channel.

**Step 3:** D1 receives the message by decrypting it. Then, it validates the message by checking the hash and timestamp, and it uses the password to retrieve its private key, which in turn is used to generate the DH session key between D1 and D2 in a similar as in [7]:

$$State{=}1 \wedge RCV(O.\{Timestamp'.Password'.Hash(Timestamp'.Password')\}\_inv(KPo)) =|>$$
$$State':=2 \wedge secret(Password',\ sec\_1,\{A,O\})$$
$$\wedge Na':=new()$$
$$\wedge Timestamp':=new()$$
$$\wedge AA' := MOD\ (EXP(G,Na'),P)$$
$$\wedge SND(A.\{A.Timestamp'.AA'.G.P.Hash(AA'.G.P.Timestamp'.A)\}\_inv(KPa))$$

**Step 4 and Step 5:** D2 validates the message by checking the hash and timestamp. To complete the DH key exchange protocol [51], D2 also sends a message (encrypted with D2's

private key) to the OTP-server to request the OTPs that will be used for second level authentication purpose. Here, the hash is used to ensure the message integrity. The OTP-server replies as follows:

*State=0* $\wedge$ *RCV(A.{A.Timestamp'.AA'.G.P.Hash(AA'.G.P.Timestamp'.A)}_inv(KPa))*

=|>

*State':= 1* $\wedge$ *Nb':=new()*

$\wedge$ *BB' := MOD(EXP(G,Nb'), P)*

$\wedge$ *SND(B.{B.Timestamp'.BB'.Hash(BB'.Timestamp'.B)}_inv(KPb))*

$\wedge$ *Key' := MOD(EXP(AA',Nb'),P)*

$\wedge$ *secret(Key', secret_key, {A,B})*

$\wedge$ *Req':=new()*

$\wedge$ *SND(B.{A.B.Timestamp'.Req'.Hash(A.B.Timestamp')}_inv(KPb))*

Where new() is a function used to create a random nonce, G and P are publicly known, and mod() and exp() are the modular and exponential operations respectively.

**Step 6 and Step 7:** After the OTP server receives the OTP-request, it validates it and generates a new timestamp; then it sends the OTPs to D1 and D2 over the infrared channel. Next, the message is encrypted with public keys of D1 and D2. Here, it should be emphasized that the secrecy of the OTPs is ensured by means of the secret() function.

*State=1* $\wedge$ *RCV(B.{A.B.Timestamp'.Req'.Hash(A.B.Timestamp')}_inv(KPb))* =|>

*State':=3* $\wedge$ *Timestamp' := new()*

$\wedge$ *SND({Timestamp'.OTP1.OTP2}_KPb)*

$\wedge$ *SND({Timestamp'.OTP2.OTP1}_KPa)*

$\wedge$ *secret(OTP1, server_otp1, {A,B,O})*

$\wedge$ *secret(OTP2, server_otp2, {A,B,O})*

**Step 8:** When D1 receives the OTPs, it updates its state to 2, then updates its session key and sends the hash of OTP2 to D2, encrypted using the newly generated session key as follows:

*State=3 /\RCV({Timestamp2'.OTP2'.OTP1'}_KPa) =|>*

*State':=4 /\Key' := XOR(Key',Hash(XOR(OTP1',OTP2')))*

*/\SND({Hash(OTP2')}_Key')*

*State=4 /\RCV({Hash(OTP1')}_Key') =|>*

*State':=5 /\witness(A,B, server_client_bb,{B.Timestamp'.BB'}_inv(KPb).Hash(OTP1'))*

*/\request(A,B, client_server_aa,{A.Timestamp'.AA'}_inv(KPa).Hash(OTP2'))*

where the functions witness() and request() are used to check for strong authentication.

**Step 9:** D2 also computes the session key in a similar way as in Step 8 and sends the hash of OPT1 to D1, protected by the session key, as follows:

*State=1 /\RCV({Timestamp2'.OTP1'.OTP2'}_KPb) =|>*

*State':=2 /\Key' := XOR(Key',Hash(xor(OTP1',OTP2')))*

*State=2 /\RCV({Hash(OTP2)}_Key') =|>*

*State':=3 /\SND({Hash(OTP1)}_Key')*

*/\witness(B,A,client_server_aa,{A.Timestamp'.AA'}_inv(KPa).Hash(OTP2'))*

*/\request(B,A,server_client_bb,{B.Timestamp'.BB'}_inv(KPb).Hash(OTP1'))*

Both D1 and D2 can now validate the received hash by computing {Hash(OTP1)}_Key' and {Hash(OTP2)}_Key' respectively. If there is a match, they have mutually authenticated each other; otherwise, the authentication has failed.

Next, the goals of keeping OTP1 and OTP2 secret, ensuring the secrecy of the session key, achieving mutual authentication between D1 and D2, and ensuring the secrecy of the password between D1 and the OTP-server, is coded as follows:

*goal*

*secrecy_of server_otp1*

*secrecy_of server_otp2*

*secrecy_of secret_key*

*secrecy_of sec_1*

*authentication_on client_server_key*

*authentication_on server_client_key*

*end goal*

### 3.2.5 Modelling of the PPIDA-PKI Scheme Using HLPSL

As per the above HLPSL syntax specifications [25], our proposed PPIDA-PKI scheme can be modelled as follows. It should be noted that the numbering of the steps in this subsection is similar to that of the PPIDA-PKI design steps described earlier in Subsection 3.2.3. The steps are as follows.

**Step 1:** This step is identical to Step1 of the PPIDA-IC scheme.

**Step 2:** The OTP-Server receives the message over its RCV channel, and updates its state to 1. Next, it validates the request by checking the timestamp, D1 identify, hash of message and service token. Then, it initiates the DH key exchange protocol [27] with D1. It should be emphasized that the SeLinux [52] is invoked here to protect the host keys. The DH key exchange is achieved using these keys. This step is modelled as follows:

*State=0* $\wedge$ *RCV(A.{Token.Timestamp'.Hash(Token.Timestamp')}_KPo) =|>State':=1*

$\wedge$ *No':=new()*

$\wedge$ *OO' := MOD(EXP(G2,No'),P2)*

$\wedge$ *SND(O.{O.Timestamp'.OO'.G2.P2.Hash(O.Timestamp'.OO'.G2.P2)}_inv(KPHo))*

Where G2 and P2 are publicly known and No' is the OTP-server's nonce.

**Step 3:** When D1 receives the message, it validates it and generates its own secret Na'; then it generates a DH key and sends the message to the OTP-server. It should be noticed that the secrecy of the session key is also checked. This step is modelled as follows:

*State=1*$\wedge$*RCV(O.{O.Timestamp.OO'.G2.P2.Hash(O.Timestamp.OO'.G2.P2)*

*}_inv(KPHo))=|>State':=2*

$\wedge$ *Na':=new()*

$\wedge$ *AA' := MOD(EXP(G2,Na'),P2)*

$\wedge$*Session_key' := mod (exp(OO',Na'),P2)*

$\wedge$*SND(A.{A.Timestamp.AA'.Hash(A.Timestamp.AA')}_inv(KPHa))*

$\wedge$*secret(Session_key',secret_key1, {A,O})*

**Step 4:** The OTP-server receives and validates the D1 message, then it generates the DH session key and sends a message containing the password encrypted with this key as follows:

*State=1* $\wedge$*RCV(A.{A.Timestamp.AA'.Hash(A.Timestamp.AA')}_inv(KPHa)) =|>*

*State':=2* $\wedge$*secret(Password,sec_1,{A,O})*

$\wedge$*Timestamp':=new()*

$\wedge$*Session_key' := MOD(EXP(AA',No'),P2)*

$\wedge$*SND(O.{{O.Timestamp'.Password.KPo.*

*Hash(O.Timestamp'.Password.KPo)}_inv(KPo)}_Session_key')*

$\wedge$*secret(Session_key',secret_key1, {A,O})*

**Step 5:** D1 receives the message by decrypting it with the session key. Next, it validates the message, then uses the password to retrieve its private key. This key is then used to generate the DH session key with D2 in a similar way than in [7], as follows:

*State=2*$\wedge$*RCV(O.{{O.Timestamp'.Password'.KPo.*

*Hash(O.Timestamp'.Password'.KPo)}_inv(KPo)}_Session_key') =|>State':=3*

$\wedge$*secret(Password',sec_1,{A,O})*

$\wedge$*Na':=new()*

$\wedge$*Timestamp':=new()*

$\wedge$*AA' := MOD(EXP(G1,Na'),P1)*

$\wedge$*SND(A.{A.Timestamp'.AA'.G1.P1.Hash(AA'.G1.P1.Timestamp'.A)}_inv(KPa))*

**Step 6 and Step 7:** These steps are modeled in a similar way that Steps 4 and 5 of the PPIDA-IC scheme were modelled, but with G and P are now replaced by G1 and P1.

**Step 8 to Step 11:** These steps are modeled in a similar way that Steps 6 to 9 of the PPIDA-IC scheme were modelled.

The goals of keeping OTP1 and OTP2 secret, ensuring the secrecy of the session keys (secret_key and secrey_key1), and the password (sec_1) secret, achieving mutual authentication between D1 and D2, and ensuring the secrecy of the password between D1 and the OTP-server, is coded as follows:

*goal*

*secrecy_of server_otp1*

*secrecy_of server_otp2*

*secrecy_of secret_key*

*secrecy_of secret_key1*

*secrecy_of sec_1*

*authentication_on client_server_key*

*authentication_on server_client_key*

*end goal*

## 3.3 Design of the RSA-ASH-SC Scheme

Our proposed RSA-based two-factor User Authentication Scheme for Smart Home using Smart Card (so-called RSA-ASH-SC) relies on a variant of the RSA algorithm [59]. For a better understanding of its design and for notations purpose, it is preferable to replicate here the steps of the RSA public key cryptosystem.

### 3.3.1 RSA Public Key Cryptosystem

The RSA algorithm [59] is composed of three steps: key generation, encryption and decryption.

**Key generation :** First, the input to this algorithm is a security parameter $n$ (also referred

to as length of the RSA). Second, two prime numbers p and q are chosen, each with $n/2$ bits long. Third, $N$ is computed as $N = p.q$. Fourth, the Euler's totient function $\phi(N)$ is calculated as $\phi(N) = (p-1).(q-1)$. Fifth, $e$ the encryption component is chosen such that $gcd(e, \phi(N)) = 1 < and < 1 < e < \phi(N)$. Typically, $e = 65537$. The next step is to derive a private key $d$ such that $d = e^{-1} mod(\phi(N))$. The public key is $(e, N)$ and private key is $(d, N)$.

**Encryption :** Either of the keys $(e, N)$ or $(d, N)$ can be used to encrypt the message (usually the public key). To decrypt the message, a second key is used. The message M should be an integer in the range $\{1, ..., (N-1)\}$. If it is not, the message from the input bit string should be formatted such that M belongs to $\{1, ..., (N-1)\}$. For this purpose, the PKCS#1 standard [60] is commonly used, and the message $M$ is encrypted as $C = M^e mod(N)$,where C is the ciphertext.

**Decryption :** The private component d and N are used to decypher the message C. To retrieve the original message M, one calculates $C^d mod(N)$ Since d and N are large numbers, it is very computation intensive, and to ensure a fast convergence of the decryption algorithm, the Chinese Remainder Theorem has been used as standard. It has been proven that its use can substantially boost the performance of the RSA (about four time faster compared to when it is not used) [60]. More precisely, $M^p$ and $M^q$ are calculated as follows:

$$M^p = C_p^{d_p} (mod\, p)$$

$$M^q = C_q^{d_q} (mod\, q)$$

Where $d_p = d(mod\, p)$ and $d_q = d(mod\, q)$, and the message M is retrieved using the Chinese Remainder Theorem [60].

### 3.3.2 RSA-ASH-SC Scheme

For the design of this scheme, the notations used are given in Table 3.3

In this algorithm, for the cryptography purpose, a fast variant of RSA, which is a com-

Table 3.3: Notations used in the proposed RSA-ASH-SC scheme

| **Notation** | **Description** |
|---|---|
| $\phi(N)$ | Euler's totient |
| $n$ | Input security parameter for key generation algorithm |
| $k$ | Distinct prime numbers in RSA key generation |
| $s$ | Size of prime numbers in RSA key generation (Rebalanced) |
| $p, q$ | Prime numbers used in RSA key generation |
| $e$ | Encryption exponent |
| $d$ | Decryption exponent |
| $mod()$ | Modulus operation |
| $gcd()$ | Greatest common divisor |
| $\oplus$ | XOR operation |
| $h(), f()$ | One way Hash function |
| $\delta T$ | Threshold time used to prevent replay attack |
| $\{\}_{S_i}$ | Symmetric key encryption/decryption, here $S_i$ is symmetric key |
| $U_i$ | $i^{th}$ user |
| $ID_i$ | $i^{th}$ user's ID |
| $PW_i$ | $i^{th}$ user's password |
| $OTT_i$ | $i^{th}$ user's One-Time-Token |

bination of Multi-power and Rebalance RSA algorithms [9], is considered. The proposed RSA-ASH-SC scheme is a two-factor remote authentication scheme, the first of which is password (i.e. what you know factor) and the second is smart card (i.e. what you have factor). The user is required to register (one time process) beforehand with the system. Each user $U_i$ submits a hash of his chosen password $PW_i$ in a secure manner and a smart card issuer generates a unique $ID_i$ and other information, which are saved in the smart-card, to be used for remote user authentication purpose. The steps of the RSA-ASH-SC scheme are as follows:

**Initialization phase:** The RSA keys are generated as per the method described in [9]. In our scheme, the card issuer (in Smart Home, the issuer can be himself) is the same as the authentication server/system. The key generation algorithm takes two security parameters

as inputs: $n$ and $k$. First, it generates $n/k$ bits long two prime numbers $p$ and $q$, such that $gcd((p-1),(q-1)) = 2$. Then, it calculates $N = p^{(k-1)}.q$. Next, it generates two random numbers $r1$ and $r2$ in such a way that $gcd(r1,(p-1)) = 1$, $gcd(r2,(q-1)) = 1$ and $r1 = r2 < mod(2)$. Then, it finds the integer $d$ such that $d = r1 < mod(p-1)$ and $d = r2 < mod(q-1)$. Finally, it calculates $e$ such that $e = d^{-1} mod(\phi(N))$. Here, the public key is $(e, N)$ and the private key is $(p, q, r1, r2)$, which is kept secret with the card issuer.

**Registration Phase:** The user submits a request, in secure manner, to the issuer for the smart card by sharing the hash of his password $HPW_i = h(PW_i)$, where $h()$ is a one-way hash function and $PW_i$ is the password of the user. Upon receiving the request, the issuer-server creates a random and unique $ID_i$ for the user. The server also creates a random one-time-token $OTT_i$ to keep the future authentication requests. Next, the server calculates $HPWID_i = h(HPW_i \oplus ID_i)$ and stores this value and $OTT_i$ in its database, which is protected by its private key. Next, the server stores the following information $(ID_i, OTT_i, h, e, N)$ on the smart card. The smart card is then physically handed over to the user.

**Login phase:** The user requires a smart card reader before starting the login process. First, the user connects the smart card reader and writer (SCRW) to the personal digital assistant (PDA). Then, he opens the application which makes use of the proposed authentication. The user enters the password $PW_i$ which is sent to the card, for example, using an application protocol data unit (APDU) [61]. This application also generates and sends a random secret key $S_i$ to the smart card. It should be noted that the secret key is generated for every new session. The card then performs the following steps. First, it computes $x$ as $x = (h((h(PW_i) \oplus ID_i))^e \oplus h(T)) mod\, N$, where T is a newly created timestamp. Then, it computes $HXOTT_i = h(x \oplus OTT_i)$, Next, it computes $y = (OTT_i||T||S_i||h(OTT_i||T||S_i||HXOTT_i)||HXOTT_i)$ and encrypts y as $C = y^e mod\,(N)$, then sends the following message $(OTT_i, C)$ to the server.

**Authentication phase:** Upon receipt, the server compares $OTT_i$ against the entries in

the database. If there is a match, it decrypts the message. To achieve this, it computes $M_1 = C^{r1} mod(p)$ and $M_2 = C^{r2} mod(q)$. Using CRT, it calculates $M \in Z_N$ such that $M = M_1 mod\,(p)$ and $M = M_2 mod\,(q)$.

The received message $M = (OTT_i||T|||S_i||h(OTT_i||T||S_i||HXOTT_i)||HXOTT_i)$. Then, it checks whether the timestamp is recent or not, i.e. $(T_c - T) < \delta T$ where $T_c$ is the current time and $\delta T$ is the acceptable timestamp difference. It also verifies $OTT_i$ within message $M$ again with the unencrypted $OTT_i$. It also verifies the hash of the message to check if the message was tempered. Then, it computes $x = ((HPWID_i)^e \oplus h(T)) mod\, N$ and $Z = h(x \oplus OTT_i)$. If $Z$ and $HXOTT_i$ are equal, it authenticates the request. The server then creates a new random token $OTT_{new}$ and current timestamp $T_{new}$, then computes the response as $M_r = (T_{new}\{h(T_{new}||OTT_{new}).OTT_{new}\}_{S_i})$ and sends that response to the user. Here, $\{\}_{S_i}$ is the encryption/decryption function and $S_i$ is the symmetric key. The server also updates the token $(OTT_i)$ of the corresponding $ID_i$ and $OTT_{new}$.

When the user receives the response, it decrypt the message $M = (T_{new}\{\{h(T_{new}||OTT_{new})$ $OTT_{new}\}_{S_i}\}_{S_i})$. First, it checks whether the timestamp is recent or not, i.e. $(T_c - T_{new}) < \delta T$. If it is the case, it stores the new token $OTT_{new}$ in the card for the new time and uses $S_i$ for further communication; otherwise it drops the authentication request.

**Password change:** To update the password, the user needs to be authenticated in advance. The user enters its password and calculates the hash of the new password as $HPW_{new} = h(PW_{new})$, then sends a password update command to the server as $CMD = \{pass_{update}, T, h(T, HPW_{new}), HPW_{new}\}_{S_i}$, where $pass_{update}$ is a known command to the server. After receiving the command, the server decrypts the message using $S_i$ and validates the timestamp and hash of the message as described above. If validated, the server computes $HPWID_{new} = h(HPW_{new} \oplus ID_i)$. then updates the $HPWID_i$ in the database corresponding to the user $ID_i$.

The proposed RSA-ASH-SC Scheme can be summarized as follows:

**Client <——————————————————————————————- >Server**

$x = (h((h(PW_i) \oplus ID_i))^e \oplus h(T)) mod\, N$

$HXOTT_i = h(x \oplus OTT_i)$

$y = (OTT_i||T||S_i||h(OTT_i||T||S_i||HXOTT_i)||HXOTT_i)$

$C = y^e mod\,(N)$

$(OTT_i, C)$

$M_1 = C^{r1} mod(p)$

$M_2 = C^{r2} mod(q)$

$Using\, CRT\, M \in Z_N$

$M = (OTT_i||T||S_i||h(OTT_i||T||S_i||HXOTT_i)||HXOTT_i)$

$If (T_c - T) < \delta T$

$Verify\, OTT_i$

$x = ((HPWID_i)^e \oplus h(T))\, mod\, N$

$Z = h(x \oplus OTT_i)$

$If\, Z == HXOTT_i$

$Create\, OTT_{new}\, and\, T_{new}$

$M_r = (T_{new}\{h(T_{new}||OTT_{new}).OTT_{new}\}_{S_i})$

$M = (T_{new}\{\{h(T_{new}||OTT_{new}).OTT_{new}\}_{S_i}\}_{S_i}$

$If\, (T_c - T_{new}) < \delta T$

$Updates\, OTT_{new}$

# Chapter 4

# Simulations and Security Analysis of the Proposed Schemes

This Chapter presents the security and performance analysis of the proposed protocols. The back-ends of the AVISPA tool [25] are used to simulate and verify the D2DA-OTP-IC, PPIDA-IC, and PPDA-PKI protocols against attacks. Through informal security analysis, the ability of the proposed schemes to resist against various common cryptographic attacks is checked.

## 4.1 Security and Performance Analysis of the D2DA-OTP-IC Scheme

### 4.1.1 SPAN/AVISPA Tool

The SPAN/AVISPA tool [25] is used to determine whether the proposed D2DA-OTP-IC protocol (in HLPSL format) is secure or not in terms of ensuring the secrecy of the session key and achieving mutual authentication. The authentication is achieved when two devices D1 and D2 can validate their identities using OTPs. The secrecy refers to ensuring that the

information that is exchanged between devices D1 and D2 cannot be deciphered or cannot be disclosed to an unauthorized entity and only the intended receiver can decipher it; integrity refers to the fact that it is not possible for an intruder to alter or destroy the information shared by devices D1 and D2 during the communication process. This is achieved by appending a hash to the message at its source and by re-computing the hash of the message at the destination, then check for a match between the two quantities. If a protocol is found unsafe, the AVISPA tool [25] will reveal the trace of the attack found; otherwise the protocol is qualified as safe against attacks. This is a distinguishing feature of the AVISPA tool compared to other available tools such as BAN Logic [26].

### 4.1.2 Simulation of the D2DA-OTP-IC Scheme Using AVISPA

In the proposed D2DA-OTP-IC scheme, the session keys (symmetric keys) are utilized to encrypt the message and the DH key exchange algorithm [51] is used for key exchange purpose over an insecure channel. The simulation results of our proposed scheme are attributed to the OFMC and CL-AtSe back-end modules of the AVISPA tool [25] as the other two back-end modules: SATMC and TA4SP [25] have reported NOT SUPPORTED and have produced INCONCLUSIVE results. The SPAN gives a better understanding of the protocol and it is used to confirm whether the specification is executable or not. It also helps visualizing the operations of the protocol as shown in Fig. 4.1

**Verification of the results:** The two back-ends modules of AVISPA, namely OFMC and CL-ATSe reported that the proposed D2DA-OTP-IC protocol is safe as confirmed in Fig. 4.2 and Fig. 4.3 respectively.

Indeed, Fig. 4.2 and Fig. 4.3 indicate that no authentication attack, nor secrecy attack has occurred on the proposed D2DA-OTP-IC protocol. In addition, no attack is found on the session key by the intruder and the secrecy of the transferred message between devices. It is also shown that the session keys are also maintained. In the proposed D2DA-OTP-IC

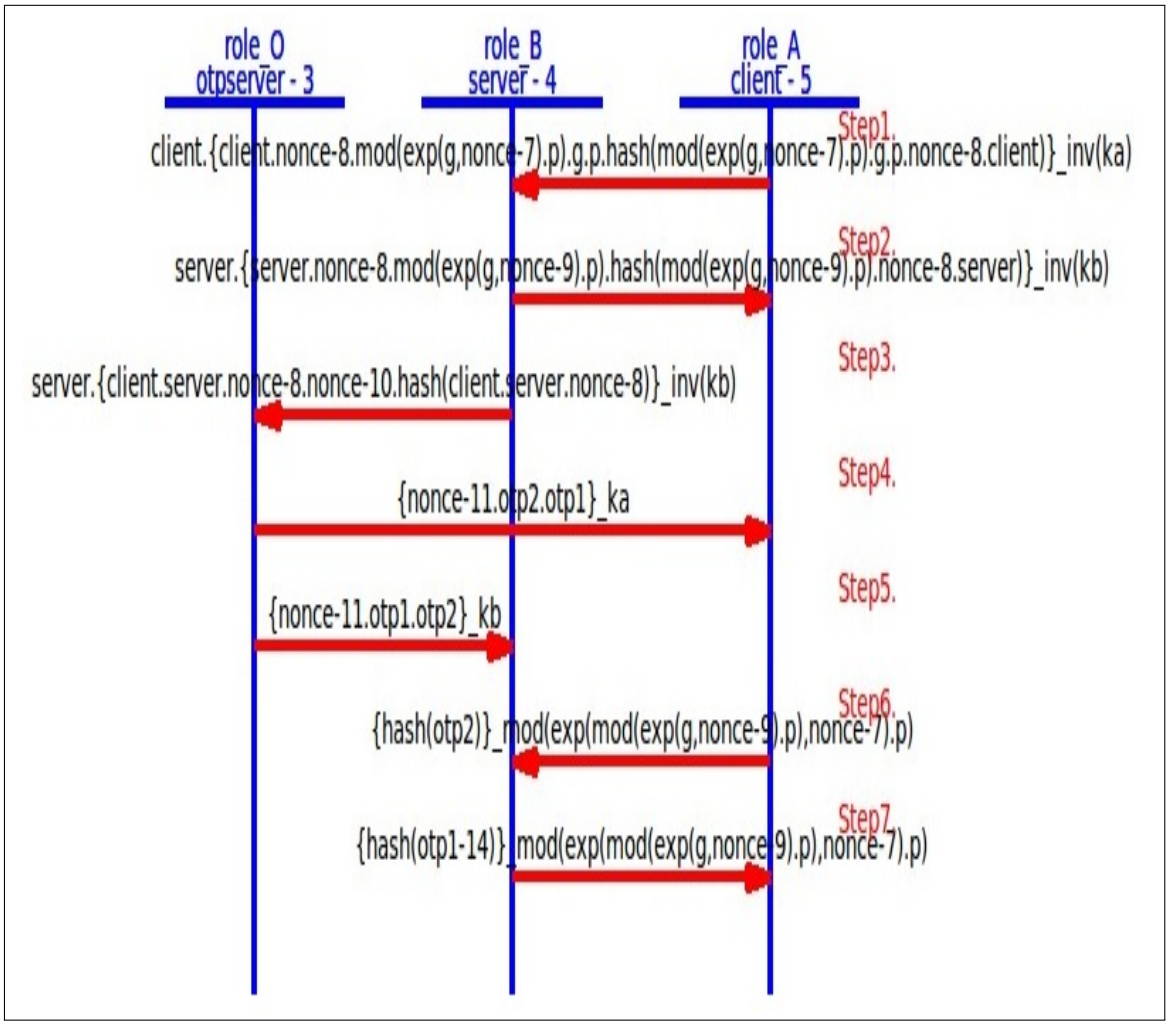


Figure 4.1: Simulation of the D2DA-OTP-IC protocol in AVISPA.

protocol, we have exchanged the session keys using the DH key exchange algorithm [51], and this was checked for secrecy purpose and the AVISPA tool validated this checking. We have also checked for the secrecy of OTP1 and OTP2, which was also validated by the AVISPA tool. In addition, our goals of keeping the OTP1 and OTP2 secret, ensuring the secrecy of the session key, and achieving mutual authentication between the client and the server, have been validated by the SPAN/AVISPA tool since the protocol is reported as safe.

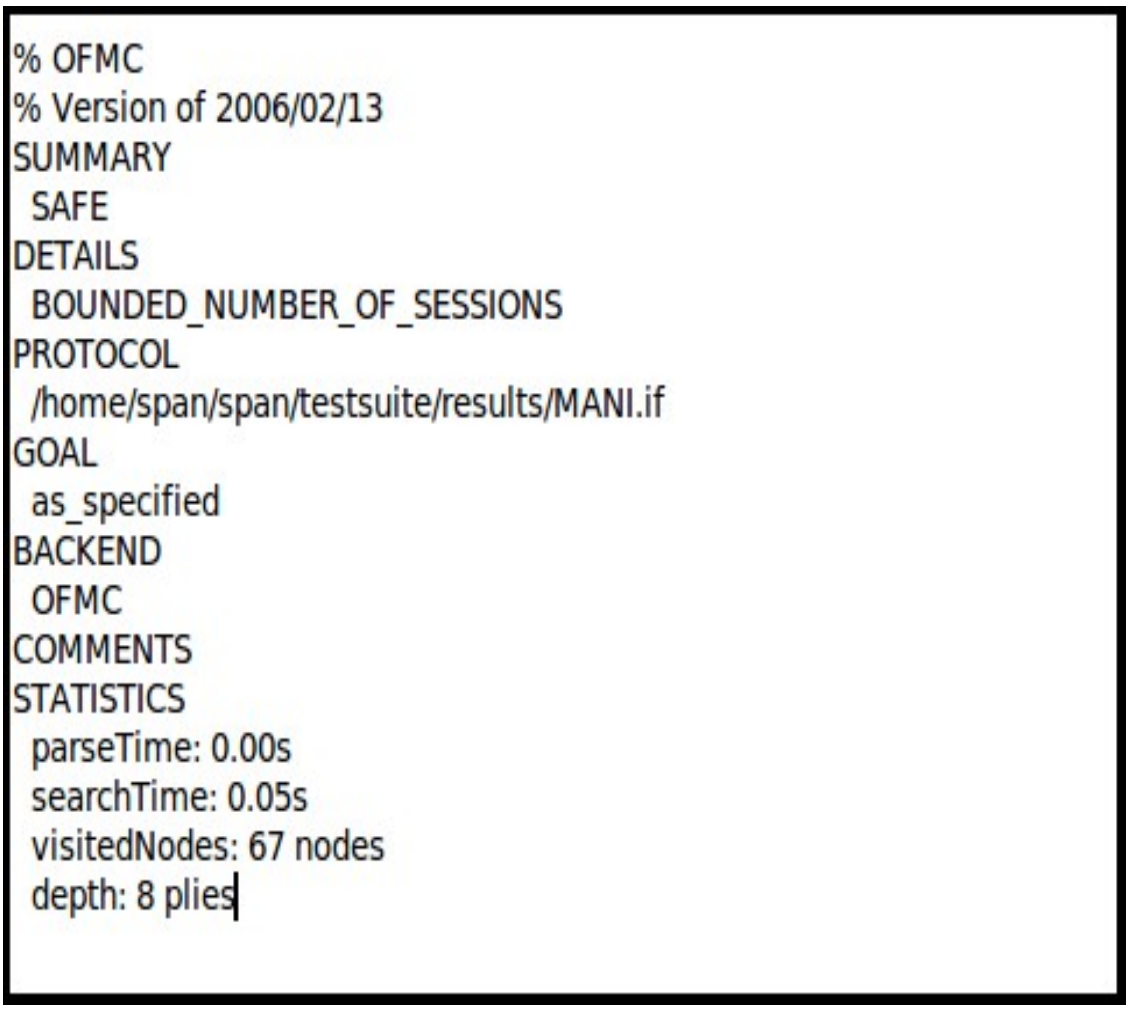


Figure 4.2: D2DA-OTP-IC protocol verification using the OFMC back-end.

### 4.1.3 Informal Security Analysis of the D2DA-OTP-IC Scheme

In this subsection, we discuss about some attacks and how our proposed D2DA-OTP-IC scheme can be used to protect against them.

**Eavesdropping attack:** In this attack, the eavesdropper silently listens to the communication of others without their knowledge. He/she may gain some sensitive information if not protected. In our scheme, the DH key exchange algorithm is used to construct the session keys, and the identity of the messages is signed using the private keys, which only the owner has access to. In addition, on top of it, the OTPs are sent over a light communication and an eavesdropper cannot get access to these entities after a successful authentication since

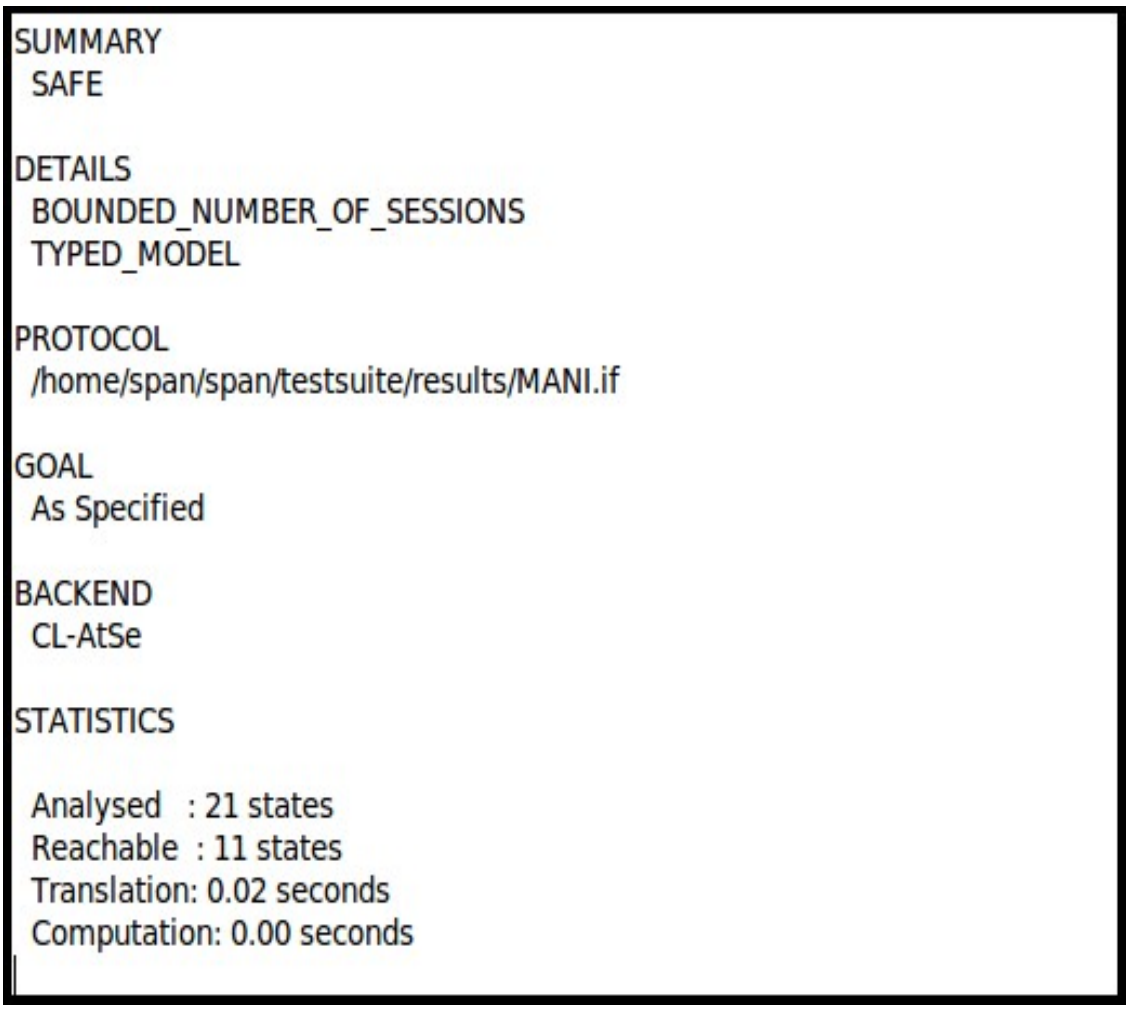


Figure 4.3: D2DA-OTP-IC protocol verification using the CL-AtSE back-end.

all the communication entities use the session key to encrypt the data and the eavesdropper cannot make anything out of it. The session keys are generated for each session in order to maintain high security.

**Masquerading/Impersonation attack:** In this kind of attack, the intruder presents himself as a genuine entity to gain the unauthorized access. Our scheme makes use of digital signature to validate the messages in the first phase, where the session keys are exchanged. Therefore, masquerading attempts will be blocked right at the first phase.

**Replay attack:** In this kind of attack, the intruder first captures the data packets which can be used later to gain access to the system. To protect against this type of attack, the

current timestamp is used in our scheme, which is included in the digital signature, and it is assumed that the systems sync on time. On top of it, OTPs are used, which are randomly generated for each session.

**Message modification attack:** In this type of attack, the intruder tries to tamper the message. Our scheme uses a public key Infrastructure in its first phase to check if the message is legitimate or not. In its second phase, the hashed OTPs are sent by encrypting them with the session keys. If the message is tampered at any phase, the authentication will fail. Therefore, our scheme can be used to protect against message modification.

**Reflection attack:** In this type of attacks, the attacker tries to gain access to the system by taking advantage of the protocols weakness, for instance, when encryption and decryption are performed without prior checking of the identity of the user. In the caseof our protocol, the attacker can only gain access to the radio channel (e.g. Wi-Fi). In the first two steps, the message is encrypted using the private keys $K'_D1 and K'_D2$), but the secret nonce N1 and N2 are never sent to a third party. Therefore, the attacker has no way to get these nonce, which are used to establish the secret session key (using the DH key exchange [51]). The communication between device D2 and the OTP-server is also signed by D2's private key; the attacker can get hold of this message if he has the public key of D2, but there is no sensitive information at this step of the protocol operations. In the final two steps, the hash of the OTPs are protected with the secret key SK before being exchanged and the attacker cannot decrypt this information. Therefore, this attack will not work.

**Multi-Protocol attack:** In this attack, multi-protocols are used to target one of the protocol and the attacker authenticates itself by misleading the protocol. In our scheme, the attacker cannot generate the signed messages between devices D1 and D2 since he does not have access to the private keys. Thus, even if he tries with multiple protocols, the response from D1 and D2 will be different since random nonce N1 and N2 (used to establish the session key for every new session) are generated each time. The attacker also does not have access to the OTPs and cannot authenticate itself. Hence, the attacker cannot compromise

the protocol.

A qualitative comparison the proposed D2DA-OTP-IC scheme against few schemes in terms of security attacks is given in Table 4.1

Table 4.1: Comparison of Selected Authentication Schemes based on Security Attacks

| | Shin et al. [32] | Kumar et al. [17] | Kumar et al. [33] | Markts-cheffel et al. [63] | Ruj et al. [62] | Santoso et al. [34] | Preethy [27] | Proposed Scheme |
|---|---|---|---|---|---|---|---|---|
| MIMA | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓* | ✓ |
| Impersonation attack | | | | | | | | ✓ |
| Replay attack | | ✓ | ✓ | ✓ | | | ✓ | ✓ |
| Password guessing attack | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Forgery attack | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ |
| Eavesdropping attack | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ |
| Brute-Force attack | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ |
| Repudiation | | | | | ✓* | ✓ | | ✓ |
| Forward Secrecy | | | | | | | | ✓ |
| Denial of Service attack | ✓ | | ✓ | ✓ | | | ✓ | |

A qualitative comparison the proposed D2DA-OTP-IC scheme against few schemes in terms of some authentication characteristics is given in Table 4.2.

A qualitative comparison the proposed D2DA-OTP-IC scheme against few schemes based on considered evaluation models is given in Table 4.3.

* means subject to special consideration. In our proposed D2DA-OTP-IC scheme, a proof-of-concept has been implemented.

Table 4.2: Comparison of Selected Authentication Schemes based on some Authentication Characteristics

| | Shin et al. [32] | Kumar et al. [17] | Kumar et al. [33] | Markts-cheffel et al. [63] | Ruj et al. [62] | Santoso et al. [34] | Preethy [27] | Proposed Scheme |
|---|---|---|---|---|---|---|---|---|
| Mutual-Authentication | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ |
| Additional-Hardware | | | ✓ | ✓ | ✓ | | | ✓ |
| Multiple-Credential | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ |
| Multiple-Authentication-Levels | | | | | | | | ✓ |
| Registration | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ |
| Offline-Phase | | | ✓ | | | | | |
| Anonymity | | ✓ | | | | | | |

Table 4.3: Comparison of Selected Authentication Schemes based on evaluation models

| | Shin et al. [32] | Kumar et al. [17] | Kumar et al. [33] | Markts-cheffel et al. [63] | Ruj et al. [62] | Santoso et al. [34] | Preethy [27] | Proposed Scheme |
|---|---|---|---|---|---|---|---|---|
| Implementation | | | | | | ✓ | | ✓* |
| Simulation | | ✓ | ✓ | | | | ✓ | ✓ |
| Theoratical-Evalution | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Performance-Evaluation | | ✓ | ✓ | | | | | |

### 4.1.4 Proof of Concept of the D2DA-OTP-IC Scheme

We have developed a proof-of-concept of the D2DA-OTP-IC scheme in the form of a hardware design, using three Raspberry-Pi (B+, Pi2, and Pi3) as home devices. For this, the OTP-server was running on Raspberry-Pi3, the server was running on Raspberry-Pi-B+, and the client was running on Raspberry-Pi2. All Raspberry devices were running with the Raspbian Operating System. Each of these devices was embedded with an infrared transceiver circuit

as shown in Fig. 4.4.

For the infrared communication, we have used an infrared emitter (IR333c LED) to send the

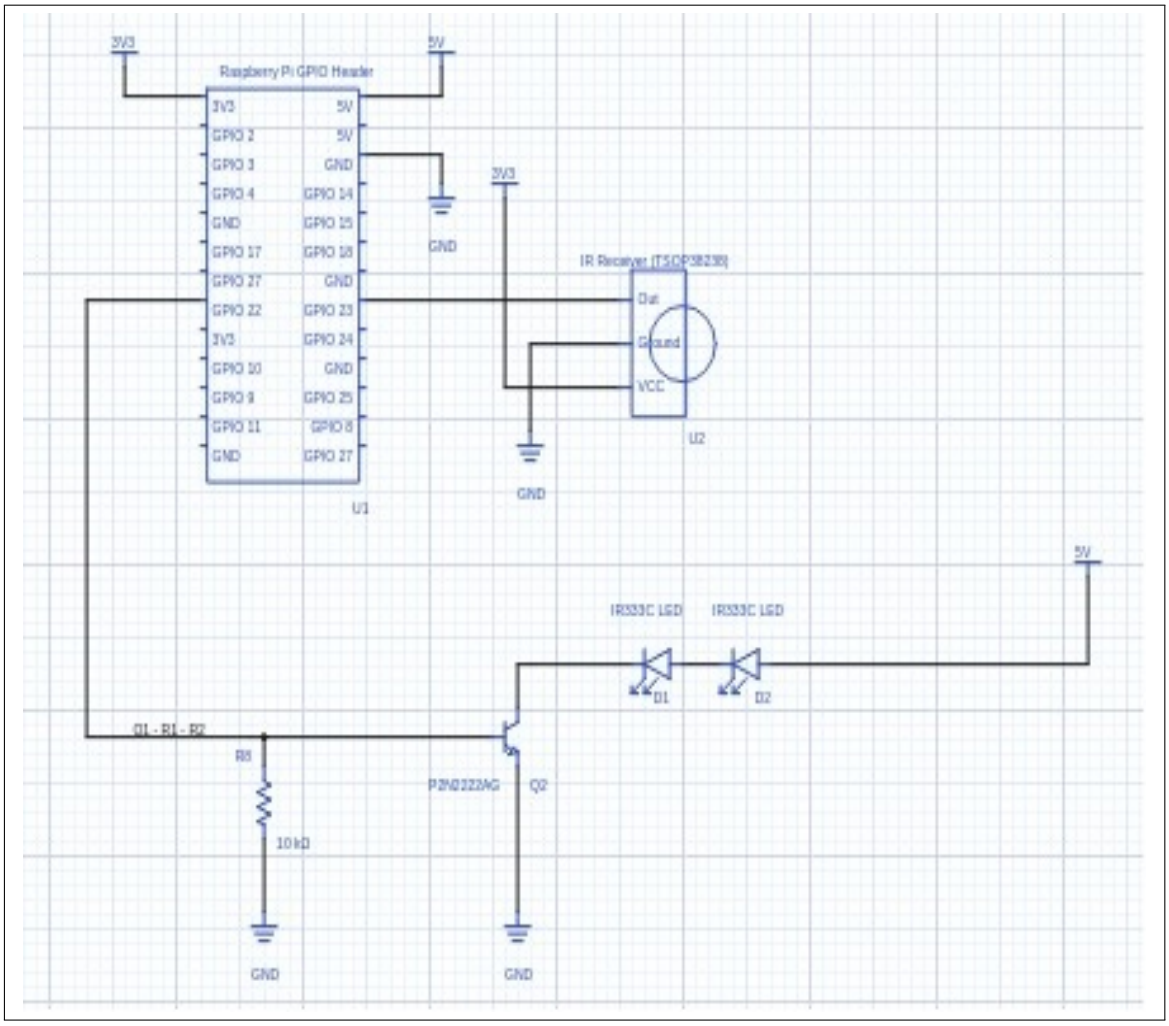


Figure 4.4: Raspberry Pi Infrared trancever circuit for LIRC [64].

infrared signal and the TSOP38238 as an infrared receiver. To send and receive the infrared signals, we have used the Linux Infrared Remote Control (LIRC) [65], which is meant for infrared remote communication. The LIRC package on Linux comes with multiple programs; we have used the "*irsend*" and "*irw*" programs. To configure the LIRC, a *config-file* [65] was used.

The OPT-server was installed with a 3D rotation device made of two SG90 Micro servo motors, which can be rotated only by 180 degrees. To build a 3D rotation, we integrated

the two servos together as shown in Fig. 4.5

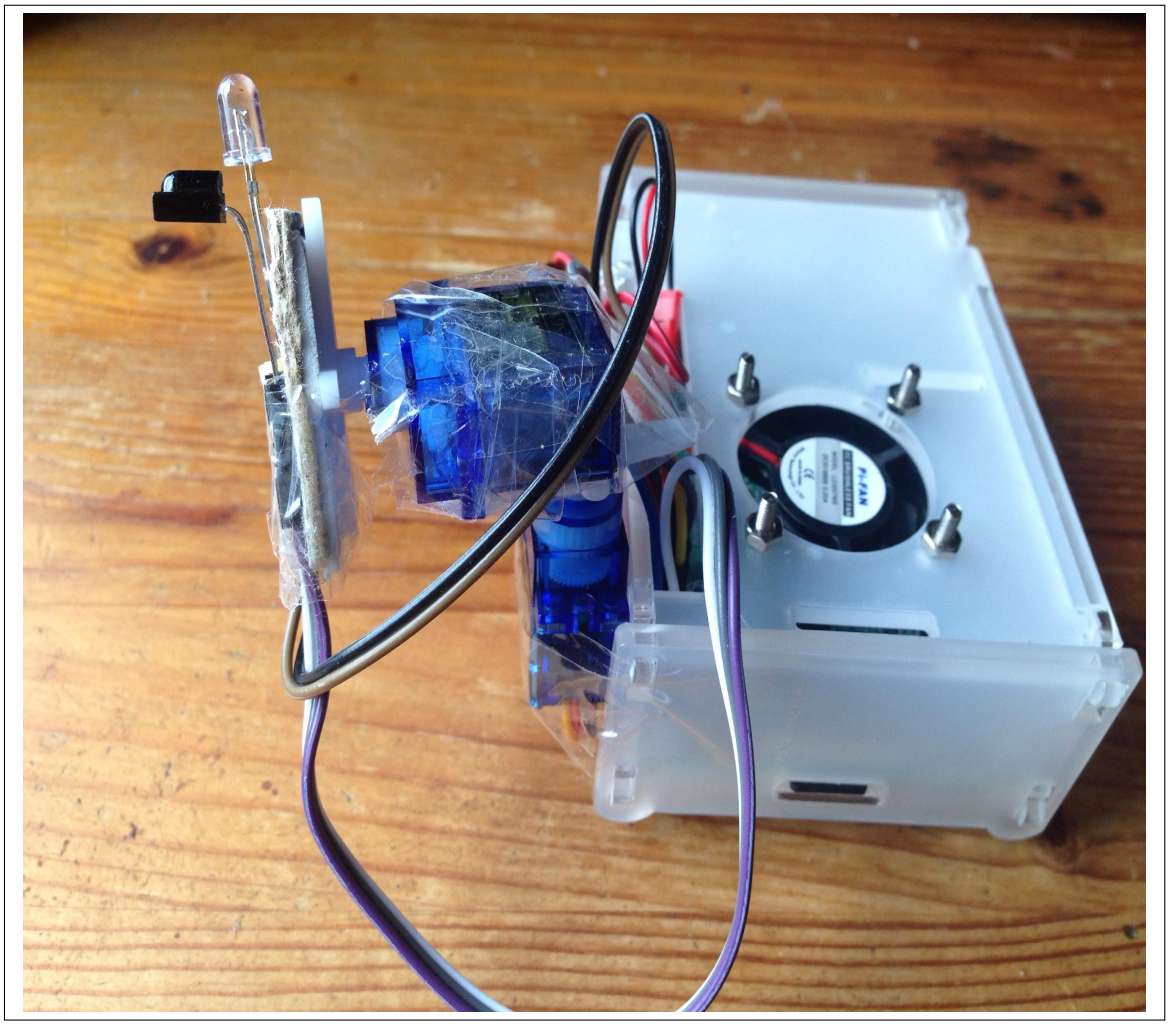

Figure 4.5: 3D rotating servo placement.

The infrared transceiver was set on top of it to send the infrared signal in the direction that the 3D device is pointing to. The servo circuit diagram is shown in Fig. 4.6.

To control the servos, a program in Python was developed using the GPIO library [65]. For the first level of authentication, the OpenSSH was utilized. On the server side, a Pluggable Authentication Module (PAM) was developed, which ran on Python; for this purpose, the "*python-pam*" python module was installed. This PAM module achieved an infrared

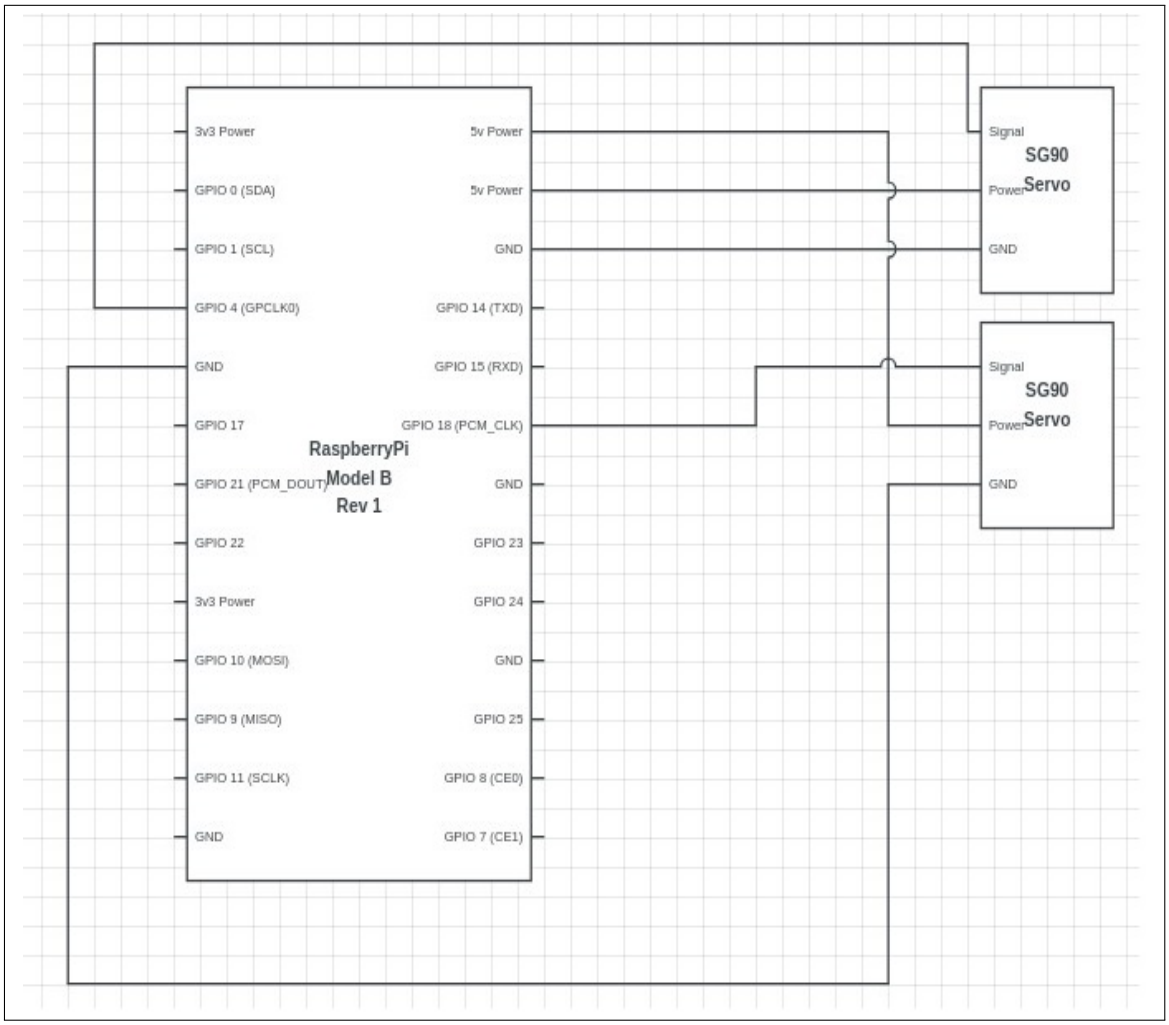


Figure 4.6: 3D rotating servo circuit.

communication-based OTP authentication as second level of authentication.

**Setup Phase:** Initially, the Raspbian OS was downloaded from [66] and installed on all the devices. Python came by default, and the "*pyopenssl*" and "*python-pam*" python modules were installed, followed by"lirc" the Linux package "*lirc*". Next, the LIRC remote config file [65] was installed and the sshd PAM *(/etc/pam.d/sshd)* was configured in order to enable the PAM module (OTP_PAM.py) for ssh. The *OTP_PAM.py* was also configured using the OTP-servers IP as follows:

*auth required pam_python.so OTP_PAM.py* . Next, the sshd config file *(/etc/ssh/sshd_config)* was configured to enable the use of public-key based authentication using PAM as follows:

*PasswordAuthentication no*

*UsePAM yes*

*AuthenticationMethods publickey,keyboard-interactive*

To create the public-private key pairs for public-key authentication, we have used *ssh-keygen*. We have used user "*pi*" for this proof of concept; ssh uses this information to create the identity. The public key was copied from the client machine to the server machine, and the public-private key pair was generated in the *PEM* format for the OTP-server using the "*openssl*" command. Then, the public key of the OTP-server was copied to both devices. The OTP-server ran OTP-server.py, which created and served the OTPs.

As the part of the setup, the client and server devices direction were registered in the *OTP-server.py* file as well, relatively to the 3D rotating device. After these configurations were done, the ssh was restarted to enable a new configuration.

To setup the hardware, the 3D rotating device was attached to the OTP-server and the *GPIOs* were connected to the device as shown in Fig. 4.5. Then, the infrared transceiver circuit was connected as shown in Fig. 4.6. For both the client and the server device, the infrared circuit were also configured as shown in Fig. 4.6.

**Experimental Phase:** To run the proposed hardware design, *LOGIN.py* was executed on the client machine (i.e. the machine which was requesting the authentication). *LOGIN.py* uses ssh to establish the session key using the DH key exchange algorithm [51]. After the server and the client have authenticated each other, and have generated a session key, the PAM module was called on the server side for the second phase authentication. It should be noted that using the ssh public-key authentication alone was not sufficient and there was a need for a second phase to successfully achieve the device-to-device authentication. The PAM module got the information on both the client and server IPs and used it to requests for OTPs to the OTP-server. In response, the OTP-server sent back two OTPs (one for the client and one for the server). It should be noticed these OTPs could further be encrypted in order to strengthen the security of the proposed scheme. Next, both the client and the

server calculated the hash of their OTPs and sent these values to each other over a Wifi channel. Then, both devices checked if the hash of each other OTPs matched with the received hashed-OTP; if that was the case, mutual authentication had occurred; otherwise, it had failed.

**Results:** For the proof of concept purpose, all the information have been recorded in log files. The experimental results are shown in Fig. 4.7.

On the left-down of Fig. 4.7, the OTP-servers logs showed which OTPs have been generated

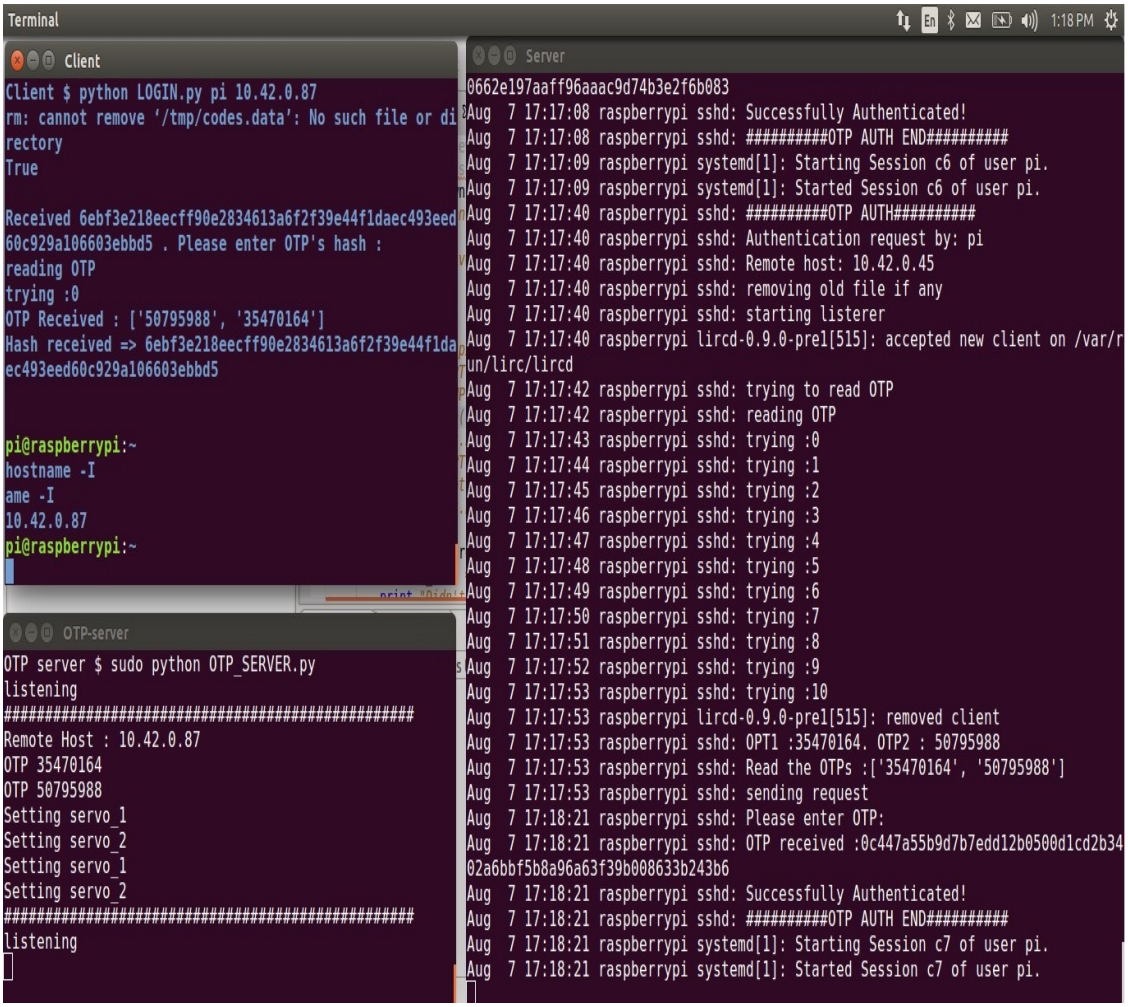


Figure 4.7: OTP based mutual authentication using infrared channel.

and which device requested the OTPs. The OTP-server sent the OTPs (OTP1, OTP2) to both devices, but in the opposite order. On the right-hand side of Fig. 4.7, the servers Syslog

file is shown, where the PAM module aws logging. The results were displayed by running the tail command on the Syslog file *(/var/log/syslog)*. This shows which device requested for authentication; once it asked for the OTP-server to provide OTPs, it had wait until both OTPs were received. It is observed that both OTPs are received over the infrared channel. It then sends the hash of OTP1 to the client over a Wifi connection and waits for the hashed OTP2. On receiving the hashed OTP2, it validates it, which is shown since it is observed that the authentication has succeeded. On left-top of Fig. 4.7, the client logs are shown, where LOGIN.py was the script used for authentication purpose. First, the public-key authentication occurred, then the OTPs (OTP2, OTP1) were received by the devices. From the server side, it received the OTP1 hash, which was validated and the hash of OTP2 was sent back.

During this experiment, we had encountered an issue in the infrared emission, i.e. the LED transmission could be detected in a wide range. To fix this problem, a narrow pipe was created and foiled, which was used to the infrared light on the infrared LED and the back side of the pipe was immediately closed after installation. In conducting the above experiments, two python programs have been developed, namely OTP_SERVER.py and servo_controller.py to generate the OTP and send it over the infrared channel.

## 4.2 Security and Performance Analysis of the PPIDA-IC and PPDA-PKI Schemes

The AVISPA tool [25] is also used to verify if the proposed PPIDA-IC and PPDA-PKI protocols satisfy the aforementioned three security related properties, namely mutual authentication; integrity and secrecy. The simulation results of our proposed scheme are also attributed to the OFMC and CL-AtSe back-end modules of the AVISPA tool [25] as the other two back-end modules: SATMC and TA4SP [25] have reported NOT SUPPORTED

and have produced INCONCLUSIVE results.

### 4.2.1 Simulation Results of the PPIDA-IC Scheme Using AVISPA

Results for the PPIDA-IC scheme are shown in Fig. 4.8 and Fig. 4.9.

In Fig. 4.8 and Fig. 4.9, it is detected that no authentication attack has occurred on the


Figure 4.8: PPIDA-IC protocol verification with the OFMC backend.

PPIDA-IC protocol, nor secrecy attack. Also, no attack is found on the session key by the intruder and the secrecy of the transferred message between devices. In addition, the session key is maintained. In the proposed PPIDA-IC scheme, we have exchanged the session keys


Figure 4.9: PPIDA-IC protocol verification with the CL-AtSe backend.

using the DH key exchange algorithm [51] and this was checked for secrecy purpose by the AVISPA tool [25], which validated it. We have also check for the secrecy of both OTP1 and OTP2, which this was validated by the AVISPA tool [25] as well. Finally, our goals of keeping the OTP1 and OTP2 secret, ensuring the secrecy of the session key, and achieving mutual authentication between the client and the server, have also been validated by the AVISPA tool [25] since the protocol is reported as safe.

### 4.2.2 Simulation Results of the PPIDA-PKI Scheme Using AVISPA

Similarly, for the PPIDA-PKI scheme, the SPAN tool [25] also gives a better understanding of the protocol and it is used to confirm whether the specification is executable or not. The results for the PPIDA-PKI scheme using AVISPA [25] are shown in Fig. 4.10 and Fig. 4.11.

Fig. 4.10 and Fig. 4.11 exhibit the same type of behavior observed in the case of the

```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /home/span/span/testsuite/results/PROXY_PASSWORD_OTP_IR_WITH_HASH.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.24s
  visitedNodes: 228 nodes
  depth: 12 plies
```

Figure 4.10: PPIDA-PKI protocol verification with the OFMC backend.

PPIDA-IC scheme (Fig. 4.8 and Fig. 4.9), and the AVISPA tool [25] alos reported the PPIDA-PKI protocol as safe.

```
SUMMARY
 SAFE

DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
 TYPED_MODEL

PROTOCOL
 /home/span/span/testsuite/results/PROXY_PASSWORD_OTP_IR_WITH_HASH.if

GOAL
 As Specified

BACKEND
 CL-AtSe

STATISTICS

 Analysed   : 15 states
 Reachable  : 8 states
 Translation: 0.02 seconds
 Computation: 0.00 seconds
```

Figure 4.11: PPIDA-PKI protocol verification with the CL-AtSe backend.

### 4.2.3 Informal Security Analysis of the PPIDA-IC and PPIDA-PKI Schemes

In this section, we discuss about some attacks and how our proposed PPIDA-IC and PPIDA-PKI schemes can be used to protect against them.

*Man-In-Middle attack (MIMA):* In this attack, the attacker can modify/delete/generate messages, delete messages. In our proposed schemes, pre-shared public keys are used and all communication is in encrypted form, except for the infrared (IR)-based password transmission. However, the IR channel is considered secure and no new keys are accepted, i.e. the

attacker has no way to succeed.

*Dictionary attack:* This kind of attack is applied to retrieve the passwords from a hashed file. In our schemes, the passwords are encrypted and a salt is used to make it even more stronger. The encryption key is protected by the private key, which in turn is protected by SELinux [11]. Thus, our scheme is secure against the dictionary attack.

*Eavesdropping attack:* In this attack, the eavesdropper silently listens to the communication of others without their knowledge. He may gain some sensitive information if not protected. In our schemes, the DH key exchange algorithm [27] is used to construct the session keys, and the identity of the messages is signed using the private keys, which only the owner has access to. In addition, the OTPs password is sent over a light communication and an eavesdropper cannot get access to these entities after a successful authentication since all the communication entities use the session key to encrypt the data and the eavesdropper cannot make anything out of it. The session keys are generated for each session in order to maintain a high security.

*Masquerading/Impersonation attack:* In this kind of attack, the intruder presents himself as a genuine entity to gain unauthorized access. Our scheme makes use of digital signature to validate the messages in the first phase, where the session keys are exchanged. The keys are protected with a passphrase; even if the keys are stolen, the attacker cannot use them directly. In addition, the second level authentication requires the use of OTPs, which are sent only to the device in its physical location. Therefore, impersonation attack cannot succeed.

*Replay attack:* In this kind of attack, the intruder first captures the data packets which can be used later on to gain access to the system. In our schemes, the current timestamp is used, which is included in the digital signature assuming that the system is sync on time. In addition, the OTPs are used, which are randomly generated for each session.

*Message modification attack:* In this type of attack, the intruder tries to tamper the message. Our scheme uses a PKI in its first phase to check whether the message is tampered or not using a hash. In its second phase, the hashed OTPs are sent by encrypting them

with the session keys. If the message is tampered at any phase, the authentication will fail. Therefore, our schemes can be used to protect against message modification.

## 4.3 Informal Security Analysis of the RSA-ASH-SC Scheme

In this subsection, we informally analyze the security of our proposed RSA-ASH-SC scheme to show that it can protect against few attacks.

*Forgery attack:* In this scheme, the anonymity is ensured by the use of one-time tokens. The attacker cannot get any information from the one-time token and encrypted message. The encryption and decryption operations are performed by using Rebalanced-Multi-Power RSA variant [22]. The login message makes use of a password, a unique ID which is associated with a user and a timestamp. For attacker, there is no way to gain this information as the communication is encrypted.

*Replay attack:* In this scheme, a timestamp is utilized to calculate $x = (h((h(PW_i) \oplus ID_i))^e \oplus h(T)) mod\, N$,which is used to construct the login message, which in turn will be different each time. Therefore, the adversary cannot launch a replay attack.

*Man-in-the-Middle attack (MIMA):* In this scheme, since our messages are encrypted and only $OTT_i$ is given in plaintext which is only for one time use. The attacker cannot perform this type of attack, unless he knows $PW_i, ID_i\, and\, OTT_i$ in advance.

*Password-Guessing attack:* In this scheme, to guess the password, the adversary needs to decrypt the login message, which is infeasible it is protected by the private key. The other way that the adversary could try is to compromise the server and get its password database; but this is not an easy task since the database itself is protected by the private key, thus, this type of attack cannot be launched by an attacker.

*Smart-card loss attack:* In case, the smart-card is lost or stolen, the adversary can try to get

$ID_i and OTT_i$ from the smart-card using invasive attacks [48], which is difficult. However, the adversary cannot get the password information. Therefore, he cannot compromise the security.

*Denial of Service (DoS) attack:* The server only requires to check $OTT_i$ to decide if a valid user is trying to authenticate. If $OTT_i$ is not valid, the server can discard the login request without processing the encrypted message, which in turn requires very less computation compared to if it has to perform the hash of symmetric/asymmetric decryption. Even if the attacker makes use of a valid $OTT_i$ by eavesdropping, the server will be able to identify the particular $OTT_i$ used for DoS attack and report the incident to the administrator. The administrator can then set firewall rules to drop the login requests which make use of that $OTT_i$ ; this operation may temporarily disable the user associated with $OTT_i$ , but the administrator can ask the user to manually update $OTT_i$ over the secure channel.

## 4.4 Comparison of Selected RSA Variants w.r.t. to Security Attacks and Selected Security Metrics

In this subsection, we qualitatively compare the proposed RSA-ASH-SC scheme against selected RSA-based variants in terms of selected security attacks. Our findings are captured in Table 4.4.

## 4.5 Comparison of Selected RSA Variants w.r.t Computational Performance

In this subsection, we compare the proposed RSA-ASH-SC scheme against the same selected RSA-based variants (listed in Table 4.4) in terms of computational performance. Our findings

Table 4.4: Comparison of selected RSA variants w.r.t. to security attacks and selected security metrics

| | Yang et al. [39] | Fan et al. [42] | Yang et al. [43] | Om et al. [22] | Om et al. [10] | Shen et al. [44] | Liu et al. [45] | Chien et al. [46] | Proposed RSA |
|---|---|---|---|---|---|---|---|---|---|
| Confidentiality | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Availability | ✓ | ✓ | | | | | | ✓ | ✓ |
| Integrity | | | | | | ✓ | | ✓ | ✓ |
| Mutual authentication | | | | | | ✓ | ✓ | ✓ | ✓ |
| MIMA | | | | ✓ | ✓ | ✓ | ✓ | | ✓ |
| Smart card loss attack | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ |
| Password-guessing attack | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Replay attack | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ |
| Forgery or Impersonation attack | | | ✓ | ✓ | ✓ | | ✓ | | ✓ |
| DoS attack | ✓ | ✓ | | | | | | ✓ | ✓ |
| Forward Secrecy | ✓ | ✓ | | | | ✓ | ✓ | | |

are captured in Table 4.5, where:

- $T_{exp}$ is the time taken by modular exponent operation.
- $T_{mul}$ is the time taken by the modular multiplication operation.
- $T_h$ is the time taken by hash function operation.
- $T_{xor}$ is the time taken by the XOR operation.
- $T_e$ is the time taken by the modular encryption exponent (e) operation.
- $T_d$ is the time taken by the modular decryption exponent (d) operation
- $T_s$ is the time taken to encrypt and decrypt using the symmetric key.

Based on Table 4.5, it is found that Chien et al. [46] scheme yields the less computation time compared to other schemes.

Table 4.5: Comparison of selected RSA variants w.r.t. to computation performance

| | Login Phase | Authentication Phase |
|---|---|---|
| Yang et al. [39] | $2T_{exp} + 3T_{mul} + 1T_h$ | $1T_e + 1T_{exp} + 1T_{mul} + 1T_h$ |
| Fan et al. [42] | $2T_{exp} + 3T_{mul} + 1T_h$ | $1T_e + 1T_{exp} + 1T_{mul} + 1T_h$ |
| Yang et al. [43] | $2T_{exp} + 3T_{mul}$ | $1T_e + 2T_{exp} + 1T_{mul}$ |
| Om et al. [22] | $1T_e + 1T_{exp} + 1T_h + 1T_{xor}$ | $1T_d$ |
| Om et al. [10] | $1T_e + 1T_{exp} + 2T_h + 1T_{xor}$ | $1T_d + 1T_{exp} + 1T_h + 1T_{xor}$ |
| Shen et al. [44] | $1T_e + 2T_{exp} + 3T_{mul} + 2T_h$ | $1T_d + 2T_h + 1T_{xor}$ |
| Liu et al. [45] | $1T_d + 1T_e + 1T_{exp} + 1T_{mul} + 2T_h + 2T_{xor}$ | $1T_e + 2T_h + 1T_{xor} + 3T_{mul} + 2T_{exp}$ |
| Chien et al. [46] | $2T_h + 2T_{xor}$ | $3T_h + 3T_{xor}$ |
| Proposed RSA-ASH-SC scheme | $2T_e + 1T_s + 6T_h + 2T_{xor}$ | $1T_d + 1T_e + 2T_h + 2T_{xor} + 1T_s$ |

## 4.6 Convergence Speed of the Proposed RSA-ASH-SC Scheme

Our proposed RSA-ASH-SC scheme is a RSA-based protocol. As such, it performance in terms of speed of convergence is heavily dependent on that of the considered RSA underlying algorithm. In this subsection, inspired from a study carried in [67], the impact of the performance of the RSA underlying algorithm (including selected RSA variants' performance) on the performance of the proposed RSA-ASH-SC scheme is quantified in terms of speed of convergence.

According to the Crypto++ benchmark [67], the RSA decryption is much slower in performance than the RSA encryption. Indeed, it was reported [24] that with a key length of 2048, the RSA encryption takes only 0.03 milliseconds/operation to complete whereas the

decryption algorithm takes 1.03 milliseconds/operation, on the Fedora Operating system, Release 25 (x86_64), where the host CPU is a 6th generation Skylake, with a frequency 3.14e + 9 Hz. Because of this, in the sequel, when we refer to " *decryption performance*", we mean "*the performance of the RSA algorithm as a whole in terms of how fast it converges*". This convention prevails for all RSA variants referred to in this Subsection, which have been proposed in the literature and validated as improvements to the decryption performance of the RSA algorithm [59], i.e. to speed up the decryption process, namely the Batch RSA scheme [68], the Multi-Prime RSA scheme [71], the Multi-Power RSA scheme [72], Rebalanced RSA scheme [73], the RPrime RSA scheme [49], and a combination of Rebalanced RSA and Multi-Power RSA scheme (here referred to as Rebalanced-Multi-Power RSA scheme [9]).

According to a performance study of these RSA variants using a 1024 bits key size [69], and that considering the cost of single modular exponentiation, the Batch RSA scheme [68] can compute several modular exponentiations effectively. Indeed, it was reported that a batch size of 4 (resp. 8) increases the decryption performance of RSA by a factor of 2.6 (resp. 3.5).

For the Multi-Prime RSA scheme [71], which generates k distinct prime numbers (using the same key size of 1024), it was reported that k should be less or equal to 3 and the decryption process requires k full exponentiations modulo $n/k$ bit numbers to complete. In contrast, running the standard RSA in conjunction with the Chinese Remainder Theorem [59], it was reported [69] that to calculate $x^d mod\, p$ , it takes $O(logd.log^2p)$, and if $d$ is at the same scale of p, the decryption performance of Multi-Prime RSA scheme [71] is $(2.(n/2)^3)/k.(n/k)^3 = k^2/4$ better than that of the standard RSA [59].

For the Multi-Power RSA scheme [72], where the moduli is formed as $N = p^{k-1}.q$ where p and q are two distinct primes of n/k bits each, it was reported [69] that 2 full exponentiations modulo $n/b$ bits are required for the completion of the decryption [69], yielding a decryption performance of $(2.(n/2)^3)/2.(n/k)^3 = k^3/8$ better than that of the standard RSA [59].

For the Rebalanced RSA scheme [73], which creates 2 prime numbers of $n/2$ bits size

each and an input parameter $s \leq n/2$ is used to generate the decryption key, it was reported [69] that the decryption performance is $(n/2)/s = n/2s$ better than that of the standard RSA [59].

For the RPrime scheme [49], which was designed as a combination of the Rebalanced RSA scheme [73] and the Multi-Prime RSA scheme [71], which also generates k distinct prime numbers (using a key size of 1024) as in Multi-Prime RSA scheme [71], it was reported [69] the decryption performance is $(k^2/4).(n/k.s) = n.k/4.s$ better than that of the standard RSA [59].

For the Rebalanced-Multi-Power RSA scheme [9], which was designed as a combination of the Rebalanced RSA [73] and Multi-Power RSA scheme [72], it was reported [69] the decryption performance is $(k^3/8).(n/k.s) = n.k^2/8.s$ better than that of the standard RSA [59].

Table 4.6 highlights the performance comparison of the above RSA variants [ [69] in terms of convergence speed]. It can be observed an increase in k (the number of prime factors used in the RSA algorithm) also yields an increase in the decryption performance of the above discussed RSA variants. In order to estimate the performance of our proposed RSA-ASH-SC

Table 4.6: Decryption performance of selected RSA variants [69]

| **RSA variant** | **Decryption performance** |
|---|---|
| Standard RSA [59] | $x$ |
| Batch RSA [68] | $2.6x(if\ batch\_size = 4); 3.5x(if batch\_size = 8)$ |
| Multi-Prime RSA [71] | $(k^2/4).x$ |
| Multi-Power RSA [72] | $(k^3/8).x$ |
| Rebalanced RSA [73] | $(n/2.s).x$ |
| RPrime RSA [49] | $(n.k/4.s).x$ |
| Rebalanced-Multi-Power RSA [9](used in our RSA-ASH-SC scheme) | $(n.k^2/8.s).x$ |

scheme in terms of convergence speed, we have considered the Rebalanced-Multi-Power RSA scheme [9] as underlying RSA algorithm, using a key length of 1024 bits (as above), but also a key length of 2048 bits (as per the NIST recommendation [70]). Using the results shown in

Table 4.6, the comparison of the RSA variants in term of decryption performance when k=3 are captured in Fig. 4.12. In Fig. 4.12, it can be observed that independently of the RSA key length, our proposed scheme is about 100% faster than the Om and Kumari scheme [10]. For a RSA key length of 2048 bits with k=3, our proposed scheme is about 14.4% faster than any of the other considered RSA variants. For a RSA key length of 1024 bits with k=3, our proposed scheme is about 7.2% more faster than the Om and Kumari scheme [10], which itself is 9.6 time faster than the standard RSA for a key length 2048 (resp. 4.8 times faster for a RSA key length 1028) [9]. Finally, it is observed that independently of the key length, our proposed scheme outperforms the other considered schemes when k=3. Hence, we conclude that our proposed scheme yields a better decryption performance compared to the other considered schemes.

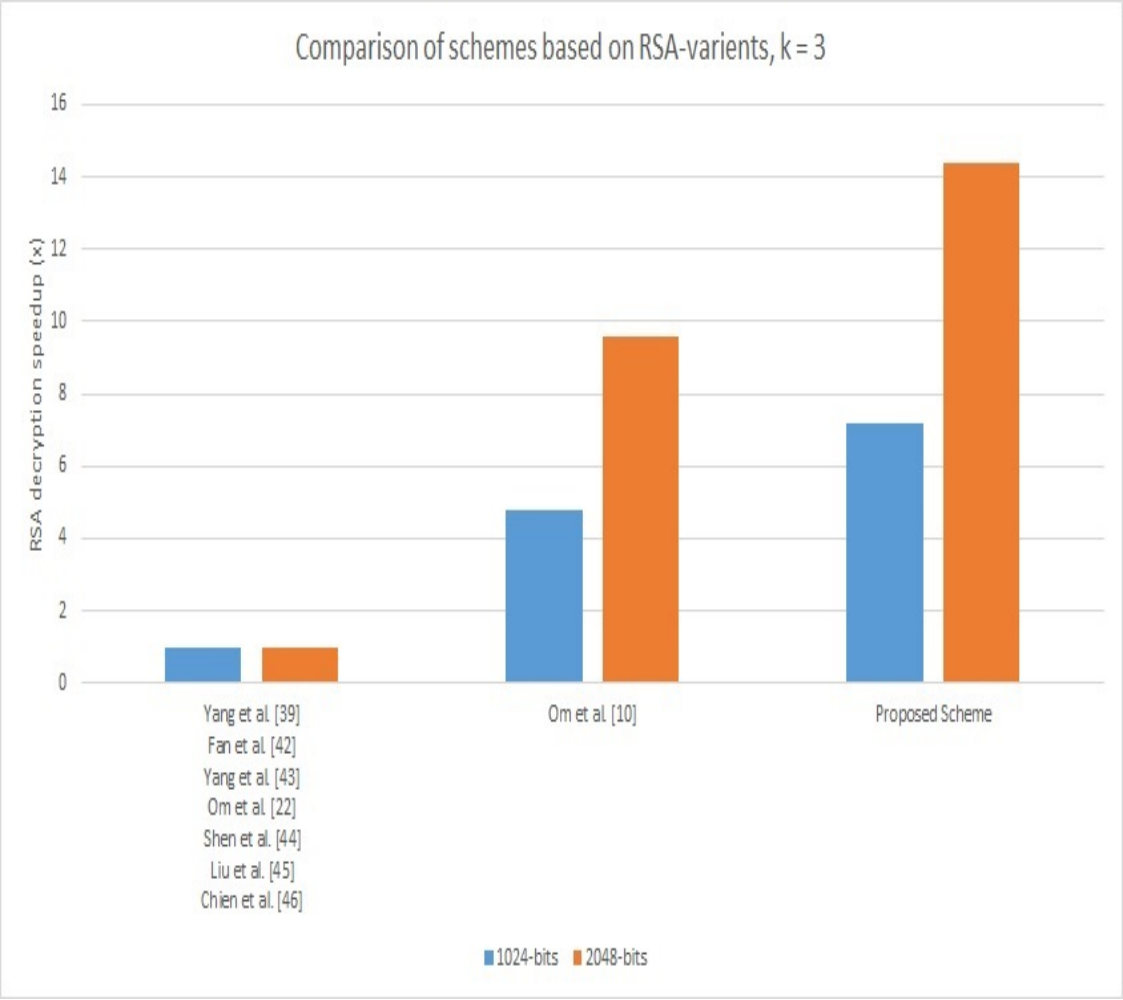


Figure 4.12: Convergence speed of selected RSA variants for key length 1024 bits vs. 2048 bits when k=3.

# Chapter 5

# Conclusion

## 5.1 Contributions of the Thesis

In this thesis, we have proposed the design of four authentication schemes for smart home networks:

- A two-factor device-to-device (D2D) Mutual Authentication Scheme for Smart Homes using OTP over Infrared Channel (so-called D2DA-OTP-IC scheme) and of proof-of-concept of this scheme iin the form of a hardware solution.
- Two enhanced versions (proxy-password of the D2DA-OTP-IC scheme (the so-called Password Protected Inter-device OTP-based Authentication scheme over Infrared Channel (PPIDA-IC) and Password Protected Inter-device OTP-based Authentication scheme using public key infrastructure (PPIDA-PKI scheme).
- A RSA-based two-factor user Authentication scheme for Smart Home using Smart Card (so-called RSA-ASH-SC scheme).

The proposed D2DA-OTP-IC, PPIDA-IC and PPIDA-PKI schemes have been modelled using HLPSL and these models have been validated using the SPAN/AVISPA tool. These protocols have also been verified using the SPAN/AVISPA tool for their efficiency in terms of achieving the goals of secrecy of secret keys and D2D mutual authentication. An informal

security analysis of the proposed RSA-ASH-SC scheme has also been conducted; as well as its performance with respect to the convergence speed. For this performance study, we have considered the Rebalanced-Multi-Power RSA scheme [9] as underlying RSA algorithm of our RSA-ASH-SC scheme, using respectively a key length of 1024 bits and 2048 bits, the results have revealed that: (1) Using a key length of 1028 (resp. 2048) with k=3, where k is the number of prime factors used in the RSA scheme, our proposed RSA-ASH-SC scheme is about 50% faster than the Om and Kumari scheme [10]; (2) Using a key length of 2048 bits with k=3, our proposed RSA-ASH-SC scheme is about 15% faster than any of the other considered RSA variants; (3) Using a key length of 1024 bits with k=3, our proposed RSA-ASH-SC scheme is about 7.2% faster than the Om and Kumari scheme [10].

Next, we have also compare selected RSA variants with respect to few security attacks and selected security metrics. Except for the Forward Secrecy attack, which our RSA-ASH-SC scheme may not prevent, all other considered attacks could be prevented based on the design features.

## 5.2 Future Work

### 5.2.1 With respect to the proposed D2DA-OTP- IC protocol

a) *Adopt light communication for smart homes:* Light communication has specific properties which are undetectable for human eyes and cannot cross the obstacles. In the D2DA-OTP-IC scheme [7], the OTPS are shared over the infrared channel with the home devices, which is more secured than radio channels (e.g. Wi-Fi). However, the DH key exchange and normal communication is over a radio channel. The vulnerabilities in the software may lead to security compromise. Therefore, smart home communication should be completely based on light communication, so that there is no room for attacks from outside the home. Examples of light communication which

can be used include Visible light communication (VLC) [15], Dark light communication (DLC) [16] and Infrared light communication (ILC) [7]. These variations of light communication have different properties, which needs more exploration for the smart home applications. The problem with VLC is that all the time the light should be in ON state, which may be irritating for the inhabitants [15]. Similarly, DLC is not suitable when inhabitants need light in their homes [16]. Mulvey et al. [74] have discussed the potential hazards of over exposure to infrared light. Therefore, each of these communication mediums has its own weaknesses and strengths, and more research is required to be able to switch completely from radio-based communication to light-based communication.

b) *Supporting dynamic movements of home devices:* In the proposed D2DA-OTP-IC scheme [7], in the registration phase, the OTP-server saves the location of the home device and this location is fixed. However, some devices may need movement within a home, therefore, the OTP-server should support dynamic movement of these kind of devices. As of now, no such support is provided. To support dynamic movements of a device, this device should be able to localize itself and communicate its coordinates to the OTP-server. Another possible solution is that the OTP-server should be able to find the devices in the home by tracking. The localization issue in smart home needs to addressed.

c) *Protecting the private key within the device:* In the proposed D2DA-OTP-IC scheme [7], the DH key exchange is supported by PKI and the private keys are not protected within the system. According to FIPS 4.7.5 standard [8], the secret information such as secret key and private keys needs protection. Protecting the private key and secret key in D2D communication is a challenge, as someone/something should protect the password, if the secret/private key is encrypted by a password and it this password is saved within the device, it does not solve the purpose since the password is the

sensitive information and according to FIPS 4.7.5 standard [8], it shouldnt be saved in the plaintext format. In our literature review, we could not find any solution to protect the keys with password. A novel solution is required where human intervention is not required. By protecting the private and secret keys, the FIPS 4.7.5 standard requirement of not storing the sensitive information can be achieved.

### 5.2.2 With respect to the proposed PPIDA-IC and PPIDA-PKI protocols

d) *Reducing the number of keys:* In the proposed password-proxy schemes (PPIDA-IC and PPIDA-PKI), the number of keys used increases as more and more devices are registered to the system. Management of these keys becomes a cumbersome process and subject to human error. Registering a new device and updating the keys is very time- consuming process. It would be helpful if the number of keys are reduced. A centralized solution is vital, for instance, using a certificate authority-based or a Kerberos-based protocol for key management.

e) *Automatic registration:* In the proposed PPIDA-IC and PPIDA-PKI schemes, the registration is manual and consumes some time as someone needs to configure the keys in the device and update the OTP-server manually. If the inhabitants do not have the security knowledge, they would need technical assistance, which may have some associated cost. Therefore, the automation of the registration process will save time and money to the inhabitants. Automatic registration process (compromising the security is the system) would ease the adoption of the proposed schemes.

f) *Supporting the dynamic movements of home devices:* The proposed PPIDA-IC and PPIDA-PKI schemes does not support the movement of home devices within the smart home. As discussed earlier, dynamic movements are desirable in smart home environment. This requires that a suitable localization solution be developed in the context

of smart home.

g) *Completely light-based communication in smart homes:* In the proposed PPIDA-IC and PPIDA-PKI schemes, the OTPs are sent over the infrared channel. Thus, these schemes should implement the light-based communication. PureLiFi [15] is one of the leading company, which is implementing the VLC-based solutions, but as discussed earlier, this technology is still in its infancy.

### 5.2.3 With respect to the proposed RSA-ASH-SC scheme protocol

h) *Private Key for each smart card:* In the proposed RSA-ASH-SC scheme, the authentication is based on password and smart card. The smart card only stores some secret information and public key of the server. In this scheme, the session keys are created by a client, and then encrypted and sent to the server over a network. If the RSA keys are compromised, the scheme will become vulnerable to forwards secrecy attack. To protect against this attack, the session keys should be exchanged in a secure manner, for instance, using the DH key exchange protocol [51]. In this way, even if the attacker records the packets and get access to the RSA keys, he cannot compromise the security of the past communication. To achieve the DH key exchange, the smart card also need its private key and the server should be able to recognize this key, thus, involving a certificate authority in this scenario might be useful.

i) *Biometric support:* It is well-known that passwords can be compromised by various hacking techniques such as shoulder surfing, social engineering, key loggers, certificate frauds, etc. Therefore, biometric might be a good alternative for authentication. Moreover, some users prefer biometric-based authentication. The proposed RSA-ASH-SC approach can be re-designed to rely on biometric rather than smart card. It can also be adjusted to rely on the three factors: smart card, password, and biometric. The tight

coupling of these three factors will add more security layers; yielding a much stronger authentication mechanism for smart home networks.

# Appendix A

# HLPSL code for D2DA-OTP- IC

(Only the roles of each entity in the model and the knowledge given to the intruder are shown)

**The client (D1) implementation in HLPSL: Here role role_A is created for device D1.**

```
// ==========HLPSL code for D2DA-OTP- IC Scheme========== //
role role_A(A:agent,
    B:agent,
    KPa:public_key,
    KPb:public_key,
    G:text,
    P:text,
    SND,RCV:channel(dy))
played_by A
def=
    local
        State:nat,
        Na:text,
```

```
    Timestamp:text,
    Timestamp2:text,
    Mod:function,
    Nb:text,
    OTP2:text,
    OTP1:text,
    Hash:function,
    AA,BB:message,
    Key:message
const
    mod:     function,
    client_server_aa: protocol_id,
    server_client_bb: protocol_id
init
    State := 0
transition
    1. State=0 /\ RCV(start) =|>
        State':=1 /\ Na':=new()
        /\ Timestamp':=new()
        /\ AA' := mod(exp(G,Na'),P)
        /\ SND(A.{A.Timestamp'.AA'}_inv(KPa))

    2. State=1 /\ RCV(B.{B.Timestamp.BB'}_inv(KPb)) =|>
        State':=2 /\ Key' := mod(exp(BB',Na'),P)
        /\ secret(Key',secret_key, {A,B})
    5. State=2 /\ RCV({Timestamp2'.OTP2'.OTP1'}_KPa) =|>
        State':=3 /\ SND({Hash(OTP2')}_Key)
    7. State=3 /\ RCV({Hash(OTP1')}_Key) =|>
        State':=4
```

```
        /\ witness(A,B,server_client_bb,{B.Timestamp'.BB'}_inv(KPb)
                .Hash(OTP1'))
        /\ request(A,B,client_server_aa,{A.Timestamp'.AA'}_inv(KPa)
                .Hash(OTP2'))
end role
Device D2 s role in HLPSL:
role role_B(A:agent,
    B:agent,
    KPa:public_key,
    KPb:public_key,
    G:text,
    P:text,
    SND,RCV:channel(dy))
played_by B
def=
    local
        State:nat,
        Na:text,
        Mod:function,
        Nb:text,
        Timestamp:text,
        Timestamp2:text,
        Req:text,
        OTP2:text,
        OTP1:text,
        Hash:function,
        AA,BB:message,
        Key:message
    const
```

```
        mod:     function,
        client_server_aa: protocol_id,
        server_client_bb: protocol_id
    init
        State := 0
    transition
        1. State=0 /\ RCV(A.{A.Timestamp'.AA'}_inv(KPa)) =|>
            State':=1 /\ Nb':=new()
            /\ BB' := mod(exp(G,Nb'),P)
            /\ SND(B.{B.Timestamp'.BB'}_inv(KPb))
            /\ Key' := mod(exp(AA',Nb'),P)
            /\ secret(Key',secret_key, {A,B})
            /\ Req':=new()
            /\ SND(B.{A.B.Timestamp'.Req'}_inv(KPb))

        4. State=1 /\ RCV({Timestamp2'.OTP1'.OTP2'}_KPb) =|>
            State':=2
        6. State=2 /\ RCV({Hash(OTP2)}_Key') =|>
            State':=3 /\ SND({Hash(OTP1)}_Key')
        /\ witness(B,A,client_server_aa,{A.Timestamp'.AA'}_inv(KPa)
                .Hash(OTP2'))
        /\ request(B,A,server_client_bb,{B.Timestamp'.BB'}_inv(KPb)
                .Hash(OTP1'))
end role
```

OTP–server role in HLPSL:

```
role role_O(O:agent,
    A:agent,
    B:agent,
```

```
    KPa: public_key ,
    KPb: public_key ,
    KPo: public_key ,
    OTP1: text ,
    OTP2: text ,
    SND,RCV: channel(dy))
played_by O
def=
    local
        State: nat ,
        Timestamp: text ,
        Req: text
    init
        State := 0
    transition
        3. State=0 /\ RCV(B.{A.B.Timestamp'.Req'}_inv(KPb)) =|>
            State':=1 /\ Timestamp' := new()
            /\ SND({Timestamp'.OTP1.OTP2}_KPb)
            /\ SND({Timestamp'.OTP2.OTP1}_KPa)
            /\ secret(OTP1, server_otp1 , {A,B,O})
            /\ secret(OTP2, server_otp2 , {A,B,O})
end role
```

In HLPSL, the following knowledge was given to the intruder.

```
intruder_knowledge = {p,g,otpserver ,client ,server ,ka,kb,ki,inv(ki)}
```

Goal setting for the protocol:

```
goal
    secrecy_of server_otp1
```

```
    secrecy_of server_otp2
    secrecy_of secret_key
    authentication_on client_server_key
    authentication_on server_client_key
end goal


OTP server actions


from socket import SOCK_STREAM, AF_INET
import socket
import random
import os
import time
REGISTERED_LOCATION_MAP = {"10.42.0.45":["180","75"],
                           "10.42.0.87":["90","100"]
                           }
def decrypt_msg(enc_msg):
    try:
        with open("msg.enc","w") as fd:
            fd.write(enc_msg)
        os.system("/usr/bin/openssl rsautl -in msg.enc -out msg.dec
        -inkey /home/pi/.ssh/key.pem -decrypt")
        with open("msg.dec","r") as fd:
            return fd.read()
    except:
        return False

    return False
```

```
def generate_OTP():
    OTP = ""
    for i in range(8):
        OTP += str(random.randint(0,9) )
    print "OTP",OTP
    return OTP

def point_system(system):
    os.system("sudo python servo_controller.py "
    +REGISTERED_LOCATION_MAP[system][0] + " "
    + REGISTERED_LOCATION_MAP[system][1])

def find_system_and_send_OTP(system, OTP1, OTP2):
    if not system in REGISTERED_LOCATION_MAP:
        print system, "is not registered!"
    time.sleep(1)
    point_system(system)
    time.sleep(1)
    send_OTP(OTP1)
    send_OTP(OTP2)
    time.sleep(3)

def send_OTP(OTP):
    key_map = {"0":"KEY_0",
               "1":"KEY_1",
               "2":"KEY_2",
               "3":"KEY_3",
               "4":"KEY_4",
               "5":"KEY_5",
```

```
                "6":"KEY_6",
                "7":"KEY_7",
                "8":"KEY_8",
                "9":"KEY_9"}
    for i in OTP:
        if i not in key_map:
            print  "OTP should have only numeric keys"
            return False
    os.system("irsend send_once HP_RC2234302/01B KEY_GREEN")
    for i in OTP:
        os.system("irsend send_once HP_RC2234302/01B "+key_map[i])
        time.sleep(.1)
    os.system("irsend send_once HP_RC2234302/01B KEY_RED")
    return True
def process_msg(msg,client):
    #print "Message received :",msg
    if msg.startswith("OTP_AUTH_REQUEST"):
        msg = msg.split()
        if len(msg) != 2:
            print "Wrong OTP_AUTH_REQUEST format!"
            return False
        OTP1 = generate_OTP()
        OTP2 = generate_OTP()
        find_system_and_send_OTP(client, OTP1, OTP2)
        find_system_and_send_OTP(msg[1], OTP2, OTP1)
        return True
    else:
        print "Don't know what to do!"
```

```
s=socket.socket(AF_INET, SOCK_STREAM)
s.bind(('10.42.0.11',54545))
s.listen(1)
try:
    while True:
        print "listening"
        connection, client_address = s.accept()
        print "#"*50
        print "Remote Host :",client_address[0]
        rec = connection.recv(1024)
        process_msg(decrypt_msg(rec),client_address[0])
        print "#"*50
except:
    print "Exception"
s.close()
```

Login program

```
import sys,syslog,os
from pexpect import pxssh
import pexpect
import time
import random
import hashlib
from socket import AF_INET,SOCK_STREAM,SOL_SOCKET,SO_REUSEADDR,SO_BROADCAST
import socket

GATEWAY_ADDR = '10.42.0.11'
```

```
PORT = 54545
def encrypt_with_publickey(msg):
    try:
        with open("msg.msg","w") as fd:
            fd.write(msg)
        os.system("/usr/bin/openssl rsautl -in msg.msg -out msg.enc
        -pubin -inkey ~/.ssh/key.pub -encrypt")
        with open("msg.enc","r") as fd:
            return fd.read()
    except:
        return False

    return False
def send_msg_to_gateway(msg):
    try:
        soc = socket.socket(AF_INET, SOCK_STREAM)
        soc.connect((GATEWAY_ADDR, PORT))
        msg_enc = encrypt_with_publickey(msg)
        soc.send(msg_enc)
    except:
        return False
    return True

def run_cmd(client):
    cmd = raw_input("")
    client.sendline(cmd)
    client.prompt()
    output = client.before[len(cmd):-1]
    print(output.rstrip())
```

```
'''
def generate_OTP():
    OTP = ""
    for i in range(8):
        OTP += str(random.randint(0,9) )
    return OTP
'''
def read_receiver_OTPs():
    print "reading OTP"
    key_map = {"KEY_0":"0",
               "KEY_1":"1",
               "KEY_2":"2",
               "KEY_3":"3",
               "KEY_4":"4",
               "KEY_5":"5",
               "KEY_6":"6",
               "KEY_7":"7",
               "KEY_8":"8",
               "KEY_9":"9"}
    file_otp = "/tmp/codes.data"
    for i in range (60):
        time.sleep(1)
        print "trying :"+str(i)
     sys.stdout.flush()
        try:
            with open(file_otp) as fd:

                data = fd.readlines()
```

```
    if len(data) < 10:
        continue
    else:
        data = [lin.split()[3] for lin in data]
        if data[0] == "KEY_GREEN" and data[11] == "KEY_GREEN":
            pass
        else:
            print "Wrong starting signal!"
            os.system("sudo pkill -f irw")
    os.system("rm /tmp/codes.data")
            return False
        if data[10] == "KEY_RED" and data[19] == "KEY_RED":
            pass
        else:
            print "Wrong ending signal!"
            os.system("sudo pkill -f irw")
os.system("rm /tmp/codes.data")
            return False
        OTP1 = ""
        for i in range(1,9):
            if data[i] in key_map:
                OTP1 += key_map[data[i]]
            else:
                print "OTP should have only numeric keys!"
                os.system("sudo pkill -f irw")
    os.system("rm /tmp/codes.data")
                return False
        OTP2 = ""
        for i in range(11,19):
```

```
                        if data[i] in key_map:
                            OTP2 += key_map[data[i]]
                        else:
                            print  "OTP should have only numeric keys!"
                            os.system("sudo pkill -f irw")
                    os.system("rm /tmp/codes.data")
                            return False
                        os.system("sudo pkill -f irw")
                os.system("rm /tmp/codes.data")
                        return [OTP1,OTP2]
            except:
                print "error occured while reading!"
    os.system("sudo pkill -f irw")
    os.system("rm /tmp/codes.data")
    return False


'''
def send_OTP(OTP):
    key_map = {"0":"KEY_0",
               "1":"KEY_1",
               "2":"KEY_2",
               "3":"KEY_3",
               "4":"KEY_4",
               "5":"KEY_5",
               "6":"KEY_6",
               "7":"KEY_7",
               "8":"KEY_8",
               "9":"KEY_9"}
    for i in OTP:
```

```
        if i not in key_map:
            print "OTP should have only numeric keys"
            return False
    os.system("irsend send_once HP_RC2234302/01B KEY_GREEN")
    for i in OTP:
        os.system("irsend send_once HP_RC2234302/01B "+key_map[i])
        time.sleep(.1)
    os.system("irsend send_once HP_RC2234302/01B KEY_RED")
    print "OTP sent"
    return True
'''
def main():
    #start IR receiver
    os.system("rm /tmp/codes.data")
    os.system("irw > /tmp/codes.data &")
    #send auth information to gateway
    #send_msg_to_gateway("CONNECTING "+sys.argv[3])
    #connect using ssh
    ssh_client = pxssh.pxssh()
    #print "Calling login"
    #sys.stdout.flush()
    ssh_client.PROMPT = "^Authenticated\W"
    #print(sys.argv)
    print ssh_client.login(sys.argv[3], sys.argv[1],
    auto_prompt_reset=False,original_prompt=pexpect.EOF)
    print(ssh_client.before)
    #print "Returned from login"
    #sys.stdout.flush()
    ssh_client.PROMPT = "^Received\W[\:]"
```

```
    ssh_client.prompt()
    print(ssh_client.before)
    sys.stdout.flush()
    #check OTPs
    received_OTPs = read_receiver_OTPs()
    if received_OTPs == False:
        print "Didn't received the OTP from Server"
        sys.exit()
    print "OTP Received :",received_OTPs
    sys.stdout.flush()
    st = ssh_client.before.split()
    print "Hash received =>",st[1]
    sys.stdout.flush()
    if hashlib.sha256(received_OTPs[0]).hexdigest() != st[1]:
        print "Received Hash doen't match!"
        sys.exit()
    #ssh_client.PROMPT = "^Please enter OTP's hash\W[\:]"
    #ssh_client.prompt()
    ssh_client.sendline(hashlib.sha256(received_OTPs[1]).hexdigest())
    ssh_client.PROMPT = "[\$\#]"
    ssh_client.prompt()
    #time.sleep(5)
    #print(ssh_client.before)
    while True:
        run_cmd(ssh_client)
if __name__ == "__main__":
    if len(sys.argv) != 3:
        print "Usage : OTP_LOGIN.py <username> <remote_server>"
        sys.exit()
```

```
try:
 main()
except:
 os.system("rm /tmp/codes.data")
 // ================================================== //
```

# Appendix B

# HLPSL code for PPIDA-IC

(Only the roles of each entity in the model and the knowledge given to the intruder are shown)

```
// ============HLPSL code for PPIDA-IC ================== //
D1 role in HLPSL:
role role_A(A:agent,
    B:agent,
    O:agent,
    KPa:public_key,
    KPb:public_key,
    KPo:public_key,
    G:text,
    P:text,
    Token:text,
    SND,RCV:channel(dy))
played_by A
def=
    local
        State:nat,
```

```
    Na: text ,
    Timestamp: text ,
    Timestamp2: text ,
    Mod: function ,
    Nb: text ,
    OTP2: text ,
    OTP1: text ,
    Hash: function ,
    AA,BB,OO: message ,
    Key: message ,
    Password: text ,
    Session_key: message
const
    mod:     function ,
    client_server_aa: protocol_id ,
    server_client_bb: protocol_id
init
    State := 0
transition
    1. State=0 /\ RCV(start) =|> State':=1
        /\ Timestamp':=new()
        /\ SND(A.{Token.Timestamp'.Hash(Token.Timestamp')}_KPo)
    2. State=1
    /\ RCV(O.{Timestamp'.Password'.Hash(Timestamp'.Password')
            }_inv(KPo))
    =|> State':=2
        /\ secret(Password',sec_1,{A,O})
        % countinue old protocol
        /\ Na':=new()
```

```
        /\ Timestamp':=new()
        /\ AA' := mod(exp(G,Na'),P)
    /\ SND(A.{A.Timestamp'.AA'.G.P.Hash(AA'.G.P.Timestamp'.A)
            }_inv(KPa))

    2. State=2
    /\ RCV(B.{B.Timestamp.BB'.Hash(BB'.Timestamp'.B)}_inv(KPb)) =|>
        State':=3 /\ Key' := mod(exp(BB',Na'),P)
        /\ secret(Key',secret_key, {A,B})
    5. State=3 /\ RCV({Timestamp2'.OTP2'.OTP1'}_KPa) =|>
        State':=4 /\ Key' := xor(Key',Hash(xor(OTP1',OTP2')))
        /\ SND({Hash(OTP2')}_Key')
    7. State=4 /\ RCV({Hash(OTP1')}_Key') =|>
        State':=5
    /\ witness(A,B,server_client_bb,{B.Timestamp'.BB'}_inv(KPb)
            .Hash(OTP1'))
    /\ request(A,B,client_server_aa,{A.Timestamp'.AA'}_inv(KPa)
            .Hash(OTP2'))
end role
D2 role in HLPSL:
role role_B(A:agent,
    B:agent,
    KPa:public_key,
    KPb:public_key,
    G:text,
    P:text,
    SND,RCV:channel(dy))
played_by B
def=
```

```
local
    State:nat,
    Na:text,
    Mod:function,
    Nb:text,
    Timestamp:text,
    Timestamp2:text,
    Req:text,
    OTP2:text,
    OTP1:text,
    Hash:function,
    AA,BB:message,
    Key:message
const
    mod:     function,
    client_server_aa: protocol_id,
    server_client_bb: protocol_id
init
    State := 0
transition
    1. State=0
    /\ RCV(A.{A.Timestamp'.AA'.G.P.Hash(AA'.G.P.Timestamp'.A)
            }_inv(KPa))
    =|> State':=1 /\ Nb':=new()
        /\ BB' := mod(exp(G,Nb'),P)
        /\ SND(B.{B.Timestamp'.BB'.Hash(BB'.Timestamp'.B)
            }_inv(KPb))
        /\ Key' := mod(exp(AA',Nb'),P)
        /\ secret(Key',secret_key, {A,B})
```

```
        /\ Req':=new()
        /\ SND(B.{A.B.Timestamp'.Req'.Hash(A.B.Timestamp')
            }_inv(KPb))

    4. State=1 /\ RCV({Timestamp2'.OTP1'.OTP2'}_KPb) =|>
        State':=2 /\ Key' := xor(Key',Hash(xor(OTP1',OTP2')))
    6. State=2 /\ RCV({Hash(OTP2)}_Key') =|>
        State':=3 /\ SND({Hash(OTP1)}_Key')
    /\ witness(B,A,client_server_aa,{A.Timestamp'.AA'}_inv(KPa)
            .Hash(OTP2'))
    /\ request(B,A,server_client_bb,{B.Timestamp'.BB'}_inv(KPb)
            .Hash(OTP1'))
end role
OTP-server role in HLPSL:
role role_O(O:agent,
    A:agent,
    B:agent,
    KPa:public_key,
    KPb:public_key,
    KPo:public_key,
    OTP1:text,
    OTP2:text,
    Token:text,
    Password:text,
    SND,RCV:channel(dy))
played_by O
def=
    local
        State:nat,
```

```
        Timestamp:text,
        Req:text,
        No:text,
        OO,AA:message,
        Hash: function,
        Session_key:message
    init
        State := 0
    transition
        3. State=0
        /\ RCV(A.{Token.Timestamp'.Hash(Token.Timestamp')
                        }_KPo) =|> State':=1
            /\ Timestamp' := new()
            /\ SND(O.{Timestamp'.Password.Hash(Timestamp'.Password)
                }_inv(KPo))
            /\ secret(Password,sec_1,{A,O})

        3. State=1
        /\ RCV(B.{A.B.Timestamp'.Req'.Hash(A.B.Timestamp')
                }_inv(KPb)) =|>
        State':=3 /\ Timestamp' := new()
        /\ SND({Timestamp'.OTP1.OTP2}_KPb)
        /\ SND({Timestamp'.OTP2.OTP1}_KPa)
        /\ secret(OTP1, server_otp1, {A,B,O})
        /\ secret(OTP2, server_otp2, {A,B,O})
end role

Intruder s knowledge:
    intruder_knowledge = {p,g,otpserver,client,server,ka,kb,ki,inv(ki)}
```

Goal for the protocol in HLPSL:

```
goal
    secrecy_of server_otp1
    secrecy_of server_otp2
    secrecy_of secret_key
    secrecy_of sec_1
    authentication_on client_server_key
    authentication_on server_client_key
end goal

    // ================================================== //
```

# Appendix C

# HLPSL code for PPIDA-PKI

(Only the roles of each entity in the model and the knowledge given to the intruder are shown)

```
// ==========HLPSL code for PPIDA-PKI========== //
D1 role in HLPSL:
role role_A(A:agent,
    B:agent,
    O:agent,
    KPa:public_key,
    KPb:public_key,
    KPo:public_key,
    KPHa:public_key,
    KPHo:public_key,
    G1,G2:text,
    P1,P2:text,
    Token:text,
    SND,RCV:channel(dy))

played_by A
```

```
def=
    local
        State:nat,
        Na:text,
        Timestamp:text,
        Timestamp2:text,
        Mod:function,
        Nb:text,
        OTP2:text,
        OTP1:text,
        Hash:function,
        AA,BB,OO:message,
        Key:message,
        Password:text,
        Session_key:message
    const
        mod:     function,
        client_server_aa: protocol_id,
        server_client_bb: protocol_id
    init
        State := 0
    transition
        1. State=0 /\ RCV(start) =|> State':=1
            /\ Timestamp':=new()
            /\ SND(A.{Token.Timestamp'.Hash(Token.Timestamp')}_KPo)
        2. State=1
        /\ RCV(O.{O.Timestamp.OO'.G2.P2.Hash(O.Timestamp.OO'.G2.P2)
        }_inv(KPHo))   =|> State':=2
            /\ Na':=new()
```

```
    /\ AA' := mod (exp(G2,Na'),P2)
    /\ Session_key' := mod (exp(OO',Na'),P2)
    /\ SND(A.{A.Timestamp.AA'.Hash(A.Timestamp.AA')}_inv(KPHa))
    /\ secret(Session_key',secret_key1, {A,O})

4. State=2
/\ RCV(O.{{O.Timestamp'.Password'.KPo
        .Hash(O.Timestamp'.Password'.KPo)}_inv(KPo)}_Session_key')
        =|> State':=3
    /\ secret(Password',sec_1,{A,O})
    % countinue old protocol
    /\ Na':=new()
    /\ Timestamp':=new()
    /\ AA' := mod(exp(G1,Na'),P1)
    /\ SND(A.{A.Timestamp'.AA'.G1.P1.Hash(AA'.G1.P1.Timestamp'.A)
    }_inv(KPa))

2. State=3 /\ RCV(B.{B.Timestamp.BB'.Hash(BB'.Timestamp'.B)
        }_inv(KPb)) =|>
    State':=4 /\ Key' := mod(exp(BB',Na'),P1)
    /\ secret(Key',secret_key, {A,B})
5. State=4 /\ RCV({Timestamp2'.OTP2'.OTP1'}_KPa) =|>
    State':=5 /\ Key' := xor(Key',Hash(xor(OTP1',OTP2')))
    /\ SND({Hash(OTP2')}_Key')
7. State=5 /\ RCV({Hash(OTP1')}_Key') =|>
    State':=6
/\ witness(A,B,server_client_bb,{B.Timestamp'.BB'}_inv(KPb)
        .Hash(OTP1'))
/\ request(A,B,client_server_aa,{A.Timestamp'.AA'}_inv(KPa)
```

```
                    .Hash(OTP2'))
end role

D2 role in HLPSL:
role role_B(A:agent,
    B:agent,
    KPa:public_key,
    KPb:public_key,
    G1:text,
    P1:text,
    SND,RCV:channel(dy))
played_by B
def=
    local
        State:nat,
        Na:text,
        Mod:function,
        Nb:text,
        Timestamp:text,
        Timestamp2:text,
        Req:text,
        OTP2:text,
        OTP1:text,
        Hash:function,
        AA,BB:message,
        Key:message
    const
        mod:     function,
        client_server_aa: protocol_id,
```

```
        server_client_bb: protocol_id
    init
        State := 0
    transition
        1. State=0
        /\ RCV(A.{A.Timestamp'.AA'.G1.P1.Hash(AA'.G1.P1.Timestamp'.A)
        }_inv(KPa)) =|>
            State':=1 /\ Nb':=new()
            /\ BB' := mod(exp(G1,Nb'),P1)
            /\ SND(B.{B.Timestamp'.BB'.Hash(BB'.Timestamp'.B)
                }_inv(KPb))
            /\ Key' := mod(exp(AA',Nb'),P1)
            /\ secret(Key',secret_key, {A,B})
            /\ Req':=new()
            /\ SND(B.{A.B.Timestamp'.Req'.Hash(A.B.Timestamp')
                }_inv(KPb))
        4. State=1 /\ RCV({Timestamp2'.OTP1'.OTP2'}_KPb) =|>
            State':=2 /\ Key' := xor(Key',Hash(xor(OTP1',OTP2')))
        6. State=2 /\ RCV({Hash(OTP2)}_Key') =|>
            State':=3 /\ SND({Hash(OTP1)}_Key')
        /\ witness(B,A,client_server_aa,{A.Timestamp'.AA'}_inv(KPa)
                .Hash(OTP2'))
        /\ request(B,A,server_client_bb,{B.Timestamp'.BB'}_inv(KPb)
                .Hash(OTP1'))
end role
```

OTP–server role in HLPSL:

```
role role_O(O:agent,
    A:agent,
```

```
    B: agent ,
    KPa: public_key ,
    KPb: public_key ,
    KPo: public_key ,
    KPHa: public_key ,
    KPHo: public_key ,
    OTP1: text ,
    OTP2: text ,
    G2: text ,
    P2: text ,
    Token: text ,
    Password: text ,
    SND,RCV: channel(dy))
played_by O
def=
    local
        State: nat ,
        Timestamp: text ,
        Req: text ,
        No: text ,
        OO,AA: message ,
        Hash: function ,
        Session_key: message
    init
        State := 0
    transition
        3. State=0
        /\ RCV(A.{Token.Timestamp'.Hash(Token.Timestamp')}_KPo)
         =|> State':=1 /\ No':=new()
```

```
            /\ OO' := mod(exp(G2,No'),P2)
    /\ SND(O.{O.Timestamp'.OO'.G2.P2.Hash(O.Timestamp'.OO'.G2.P2)
    }_inv(KPHo))
        3. State=1 /\ RCV(A.{A.Timestamp.AA'.Hash(A.Timestamp.AA')
        }_inv(KPHa)) =|> State':=2
        /\ secret(Password,sec_1,{A,O})
        /\ Timestamp':=new()
        /\ Session_key' := mod(exp(AA',No'),P2)
        /\ SND(O.{{O.Timestamp'.Password.KPo
                .Hash(O.Timestamp'.Password.KPo)}_inv(KPo)}_Session_key')
        /\ secret(Session_key',secret_key1, {A,O})
        3. State=2
        /\ RCV(B.{A.B.Timestamp'.Req'.Hash(A.B.Timestamp')
        }_inv(KPb)) =|>
            State':=3 /\ Timestamp' := new()
            /\ SND({Timestamp'.OTP1.OTP2}_KPb)
            /\ SND({Timestamp'.OTP2.OTP1}_KPa)
            /\ secret(OTP1, server_otp1, {A,B,O})
            /\ secret(OTP2, server_otp2, {A,B,O})
end role

Intruder s knowledge:
intruder_knowledge = {p1,p2,g1,g2,otpserver,client,server,ka,kb,ki,inv(ki)}
Goal set for the protocol:
goal
    secrecy_of server_otp1
    secrecy_of server_otp2
    secrecy_of secret_key
    secrecy_of secret_key1
```

```
    secrecy_of sec_1
    authentication_on client_server_key
    authentication_on server_client_key
end goal


//=========================================================== //
```

# Appendix D

# Servo controller Code

```
 // ==================Servo controller Code================== //
import RPi.GPIO as GPIO
import time
import sys

if len(sys.argv) != 3:
    print "Usage: servo_controller.py
    <servo1_position[0-180]> <servo2_position[0-180]>"
    sys.exit()
if int(sys.argv[1]) < 10 or int(sys.argv[1]) > 180:
        print "Usage: servo_controller.py
        <servo1_position[0-180]> <servo2_position[0-180]>"
        sys.exit()
if int(sys.argv[3]) < 10 or int(sys.argv[3]) > 180:
        print "Usage: servo_controller.py
        <servo1_position[0-180]> <servo2_position[0-180]>"
        sys.exit()
GPIO.setmode(GPIO.BOARD)
```

```
GPIO.setup(12, GPIO.OUT)
GPIO.setup(7, GPIO.OUT)
try:
    p_1 = GPIO.PWM(12,50)
    p_2 = GPIO.PWM(7,50)
    x_1 = 1.0/18.0 *int(sys.argv[1]) +2
    print "Setting servo_1 "
    p_1.start(x_1)
    time.sleep(2)
    x_2 = 1.0/18.0 *int(sys.argv[3]) +2
    print "Setting servo_2"
    p_2.start(x_2)
    time.sleep(2)
except:
    print "Exception occurred"
p_1.stop()
p_2.stop()
GPIO.cleanup()
// ================================================== //
```

# Bibliography

[1] M. H. Miraz, M. Ali, P. S. Excell, R. Picking, A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT), Proc. of IEEE Internet Technologies and Applications (ITA), Glyndwr University, Wrexham, North East Wales, UK, Sept. 8-11, 2015, pp. 219-224.

[2] E. Stobert, R. Biddle, Authentication in the Home", Workshop on Home Usable Privacy and Security (HUPS), July 24, 2013, Newcastle, UK, http://cups.cs.cmu.edu/soups/2013/HUPS/HUPS13-ElizabethStobert.pdf. Retrieved Dec. 4, 2017.

[3] P. Sherin and K. G. Raju, Multi-level authentication system for smart home-security analysis and implementation, Proc. of IEEE International Conference on Incentive Computation Technologies (ICICT), Aug. 26-27, Coimbatore, India, DOI: 10.1109/INVENTIVE.2016.7824790

[4] P. Madsen, Authentication in the IoT: challenges and opportunities, http://www.secureidnews.com/news-item/authentication-in-the-iot-challenges-and-opportunities. Retrieved Dec. 4, 2017.

[5] U. Saxena, J. S. Sodhi, Y. Singh, Analysis of security attacks in a smart home networks, Proc. of 7th Intl. Conference on Cloud Computing, Data Science & Engineering, Noida, India, Jan. 12-13, 2017, pp. 431-436, 2017.

[6] S. Peter and R. K. Gopal, Multi-level authentication system for smart home-security analysis and implementation, Proc. of Intl. Conference on Incentive Computation Technologies (ICICT), Coimbatore, India, Aug. 26-27, 2016, DOI: 10.1109/INVENTIVE.2016.7824790

[7] M. S. Raniyal, I. Woungang, S. K. Dhurandher, An Inter-Device Authentication Scheme for Smart Homes using One-Time-Password over Infrared Channel, (Accepted Aug 10, 2017) to the International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments (ISDDC 2017), Oct. 26-28, 2017, Vancouver, BC, Canada.

[8] NIST FIPS, http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf. Retrieved Dec. 4, 2017.

[9] D. Garg and S. Verma, Improvement over Public Key Cryptographic Algorithm, IEEE International Advance Computing Conference (IACC 2009), Patiala, India, March 2009

[10] H. Om and S. Kumari, Comment and modification of RSA based remote password authentication using smart card, Journal of Discrete Mathematical Science and Cryptography, 2017, p. 625-635.

[11] K. Ashton, That 'Internet of things' thing, RFID Journal, 2009, http://www.rfidjournal.com/articles/view?4986. Retrieved Dec. 4, 2017

[12] Z. Dawy, W. Saad, A. Ghosh, J. G. Andrews, E. Yaacoub, Towards massive machine type cellular communications, IEEE Wireless Communications, Vol. 24, Issue 1, Feb. 2017, pp. 120-128.

[13] V. L., Shivraj, M. A. Rajan, Singh Meena, P. Balamuralidhar, "One time password authentication scheme based on elliptic curves for Internet of Things (IoT)". Proc. of IEEE 5th National Symposium on Information Technology: Towards New Smart World (NSITNSW 2015), Feb. 17-19, 2015, pp. 1-6.

[14] E. Stobert, R. Biddle, Authentication in the Home", Workshop on Home Usable Privacy and Security (HUPS), July 24, 2013, Newcastle, UK, http://cups.cs.cmu.edu/soups/2013/HUPS/HUPS13-ElizabethStobert.pdf. Retrieved Dec. 4, 2017.

[15] PureLiFi, http://purelifi.com. Retrieved Dec. 4, 2017.

[16] Z. Tian, K. Wright, X. Zhou, The DarkLight Rises: Visible Light Communication in the Dark", http://www.cs.dartmouth.edu/ xia/papers/mobicom16-darklight.pdf. Retrieved Aug. 9, 2017.

[17] P. Kumar, A. Braeken, A. Gurtov, J. Iinatti, P. H. Ha, Anonymous Secure Framework in Connected Smart Home Environments, IEEE Transactions on Information Forensics and Security, Vol. 12, Issue 4, Jan. 2017, pp. 968-979.

[18] C. C. Yang, R. C. Wang, and T. Y. Chang, An improvement of the Yang-Shieh password authentication schemes, Applied Mathematics and Computation, vol. 162, 2005, pp. 1391-1396.

[19] N. Shone, C. Dobbins, W. Hurst, Q. Shi, Digital Memories Based Mobile User Authentication for IoT, Proc. of IEEE Intl. Conference on Computer and Information Technology, Ubiquitous Computing and Communications, Dependable, Autonomic and Secure Computing, Pervasive Intelligence and Computing, 2015, Oct. 26-28, Liverpool, UK, pp. 1796-1802

[20] P. P. Gaikwad, J. P. Gabhane, S. S. Golait, 3-Level Secure Kerberos Authentication for Smart Home Systems Using IoT, Proc. of 1st IEEE Intl. Conference on Next Generation Computing Technologies (NGCT) Dehradun, India, Sept. 4-5, 2015, pp. 262-268.

[21] T. Borgohain, A. Borgohain, U. Kumar, S. Sanyal, Authentication Systems in Internet of Things, https://arxiv.org/abs/1502.00870. Retrieved Dec 4, 2017.

[22] H. Om and M. Reddy, RSA based remote password authentication using smart card. Journal of Discrete Mathematical Science & Cryptography, 2012, 15(2): 105-111, 2012.

[23] X. Wang, W. Zhang, An efficient and secure biometric remote user authentication scheme using smart cards, Proc. of Pacific-Asia Workshop on Computational Intelligence and Industrial Application (PACIIA'08), Wuhan, China, Dec 19-20, 2008, DOI: 10.1109/PACIIA.2008.382, pp. 913-917.

[24] Remote user authentication using NFC, US patent US20110212707, https://www.google.ch/patents/US20110212707. Retrieved Dec. 4, 2017.

[25] SPAN/AVISPA Tool, http://www.avispa-project.org. Retrieved Dec. 4, 2017.

[26] M. Burrows, M. Abadi, R. Needham, A logic of authentication, ACM Trans. on Computer Systems, Vol.8 (1), pp.18-36,1990.

[27] P. Wilson, Inter-Device Authentication Protocol for the Internet of Things, MSc Thesis, Department of Electrical and Computer Engineering, University of Victoria, B.C, Canada, July 2017.

[28] X. Yao, X. Han, X. Du, X. Zhou, "A lightweight multicast authentication mechanism for small scale IoT applications", IEEE Sensors Journal, Vol. 13, Issue 10, June 2013, pp. 3693-3701.

[29] J. L. Hernandez-Ramos, M. P. Pawlowski, A. J. Jara, A. F. Skarmeta, L. Ladid, Toward a lightweight authentication and authorization framework for smart objects, IEEE Journal on Selected Areas in Comm., Vol. 33, Issue 4, Apr. 2015, pp. 690-702.

[30] P. Hughes, ARM and Sensor Platforms Deliver an Open Source Framework for Sensor Devices, ARM, https://www.arm.com/about/newsroom/arm-and-sensor-platforms-deliver-an-open-source-framework-for-sensor-devices.php. Retrieved Aug. 9, 2017.

[31] Y. Sharaf-Dabbagh, W. Saad, On the authentication of devices in the Internet of Things, Proc. the 17th IEEE Int. Symposium on World Wireless, Mobile Multimedia Networks, Coimbra, Portugal, Jun. 2016, pp. 1-3.

[32] S. Shin; H. Yeh; K. Kim, An effective device and data origin authentication scheme in home networks, Proc. of 9th Intl. Conference and Expo. on Emerging Technologies for a Smarter World (CEWIT), Nov. 5-6, Incheon, South Korea, Sept. 2012, pp. 1-5.

[33] P. Kumar, A. Gurtov, J. Iinatti, M. Ylianttila, M. Sain, Lightweight and Secure Session-Key Establishment Scheme in Smart Home Environments, IEEE Sensors Journal, Vol. 16, Issue 1, Jan. 2016, pp. 254-264.

[34] F. K. Santoso, N. C. H. Vun, Securing IoT for smart home system, Proc. of Intl. Symposium on Consumer Electronics (ISCE), Madrid, Spain, June 24-26, 2015, pp. 1-2.

[35] Steve Friedl's Unixwiz.net Tech Tips, An Illustrated Guide to SSH Agent Forwarding, http://www.unixwiz.net/techtips/ssh-agent-forwarding.html. Retrieved Dec. 4, 2017.

[36] N. Park and N. Kang, Mutual Authentication Scheme in Secure Internet of Things Technology for Comfortable Lifestyle, Sensors 2016, 16(1), 20; doi:10.3390/s16010020

[37] J. Kang, G. Park, J-H. Park, Design of secure authentication scheme between devices based on zero-knowledge proofs in home automation service environments, Journal of Supercomputing, Vol. 72, DOI: 10.1007/s11227-016-1856-y, 2016, pp. 4319-4336.

[38] B. Vaidya, D. Makrakis, H. T. Mouftah, Device Authentication Mechanis m for Smart Energy Home Area Networks, Proc. of IEEE Intl. Conference on Consumer Electronics (ICCE), Jan. 9-12, 2011, Las Vegas, NV, USA, pp. 787-788

[39] W. H. Yang and S. P. Shieh, Password authentication schemes with smart cards, Computers and Security, Vol. 18, no. 8, pp. 727- 733, 1999.

[40] C. K. Chan, L. M. Cheng, Cryptanalysis of a timestamp-based password authentication scheme, Computers and Security 21 (I), 2002, pp. 74-76.

[41] H. M. Sun, H. T. Yeh, Further cryptanalysis of a password authentication scheme with smart cards, IEICE Transactions and Communications E86-B (4), 2003, pp. 1412-1415.

[42] L. Fan, J. H. Li, and H. W. Zhu, An enhancement of timestamp based password authentication scheme, Computers & Security, vol. 21, pp. 665-667, 2002.

[43] C. C. Yang, R. C. Wang, and T. Y. Chang, An improvement of the Yang-Shieh password authentication schemes, Applied Mathematics and Computation, vol. 162, pp. 1391-1396, 2005.

[44] J.J. Shen, C.W. Lin, M. S. Hwang, Security enhancement for the timestamp-based password authentication scheme using smart cards, Computers and Security 22 (7) (2003) 591595.

[45] Y. Liu, A. M. Zhou, M. X. Gao, A new mutual authentication scheme based on nonce and smart cards, Computer Communications 31 (10) (2008) 2205-2209.

[46] H. Y. Chien, J. K. Jan, and Y. M. Tseng, An efficient and practical solution to remote authentication: smart card, Computers & Security, vol. 21, no. 4, pp. 372375, 2002.

[47] C. L. Hsu, Security of Chien et al.'s remote user authentication scheme using smart cards, Computer Standards & Interfaces, 2003.

[48] H. Bar-El, Known Attacks Against Smart cards, *http* : $//www.infosecwriters.com/text_resources/pdf/Known_Attacks_Against_Smartcards.pdf$, Retrieved Dec. 4, 2017

[49] C. A. M. Paixao and D L G Filho. An efficient variant of the RSA cryptosystem, Eprint Archive, 2003.

[50] Zero-Day Attacks, http://www.pctools.com/security-news/zero-day-vulnerability. Retrieved Dec. 4, 2017

[51] Diffie-Hellman Key Exchange Protocol, $http://www.math.ucla.edu/\ baker/40/handouts/rev_DH/node1.html$. Retrieved Dec. 4, 2017

[52] SELinux Project, $https://selinuxproject.org/page/Main_Page$. Retrieved Nov 24, 2017

[53] SELinux, $https://wiki.centos.org/HowTos/SELinux$. Retrieved Dec. 4, 2017

[54] Restricting even root access to a folder, http://blog.siphos.be/2015/07/restricting-even-root-access-to-a-folder/. Retrieved Dec. 4, 2017

[55] User Management, https://help.ubuntu.com/14.04/serverguide/user-management.html. Retrieved Dec. 4, 2017

[56] RedHat-Disallowing Root Access, $https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Security_Guide/s2-wstation-privileges-noroot.html$. Retrieved Dec. 4, 2017

[57] Ubuntu Sudoers, https://help.ubuntu.com/community/Sudoers. Retrieved Dec. 4, 2017

[58] Limiting root access with sudo, http://www.techrepublic.com/article/limiting-root-access-with-sudo-part-1/. Retrieved Dec. 4, 2017.

[59] R. L. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM, Vol 21, No. 2, Feb. 1978, pp. 120-26

[60] RSA Labs. Public Key Cryptography Standards (PKCS), Number 1., https://tools.ietf.org/html/rfc3447. Retrieved Dec. 4, 2017

[61] Smart card application protocol data unit, *https* : *//en.wikipedia.org/wiki/Smart*$_c$*ard*$_a$*pplication*$_p$*rotocol*$_d$*ata*$_u$*nit*, Retrieved Dec. 4, 2017

[62] S. Ruj, A. Nayak, S. Naik, Securing home networks using Physically Unclonable Functions, Proc. of 4th International Conference on Ubiquitous and Future Networks (ICUFN), July 4-6, Puket, Thailand, 2012, pp. 288-293.

[63] T. Marktscheffel; W. Gottschlich; W. Popp, P. Werli, S. D. Fink, A. Bilzhause; H. de Meer, QR code based mutual authentication protocol for Internet of Things, Proc. of IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), June 21-26, 2016, Coimbra, Portugal, pp. 1-6.

[64] Open Source Universal Remote, *https* : *//upverter.com/alexbain/f*24516375*cfae*8*b*9*/ Open − Source − Universal − Remote/#/*. Retrieved Dec. 4, 2017.

[65] Linux Infrared Remote Control, http://www.lirc.org. Retrieved Dec. 4, 2017.

[66] Raspbian Operating System, https://www.raspberrypi.org/downloads/raspbian. Retrieved Dec. 4, 2017.

[67] Crypto++ Benchmark, https://www.cryptopp.com/benchmarks.html, Retrieved Nov. 28, 2017.

[68] A. Fiat. Batch RSA. In G. Brassard, ed., Proceedings of Crypto 1989, vol. 435 of LNCS, pp. 175185. Springer-Verlag, Aug. 1989.

[69] B. D, Shacham H. Fast variants of RSA. In RSA Laboratories' Crypto bytes, https://cseweb.ucsd.edu/ hovav/dist/survey.pdf. Retrieved Dec. 4, 2017

[70] NIST, Recommendation for key management, "http://nvlpubs.nist.gov/nistpubs /SpecialPublications/NIST.SP.800-57Pt3r1.pdf", Retrieved Nov. 28, 2017

[71] T. Collins, D. Hopkins, S. Langford, and M. Sabin, Public Key Cryptographic Apparatus and Method, US Patent #5848159, Jan. 1997.

[72] T. Takagi. Fast RSA-type Cryptosystem Modulo pkq. In H. Krawczyk, ed., Proceedings of Crypto 1998, vol. 1462 of LNCS, pp. 318326. Springer-Verlag, Aug. 1998. 36(3):553558. May 1990.

[73] M. Wiener. Cryptanalysis of Short RSA Secret Exponents, IEEE Trans. Information Theory, Vol. 36, Issue 3, May 1990, pp. 553-558

[74] F. Mulvey, A. Villanueva, D. Sliney, R. Lange, M. Donegan, "Safety issues and infrared light", Assistive Technologies: Concepts, Methodologies, Tools, and Applications, pp.1062-1083, 2013