Secure Neighbor Discovery in Wireless Networks

Berk Türetken, Ishani Bhardwaj, Javier Alberto Rosales Flores, Jinjia Zhang, Salem Wollel, Zainab Khan

Course Responsible and Examiner: Dejan Manojlo Kostic
Teaching Assistant: Ahmed Hussain

Communication System Design Final Report

Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

11 January 2024

Abstract

Neighbour discovery plays a crucial role in the communication of wireless devices, which enables those devices to discover each other and establish a connection. Wireless devices identify each other based on their physical proximity or communication range. However, neighbor discovery protocols are vulnerable to various kinds of attacks, particularly man-in-the-middle attacks. In these attacks, the attacker relays messages between non-neighboring devices by deceiving them into believing that they are legitimate neighbors. Secure neighbor discovery protocols should thus be put into place to deal with these problems. Neighbor discovery procedures can be secured in several ways. This course project explores six distinct methods for enhancing the security of neighbor discovery protocols including time-based, location-based, time-location-based, and GeoTimeCert authentication. The project demonstrates how the GeoTimeCert authentication approach is effective in minimizing potential active attack scenarios. However, it is acknowledged that no single approach can universally guarantee security. Consequently, the project encourages the exploration of potential attacks against the protocol.

Keywords: neighbour discovery, secure neighbor discovery, man-in-the-middle attack, time-based, location-based, time-location-based.

Acknowledgements

We would like to express our deepest appreciation to our supervisor, Ahmed Hussain, for his unwavering support and insightful guidance throughout the course of this project on secure neighbor discovery. His expertise have been invlauable throughout the entire process of researching, implementing, and writing for this project. We are also very grateful to the course teacher Dejan Manojlo Kostic for the time and effort he invested in providing us with constructive feedback and encouragement which led to the refinement of the project.

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms and Abbreviations

This document requires readers to be familiar with terms and concepts described in RFC 1235 [2]. For clarity, we summarize some of these terms and give a short description of them before presenting them in the next sections.

AES	Advanced Encryption Standard
CA	Certificate Authority
CBC	Cipher Block Chaining
CQI	Channel Quality Indicator
CSR	Certificate Signing Request
DS	Digital Signature
Dev$_1$	Device 1
Dev$_2$	Device 2
ECDH	Elliptic-curve Diffie–Hellman
EdDSA	Edwards-curve Digital Signature Algorithm
GCM	Galois Counter Mode
HMAC	Hash-based message authentication code
IPv4	Internet Protocol version 4 (RFC 791 [3])
IPv6	Internet Protocol version 6 (RFC 2460 [4])
IV	Initialization Vector
lat	Latitude
lng	Longitude

MAC Address	Media Access Control Address
MitM	Man-in-the-Middle
ND	Neighbor Discovery
OAEP	Optimal Asymmetric Encryption Padding
PK	Public Key
PKCS	Public-Key Cryptography Standards
PR	Private Key
RF	Radio Frequency
RKM	Random Key Material
SND	Secure Neighbor Discovery
SSK	Shared Secret Key
ToF	Time of Flight

Chapter 1

Introduction

Wireless mobile communication technologies have grown significantly over the past decade, wireless devices have become ubiquitous in homes, factories, and hospitals [5]. Emerging mobile ad hoc and sensor networking paradigms create multi-hop topology, relaying packets across multiple wireless links. These developments enable various applications to communicate in urban environments, such as building monitoring, disaster relief operations, tactical operations, static ad hoc networks, and low-mobility ad hoc networks. Examples include handheld devices, wearable gadgets and radio frequency identifiers.

Wireless communications are flexible, allowing devices to communicate instantly without a cable connection. As wireless connections are frequently established, direct device discovery is an essential part of the communication [6].

Neighbor discovery (ND) enables wireless devices to discover one another in order to establish direct communication [7]. It is the fundamental aspect of wireless networking, as the connections are frequently initiated and terminated [5]. The challenge lies in identifying nearby devices, which can be in close proximity. It is assumed that if two nodes can communicate directly, they are within each other's communication range. However, physical proximity does not necessarily guarantee communication.

1.1 Problem description

Wireless communications' open nature makes attacks against ND easy, as adversaries can easily mislead disconnected nodes into believing that they are neighbours [7]. By creating a false path, an adversary can relay messages between non-neighboring devices. This deceives the two devices into believing that they are communication neighbors. The significance of this relay attack is due to the fact that it is easy for an adversary to act at the physical layer without any node compromise or possession of cryptographic keys. The adversary can then eavesdrop, modify, or abruptly drop messages, leading to a denial-of-service [6]. Secure neighbor Discovery (SND) prevents attacks by identifying only true neighbors as neighbors [7].

1.2 Hypothesis

This project hypothesizes that the GeoTimeCert authentication approach is optimal for significantly enhancing the security of secure neighbor discovery (SND) protocols, even though time-based, location-based, and time-location-based approaches enable certain levels of security. The implementation and performance evaluation of the Secure Neighbor Discovery (SND) protocols will demonstrate the tangible level of effectiveness of the protocols in mitigating relay attacks.

1.3 Purpose

The purpose of this project is to address the security issues that come with the traditional ND protocol in wireless communication. The project aims to contribute to the practical advancement of secure neighbour discovery by implementing and evaluating SND protocols and testing them against passive and active attack scenarios, ensuring wireless communications are reliable, secure, and trustworthy.

1.4 Research Goals

The goals of the project are presented as follows:

- Implement a basic ND protocol.
- Enhance security by adding authentication or encryption to the ND protocol.
- Implement attack scenarios against SND protocol and its variants.
- Evaluate the performance of the ND and SND (and its variants) protocols.

1.5 Research Questions

To guide our investigation into SND protocols, we formulate the following research questions:

- Why are traditional ND protocols not secure?

- What are the possible attack scenarios?
- How can an attacker try to manipulate wireless communications, and what methods do they use?
- What extra methods can be added to the ND protocol to make it more secure?
- Why isn't relying solely on time and location enough to secure the network, and what are their limitations?
- Why is the GeoTimeCert authentication strategy effective in preventing man-in-the-middle (MitM) attacks?
- How can we measure the efficiency of the protocols?

1.6 Research Methodology

To address the above-mentioned research questions related to designing, enhancing, and evaluating SND protocols, a combination of qualitative and quantitative methods for analysis and evaluation are utilized. In order to accomplish the work, Alfa devices, AWUS036ACHM, are used to simulate wireless communications. The research methodologies are divided into the following phases:

1. **Phase 1**

   In order to grasp a thorough understanding of the domain, our team starts to study and examine a large number of research articles. The reading sources cover a range of topics, including ND, SND, and the implementation of a verification protocol. We also look into articles regarding some cryptographic functions involved in SND to authenticate and encrypt communication. The literature review gives the team a head start on the research journey by making sure that all the team members have a certain level of insight to move forward with the implementation.

2. **Phase 2**

   The implementation phase takes the next step to experimentally answer the research questions. According to the knowledge that is gathered in the first phase, the implementation of the ND protocol is launched. Once the ND protocol becomes successful, encrypting and authenticating the communication follows. To check the level of security of the ND protocol, a passive and active attacker model is designed against it. The next step is to enhance the security of the protocol by using time-based, location-based, and time-location-based models. We also show why these methods are not sufficient by themselves by designing various attackers. Finally, the GeoTimeCert authentication approach is implemented, showing that it is the optimal way to secure wireless neighbor discovery communication.

3. **Phase 3**

   The last and final step is the evaluation and analysis of the protocols. Performance benchmarking is assessed by comparing the efficiency and overall performance of the cryptographic functions by running them in multiple iterations. The quantitative results, which are obtained from this phase, enable the team to answer the research questions and reach a conclusion.

The implementation of the protocols then follows an empirical approach, which includes a tremendous amount of experimentation, testing, and observation. The performance evaluation, which is the last phase, compares the effectiveness of the protocols using a quantitative approach.

1.7 Delimitations

- **Limited distance range:** Alfa devices operate within a limited device range.
- **Single-path communication:** The evaluation is based on a single and direct communication path. Multi-hop communication is not considered in the study.
- **No absolute security:** We acknowledge that no system is absolutely secure.
- **Cryptographic limitations:** The study recognizes limitations in cryptographic functions themselves.
- **Focus on application layer:** We mainly consider security in the application layer, neglecting security assessments in the lower layers.
- **Virtual environment divergence:** The use of virtual environments may deviate from the actual system configuration.
- **Performance deviation:** Performance evaluations conducted in virtual environments may differ when applied in real-world scenarios.

- **Scalability challenges:** Since the study is conducted on a limited number of Alfa devices, possible deviations might occur in a cluster.
- **Complex Threat Scenarios:** Recognizing that actual threat scenarios may be more complicated than those explored in this study.
- **Exclusion of SND Verification Protocol:** The SND verification protocol is beyond the defined scope of this study.

1.8 Ethical Considerations

1.8.1 Guiding principles

Even though the implementation is being done in virtual environments, it is crucial to follow certain ethical considerations to avoid undesirable results:

1. **Informed consent**

   Participants should be fully informed about the nature, goals, risks, and rewards of the research prior to their participation. This should also be applied to the project team members, ensuring that everyone is aware of the project goals and the direction of the research.

2. **Minimizing harm and responsible use**

   While performing the implementation, only devices that belong to the team members should be utilized with their consent. In addition, the attack scenarios should only be executed against the devices that are meant to communicate, according to the research. The attack should not target any devices outside of the controlled setup.

3. **Privacy and data protection**

   Aside from the objectives of the study, any personally identifying information from the participating devices should not be gathered, stored, transmitted, or processed. It is also crucial to ensure that no physical gadget is harmed throughout the experiment. Although the experiment is conducted in a virtual world, caution is required not to accidentally break things in the real world.

4. **Transparency and honesty**

   If something unexpected happens during the experiment, the people involved should be immediately notified, and the incident should be documented whenever it is necessary to do so. Transparency should be emphasized by offering comprehensive records of algorithms, techniques, and outcomes. Acknowledge the limitations of the research and state them explicitly.

5. **Results communication**

   Before announcing, sharing, or publishing the results of the research, consult with every team member. As agreed upon by all team members, delay the releasing of any results or restrict the amount of information. Any disclosure of methodologies, code implementations, documentation, or algorithms falls under this category.

1.8.2 Preventing research misuse

To promote space use of the findings, this section examines fundamental guidelines for usage. These principles provide an initial commitment to guarantee the ethical integrity of the research by concentrating on risk mitigation and potential misuse prevention.

1. **Reconnaissance and vulnerability analysis risks on research**

   Although navigating the project is permitted to comprehend the dynamics of the system, if this is not done carefully, it could result in misuse. Malicious actors may wish to examine the system to identify weaknesses that can be exploited. It should be recognized that vulnerability analysis and reconnaissance efforts of any kind, aside from additional research and educational objectives, are not appropriate. The goal of this proactive strategy is to defend against potential misuse and preserve the integrity of the study.

2. **Potential for reverse engineering**

   If the team chooses not to release implementation code or specifics, malicious parties could attempt to reverse engineer the protocol to identify vulnerabilities and launch an attack. This could be avoided by providing only broad guidelines and disclosing implementation details in restricted detail. However, this is not a cause for concern if the team decides to release the implementation code later on.

3. **Restricted to educational and research purposes**

   The utilization of the information, methodologies, and findings of the research project is exclusively intended for learning and research purposes. It is in our best interest to prevent any malicious and unethical applications of the knowledge and research outcomes.

1.9 Sustainable development

1. **Environmental considerations**

    - **Energy efficiency:** integrate low-power communication modes into SND implementations on Alfa devices to reduce energy consumption. Consider mechanisms for energy-saving mechanisms during idle times.
    - **Resource optimization:** ensure responsible and effective usage of the hardware components, Alfa devices, to avoid electronic waste and the need for repair and maintenance.
    - **Processing Power and Bandwidth utilization:** Reduce the amount of computing power and requirements by streamlining the protocols. Analyze how SND processes affect bandwidth consumption and explore methods for improving data transfer.

2. **Economic impacts**

    - **Cost effectiveness:** evaluate the economic impacts of the implementation of the SND protocols, including device compatibility and the need for potential upgrades

3. **Social implications**

    - **Safety and privacy:** SND protocol usage must be secure enough to guarantee individual's online safety and privacy. Sensitive information should be kept confidential by designating the SND protocols in such a manner.
    - **Digital Inclusion:** There is common ground that devices with different technological characteristics should be able to access and use the SND protocols. Designing and recommending an SND protocol should take this into consideration.

1.10 Structure of the Report

The report is structured into key sections: Chapter 2 covers the background, including ND, SND, RSA encryption, and digital signatures. Chapter 3 presents the system architecture. Chapter 4 outlines the scenario and threat model. Chapter 5 evaluates system performance. In Chapter 6, we draw our research to a conclusion, emphasizing the essential takeaways and charting potential paths for future investigations.

Chapter 2

Background and Related Work

2.1 Neighbor Discovery

2.1.1 Classical Neighbor Discovery

Neighbor discovery (ND) is a crucial process in wireless networking to establish communications between neighbors. Devices, that may or may not be geographically located near to each other but are within transmission range of each other and have a common channel among them, are considered neighbors [8]. ND protocols are fundamental for traditional wireless networks, particularly in multi-hop communication involving data forwarding and route discovery.

It's important to note that different types of protocols exhibit varying performance characteristics under different wireless network settings, such as symmetric duty cycles or asymmetric duty cycles. Hence, specific network conditions and energy constraints are critical considerations in the design and deployment of an ND protocol [9].

2.1.1.1 Neighbor Discovery Algorithm Classifications According to Approach

Khan *et al.* [10] classify the ND protocols into five categories based on their approaches as follows:

- **Distance bounding approach:** In this approach, the time it takes for a signal to travel to a potential neighbor and back is measured, and then this time measurement is combined with the speed at which the signal propagates to determine the distance to that neighbor. These protocols incorporate cryptographic techniques to ensure a secure discovery process. Particularly with radio frequency (RF) signals, it is highly challenging for an adversarial node to manipulate or reduce the estimated distance. The distance bounding approaches come in two variations: challenge-response delay measurement and message time-stamping.

- **Location-based approach:** ND relies on location information as the primary attribute in this approach. Location-based protocols fall into two categories: geographic packet leashes or guard-based defense against wormholes.
- **Directional and omnidirectional antenna approach:** Directional antenna techniques are employed to enhance protection by concentrating transmission energy in a specific direction. However, they come with limitations that can be overcome by using omnidirectional antennas, which are widely used in most wireless applications due to their simplicity.
- **RF fingerprinting and connectivity approach:** Certain signal patterns generated by radio transmitters are distinctively recognized by receivers, forming the basis of RF fingerprinting approaches. On the other hand, connectivity approaches utilize local network connectivity information to ensure SND. These methods do not necessitate specialized node hardware.
- **Centralized approach:** Centralized approaches encompass ND schemes that prioritize non-security-focused schemes. Designed for operation in long-range environments, these approaches primarily aim to detect the presence and approximate location of potential attackers.

Most ND protocols adopt a time-slotted communication model where continuous time is broken down into distinct slots, each characterized by different states. Researchers have classified neighbor discovery protocols into four categories based on underlying principles: randomness, over-half occupation, rotation-resistant intersection, and coprime cycles [9].

2.1.1.2 Possible Vulnerabilities of Neighbor Discovery Protocols

It is important to note that the characteristics of wireless communication create the following vulnerabilities regarding privacy and security for ND protocols:

- **Vulnerability to relay attacks:** ND protocols are susceptible to relay attacks, where malicious nodes can deceive honest nodes by relaying messages in a deceptive manner. This can lead to the formation of fictitious links between nodes, potentially compromising network integrity.
- **Lack of authentication:** Traditional ND protocols often lack robust authentication mechanisms. This means that nodes may have difficulty

verifying the identity of their neighbors, making them susceptible to impersonation attacks.

- **Privacy concerns:** ND protocols may inadvertently leak information about the network's structure and the presence of nodes. Attackers can use this information to perform network reconnaissance, track node movements, or identify valuable targets.

- **Eavesdropping:** In many wireless communication scenarios, ND protocols transmit information in the clear, making it susceptible to eavesdropping. This can lead to the disclosure of sensitive data and compromise privacy.

- **Limited security measures:** ND protocols may not incorporate strong security measures, especially in resource-constrained networks. This can leave nodes vulnerable to various attacks as security mechanisms are often not a primary focus.

- **Reliance on trust:** Some ND protocols rely on the assumption of trust between nodes within the network. In practice, trust may not always be guaranteed, and malicious nodes can exploit this assumption.

- **Verification challenges:** In wireless networks, it can be challenging to verify the physical proximity of nodes accurately. This can lead to false positives or negatives in neighborhood discovery, making it difficult to establish a reliable security perimeter.

- **Lack of forward secrecy:** Many ND protocols do not provide forward secrecy, meaning that if an attacker compromises a node's keys or identity, they can potentially decrypt past communication, compromising historical data privacy.

- **Inadequate key management:** The management of cryptographic keys in ND protocols can be complex. Weak key management practices can weaken security and make it easier for attackers to compromise communication.

- **Resource constraints:** In resource-constrained environments, strong security measures can be challenging to implement due to limitations in processing power, memory, and energy. This can make ND protocols more susceptible to attacks.

- **Failure to address new threats:** ND protocols may not be designed to adapt to emerging security threats effectively. As new attacks evolve, these protocols may become outdated and vulnerable.

The exploit of these vulnerabilities can lead isolated nodes to mistakenly believe they communicate directly. Such attacks can potentially undermine the functioning of higher-layer protocols and applications [7].

While ND protocols are critical for network operations, addressing these security and privacy concerns often requires integrating additional security measures, such as encryption, authentication, intrusion detection, and digital signatures, to make them more robust and resistant to attacks. Therefore, it is crucial to design SND protocols that enhance the security and privacy aspects of ND protocols.

2.1.2 Secure Neighbor Discovery

Secure Neighbor Discovery (SND) focuses on the characteristics of the signals used for message exchange. There are two kinds of neighbors: devices that can communicate directly are classified as *communication neighbors*, and devices within proximate distance are called *physical neighbors*. The neighborhood is prone to attacks by an adversary wherein an adversary can perform relay attacks. Therefore, securing ND is important to safeguard the systems and protocols from being compromised due to the dynamic nature of wireless devices.

The primary requirement for SND protocols is correctness, ensuring accurate identification of genuine neighbors while preventing malicious attackers from tricking the system. The notion of verification is incorporated into the SND functionality, playing a crucial role in confirming the status of a given system as a neighbor. This inclusion adds an additional layer of security to ND protocols. However, if message delivery fails or communication is jammed, several neighbors would not get identified, thus making ND partial. Moreover, in a properly managed environment, if a protocol successfully discovers all honest or correct devices in its neighborhood, it is referred to as a *complete* ND protocol [5].

SND mitigates the vulnerabilities and enhances the privacy and security of ND protocols by addressing various challenges and risks associated as follows:

- **Protection against relay attacks:** SND includes mechanisms to protect against relay attacks. It typically employs secure message exchanges using Cryptographically Generated Addresses [11] or Digital Signature (DS) to ensure that messages between nodes cannot be tampered with or relayed maliciously. Figure 2.1 represents the primary purpose of DS where a unique code from the sender is attached to verify the data's source integrity. This code is like a sender's key, which the recipient uses to verify the authenticity of the data. By authenticating communication and verifying message integrity, SND prevents the formation of fictitious links between nodes, thus maintaining network integrity.

- **Countermeasures against replay attacks:** Timestamp and nonce options serve as countermeasures against replay attacks in SND protocols by adding an additional layer of security. When combined, these options make it challenging for attackers to capture and replay messages successfully. These options can be used to verify the freshness and authenticity of incoming messages, helping to maintain the integrity and security of the network.

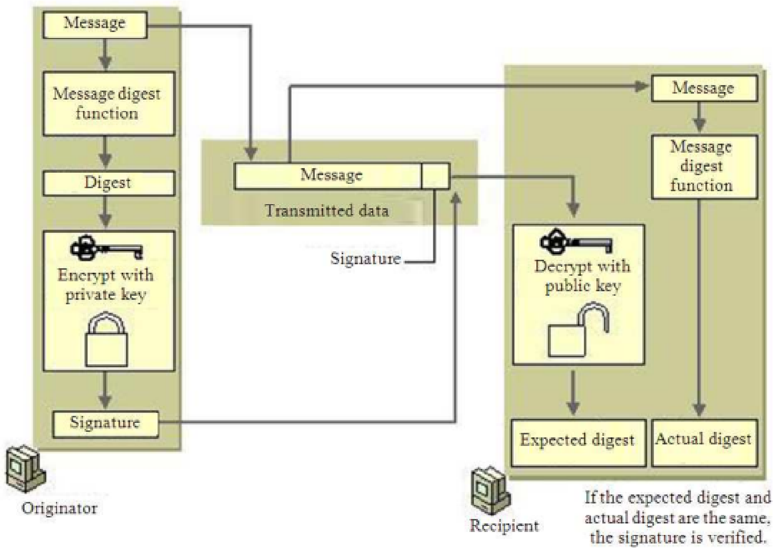


Figure 2.1: Digital Signature Mechanism [1]

- **Eavesdropping:** SND addresses the issue of eavesdropping by encrypting communication and implementing secure channels. This ensures that sensitive data remains confidential and cannot be intercepted easily, preserving privacy.

- **Forward secrecy:** SND typically incorporates forward secrecy by regularly updating cryptographic keys. This ensures that even if an attacker compromises a node's keys or identity, they cannot decrypt past communications, preserving historical data privacy.

- **Reliance on trust:** SND protocols reduce the reliance on trust assumptions. Instead, they rely on cryptographic mechanisms and authentication to establish trust, making it more difficult for malicious nodes to exploit trust-based vulnerabilities.

- **Adequate key management:** SND pays special attention to key management, using strong cryptographic practices and protocols to secure keys. This minimizes the risk of key compromise and associated communication breaches.

2.2 Cryptographic Preliminaries

2.2.1 RSA Encryption

The RSA algorithm is a suite of cryptographic algorithms that is based on public-key encryption (i.e., asymmetric encryption) for securely sending sensitive data over insecure networks.

There are two different keys involved in RSA. One is public and the other is private key. The public key can be shared with everyone whereas the private key must be kept secret. Any of these two keys can be used to encrypt the message, but only the other one can be used to decrypt it on the receiver end.

RSA ensures data integrity, confidentiality, non-repudiation, and authenticity during communication [12].

RSA encryption has some basic assets which are:

1. A public key
2. A private key
3. Product of two prime numbers (i.e., p and q)

The security relies on the assumption that a private key is supposed to be kept secret and additionally, it is very difficult to factorize the large integers that are the product of prime numbers p and q.

To understand more clearly, consider Alice and Bob as two legitimate users who want to communicate with each other using RSA encryption. Alice generates her keys with RSA encryption. Now, when Bob wants to send a message to Alice, he encrypts it with the public key of Alice. Upon receiving the encrypted message from Bob, Alice can only decrypt it using her private key.

Moreover, to ensure authenticity and non-repudiation of the message, Alice can digitally sign the message. This means creating a message hash, encrypting it with the RSA private key, and adding this encrypted hash to the message. On the other end, Bob can decrypt the hash value with Alice's public key and if that matches with the original message then it will be treated as an authenticated message from Alice.

In the basic authentication protocol of this project, we use *PKCS*#1 v1.5 (RSA) [13] for signature and *PKCS*#1 OAEP (RSA) [14] for encryption. This is further discussed in Section 3.2.

2.2.2 Advanced Encryption Standard (AES)

In the process of implementing basic authentication within the project, we encounter certain limitations associated with asymmetric encryption. Notably, asymmetric encryption demonstrates a high computational cost and imposes restrictions on data length. Consequently, in light of these challenges, we opt for symmetric encryption.

AES is another electronic data encryption scheme which is a symmetric block cipher. It encrypts data in blocks of 128 bits using the cryptographic keys of 128/192/256 bits. This type of symmetric encryption uses a shared secret key (*SSK*) between the sender and receiver for encryption and decryption purposes.

AES encryption consists of rounds which vary depending on the length of keys. For 128-bit keys, there are 10 rounds, 12 rounds for 192-bit, and 14 rounds for 256-bit keys. Each round consists of numerous processing steps to transform the plaintext input into the final ciphertext output.

Each round in the encryption process consists of the following four steps:

- Substitution of bytes
- Shifting data rows
- Mixing columns
- XOR operation between the data and round key

Similarly, the decryption process also consists of multiple rounds, which are opposite to the encryption, as can be seen below:

- Adding round key
- Inversing mixed columns
- Shifting rows
- Inversing substitution of bytes

2.2.2.1 Cipher Block Chaining (CBC) Mode

In CBC mode, the encryption of the same plaintext data results in different ciphertexts. This is achieved with the usage of a unique initialization vector (IV). Initially, an XOR operation is performed on the plaintext block and the IV, and then the process is carried out with an encryption key. After these steps, the output of each block goes through an XOR operation with the next plaintext block [15].

CBC mode has some shortcomings associated with it, which is why we shift to GCM mode for encrypting the data in Section 3.3. Some of the common shortcomings are:

- It supports encryption of sequential blocks only, which means that you need to calculate each block before moving to the next one. Hence, the process cannot be parallelized.
- It uses an IV which is vulnerable to attacks if it is not completely random.
- This mode can leak a large amount of data, especially in the case of large messages.

2.2.2.2 Galois Counter Mode (GCM)

GCM mode provides authenticated encryption to ensure confidentiality and authentication, as well as it is capable of checking the integrity of data. It has two operations which are authenticated encryption and decryption. The GCM mode is also faster than CBC mode and more secure with better implementation for table-driven field operations [15].

The GCM mode takes the following four inputs [16]:

1. A secret key
2. Nonce or IV
3. Plaintext message
4. Additional authenticated data

It then produces the following two outputs:

1. Ciphertext
2. Authentication tag - also known as message authentication code

It accepts pipelined implementations and exhibits minimal computational latency.

2.2.3 Diffie-Hellman Key Exchange

For having a more secure *SSK* exchange between the transmitter and receiver, we opt to use the DH key exchange protocol which is discussed below.

DH key exchange is one of the first widely used protocols for exchanging keys over an insecure channel between two communicating parties. The main problem solved by Diffie Hellman is the exchange of shared secrets between two parties who know nothing about each other. These shared secrets can be used to derive keys which can later be used with symmetric-key algorithms, such as AES, to encrypt the communication.

DH key exchange works in a way that the two communicating parties (i.e., Alice and Bob) first mutually decide upon two numbers (modulus *p* and a base *g*) to start with. Both of these numbers are publicly available and can be easily known to any potential eavesdropper. But the security lies in the fact that both Alice and Bob choose secret numbers (i.e., a and b) which are private to both of them and are not shared. They then perform a mod operation using the known public number of each other along with their secret number:

$$x(Alice) = g^a mod p$$

and,

$$y(Bob) = g^b mod p$$

The result of this operation is a generated key on each end, which is then sent over the network to be received by the other one. Having the generated key from Bob, Alice then calculates the shared secret using the key she received from Bob and her secret number, and Bob does the same such that:

$$k(Alice) = y^a mod p$$

and,

$$k(Bob) = x^b mod p$$

Algebraically, it is proved that $k(Alice)$ is equal to $k(Bob)$. This means that they both now have a symmetric *SSK* which can now be used to encrypt further communication [17, 18].

2.2.4 Edwards-curve Digital Signature Algorithm (EdDSA)

In the authentication with the ECDHE protocol, we use EdDSA for digital signatures. To be particular, we use the Ed25519 elliptic curve algorithm but there is another famous one which is Ed448. We conduct performance tests of both Ed25519 and Ed448 on different digest modes and it is discussed in Chapter 5. The implementation details of Ed25519 in this project are further discussed in Chapter 3.3.

EdDSA is a secure and modern algorithm for digital signatures that is based on performance-optimized elliptic curves [19].

The EdDSA signing function takes a text message and the signer's EdDSA private key as input parameters and produces an output signature which is essentially a pair of integers {R,s}.

$$EdDSA\_sign(msg, privKey) \rightarrow \{R, s\}$$

For verification of the signature, the algorithm takes an input message, the signer's EdDSA public key, and the previously produced EdDSA signature as input parameters. Then, it produces a boolean value as output which indicates if the signature is valid or not.

$$EdDSA\_signature\_verify(msg, pubKey, signature\{R, s\}) \rightarrow valid/invalid$$

2.2.5 Hash-based message authentication code

Another cryptographic authentication technique that we use in this project is a *Hash-based message authentication code (HMAC)*. HMAC uses a secret key and hash function to authenticate the data. In comparison to asymmetric cryptography and signature-based approaches, HMAC helps in verifying the authentication and integrity of the exchanged data with the usage of the shared secret key.

HMAC proposes a valid solution for the two users who intend to communicate with each other but distrust the internet, want to ensure that their communication remains private, and also do not want to compromise on data modification during the packet exchange.

Primarily HMAC relies on two factors:

- **Shared cryptographic secret keys:** HMAC uses *SSK*s to decrypt and read the altered data by the encryption algorithm.
- **Hash function:** HMAC uses cryptographic hash functions like SHA-1, SHA-256, MD5, or RIPEMD-128/60 to alter the data one more time to add an extra layer of security, making it more difficult to be tempered by the adversary.

Also, both the communicating parties need to agree on the following for HMAC to work:

- ***SSK*:** They both need to have a *SSK* for decrypting the information received from each other.
- **Algorithm:** They must agree on one hash function that is used for hashing the messages before they go out into the network.

HMAC provides an added layer of security by making messages irreversible and resistant to hacking, even making the length of the message unpredictable. In essence, without the shared secret key, the message contents become entirely inaccessible to any unauthorized attempts at reading them [20].

2.2.6 Certificate Authority (CA)

To increase the authenticity and trustworthiness of the communication between the devices/nodes, we implement certificate authority (CA) in the project.

CA is a trusted entity that issues digital certificates that link an entity cryptographically with a public key. CA-issued digital certificates ensure the communicating nodes that they are exchanging messages with a legitimate entity, instead of an attacker.

This digital certificate validates the identity of an entity. It typically contains information about the entity like, name, domain name, public key, expiry date, and certificate issue date. The process of issuing a digital certificate by CA is as follows:

1. The entity that needs the certificate generates a pair of private and public keys.
2. It initiates a certificate signing request (CSR) which includes the public key of the entity along with the domain name, organization name, and additional information.
3. This CSR request is sent to the CA. The CA then verifies the information and the identity of the applicant. Afterward, it issues a digital certificate to the applicant by signing it with its private key and sends it back to the applicant.
4. Finally, the certificate can be used for authentication purposes when communicating over the Internet. Anyone can authenticate it by using CA's public key.

The CA certificate upholds the integrity of documents signed with it through encryption, ensuring secure communication over the internet [21].

2.3 Distance Estimation

2.3.1 Euclidean Distance Formula

Euclidean distance in coordinate geometry is the distance between two points on a straight line. It is used to calculate the length of the segment connecting two points on the plane to locate them.

The Euclidean distance formula for two points (x_1, y_1) and (x_2, y_2) in a two-dimensional plane is defined as following:

$$d = \sqrt{[(x_2 - x_1)^2 + (y_2 - y_1)^2]}$$

where,

- (x_1, y_1) are the coordinates of the first point,
- (x_2, y_2) are the coordinates of the second point, and
- d is the distance between these two points.

Euclidean suits the best in a situation where direct distance is calculated between two points on a straight line, but when there are obstacles on the plane or the line is not straight such as the length of the diagonal of a triangle, Euclidean's heuristic function is not the most appropriate choice. However, for such cases, there is another equation called the Haversine formula which calculates the distance of an arc between two points on longitude and latitude [22].

2.3.2 Haversine Formula

The Haversine formula [23] is used to compute the most direct distance between two locations on a sphere by considering their latitudinal and longitudinal coordinates along the surface. It is widely used in navigation. In trigonometric terms, the Haversine formula can be expressed as:

$$haversine(\theta) = sin^2(\theta/2)$$

For the central angle (d/r) the haversine is calculated as:

$$(d/r) = haversine(\phi_2 - \phi_1) + cos(\phi_1)cos(\phi_2)haversine(\lambda_2 - \lambda_1)$$

Here, r represents the radius of the earth, d is the distance between the two points, ϕ_1, ϕ_2 are the latitudes, and λ_1, λ_2 are the longitudes of the two points respectively. If we apply inverse haversine or use the inverse sine function to compute d, we get:

$$d = 2rsin^{-1}(\sqrt{sin^2((\phi_2 - \phi_1)/2) + cos(\phi_1)cos(\phi_2)sin^2((\lambda_2 - \lambda_1)/2)})$$

2.4 Wireless Channel Quality

2.4.1 Channel Quality Indicator (CQI)

Channel Quality Indicator (CQI) is used to determine the strength or quality of a wireless channel in a wireless communication system. It is used in the optimization of wireless networks to ensure reliability and performance [24].

CQI is a metric value that shows the quality or strength of the communication in a wireless network at any given point in time. It uses several radio parameters to reflect the characteristics of the wireless channel, such as received signal strength indication, signal-to-noise ratio, reference signal received quality and reference signal received power.

Chapter 3

System Architecture

In this chapter, we delve into the architectures of the SND systems. We consider six different types of SND protocols, namely basic authentication, authentication with Elliptic-Curve Diffie-Hellman Ephemeral (ECDHE), time-based authentication with ECDHE, location-based authentication with ECHDE, time-location-based authentication with ECDHE, GeoTimeCert authentication with ECDHE, each designed to progressively enhance the security and reliability of the ND process in wireless networks. In the following sections of this chapter, we provide a detailed explanation of the testbed setup, the above-mentioned SND protocols, and how they contribute to the overall security of the ND.

3.1 Testbed Setup

In our SND testbed, we have established a setup that includes three components as depicted in Figure 3.1. Two Alfa AWUS036ACHM 802.11ac Wi-Fi USB adapters are configured as honest and legitimate neighbor devices (i.e., Dev_1 and Dev_2) while the third Alfa adapter is configured as an attacker.

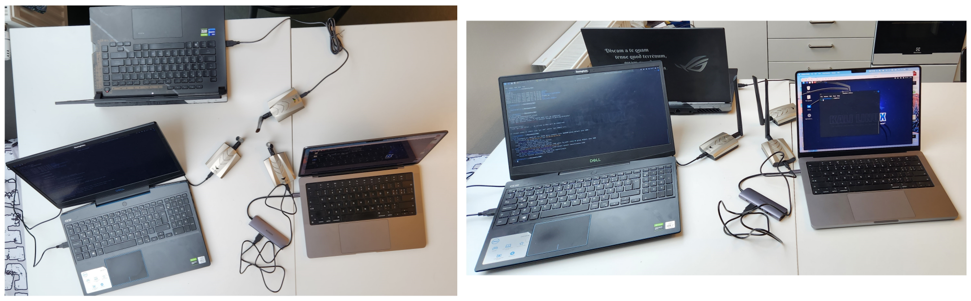


Figure 3.1: Project setup including the attacker

1. Legitimate devices: Both Dev_1 and Dev_2 are equipped with high-gain antennas and serve as essential components in the SND testbed, contributing to the realism of our wireless network simulations. They play a critical role in SND protocol evaluation, helping us assess the security and reliability of ND in wireless networks.

2. Attacker: It is used to simulate various attack scenarios, such as MitM and other security threats. This attacker node enables us to evaluate how SND protocols perform when faced with adversarial challenges.

By incorporating these three components into our testbed, including Dev_1, Dev_2 as honest neighbors, and Alfa Adapter 3 as the attacker, we can create realistic wireless network scenarios, evaluate SND methods under different conditions, and assess the effectiveness of security measures against potential threats. This testbed setup provides valuable insights into the robustness and security of SND protocols in the presence of both legitimate devices and potential attackers.

3.2 Basic Authentication

Figure 3.2 provides an overview of the basic authentication, which serves as the inaugural SND protocol. For simplicity, we presume the presence of only two legitimate devices capable of establishing a wireless network connection. Initially, Dev_1 transmits a message containing the term *hej*, the device's public key, its name, and a signature. Signatures are utilized for authenticity protection, integrity protection, and non-repudiation. We employ an RSA-based signature scheme, where messages are signed using the corresponding device's private key. In that case, Dev_1 signs the message with its private key. Subsequently, the messages, concatenated with the signature, form message *1* in Figure 3.2 and are sent to Dev_2 as an initialization message to establish a connection. Upon receiving message *1*, Dev_2 verifies its origin by referencing Dev_1's device name and validates message integrity by decrypting the signature using Dev_1's public key in step *2*. Once integrity is confirmed, Dev_2 generates a *SSK*. Following this, Dev_2 sends a similar message to Dev_1, containing the custom term *hejhej*, Dev_2's public key, its name, the encrypted version of *SSK* by Dev_1's public key, and a signature where Dev_2 signs all messages using its private key which is depicted in Figure 3.2 as message *3*. These steps represent the initial handshake process of the basic authentication protocol, ensuring secure and authenticated communication between the two devices.

The protocol description can be found below:

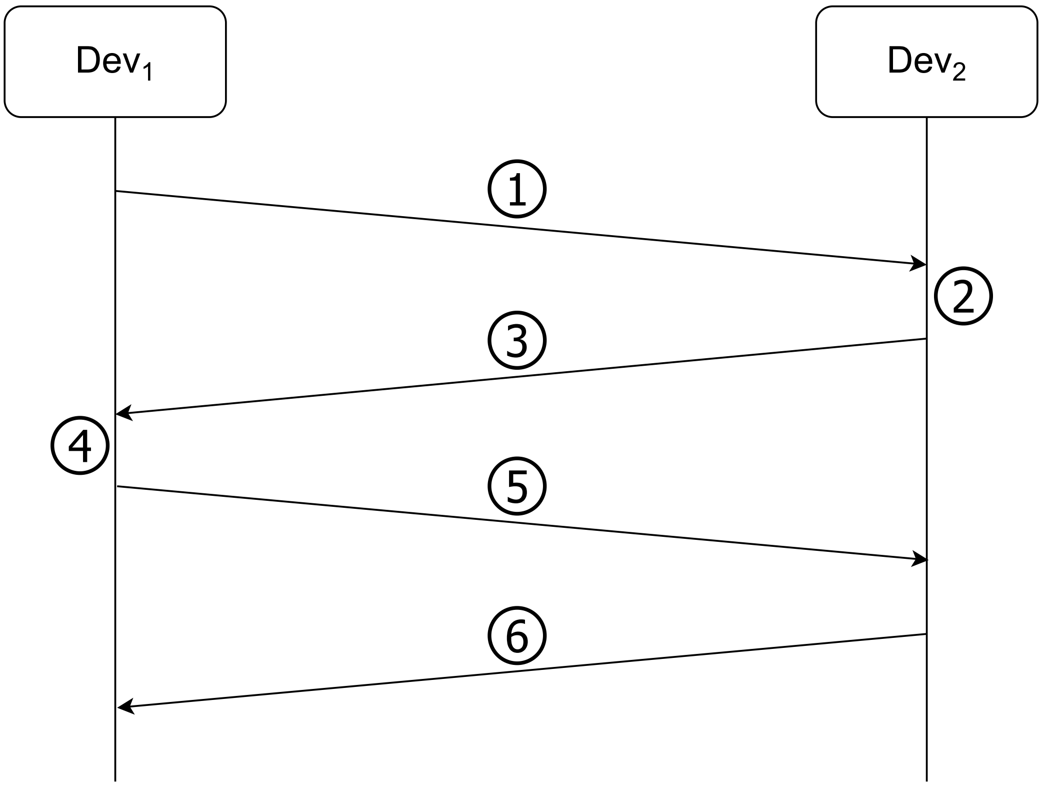


Figure 3.2: Basic Authentication Protocol Diagram

1. hej, PK_1, Dev_1, sign$_{PR1}$(hej, PK_1, Dev_1)
2. Signature verification
3. hejhej, PK_2, Dev_2, $E_{PK1}(SSK)$, sign$_{PR2}$(hejhej, PK_2, Dev_2, $E_{PK1}(SSK)$))
4. Signature verification
5. $E_{SSK}(snd\_packet_1)$
6. $E_{SSK}(snd\_packet_2)$

In step *4*, Dev_1 identifies that the message originates from Dev_2 by referencing the device name and verifies the message's integrity by decrypting the signature using Dev_2's public key. With this signature verification completed, the key exchange part of the protocol concludes, and both parties agree on the *SSK* which will be utilized for encrypting subsequent communications. Subsequently, in message *5*, $Dev1$ transmits a message known as the *snd_packet* to Dev_2, signaling its connection request. As illustrated in Listing 3.1, *snd_packet*1 is a message in which Dev_1 encrypts and signs the custom message, payload, and device name using its private key. The encryption is performed using the agreed-upon *SSK* from the key exchange phase, employing the AES algorithm in *CBC* mode. Since Dev_2 possesses knowledge of the *SSK* and $Dev1$'s public key, it verifies the message's integrity by checking the signature. After successful verification, Dev_2 can decrypt the message by using *SSK*. Upon successful verification, Dev_2 can decrypt the message using the *SSK*. If the content of Dev_1's *snd_packet* message satisfies the neighbor criterion, Dev_2 responds by sending its own *snd_packet* message which includes a custom message, payload, and device name, encrypted and signed with its private key using the pre-established *SSK*. Similar to Dev_1, Dev_2 uses the predetermined *SSK* for encryption. Finally, Dev_2 sends message *6* back to Dev_1, signaling the successful establishment of the connection. All parameter definitions utilized in the protocol description are outlined in Table 3.1.

Listing 3.1: Example snd_packet message in JSON format

```
{
  "msg": "snd_packet",
  "payload": {
    "msg": "hello, I am Dev1",
  },
  "node_name": "Dev1",
}
```

hej: custom message sent by Device 1	hejhej: custom message sent by Device 2
PK_1: Dev_1's public key	PK_2: Dev_2's public key
Dev_1: Device 1	Dev_2: Device 2
PR_1: Dev_1's private key	PR_2: Dev_2's private key
$sign_{PR1}(...)$: sign with Dev_1's private key	$sign_{PR2}(...)$: sign with Dev_2's private key
SSK: Shared Secret Key	
$E_{PK1}(...)$: encrypt with Dev_1's public key	
snd_packet_1: Dev_1's custom message	snd_packet_1: Dev_2's custom message
$E_{SSK}(...)$: encrypt with the *SSK*	

Table 3.1: Parameters in Basic Authentication Protocol

It's essential to acknowledge that this protocol has been intentionally designed to explore and understand the potential vulnerabilities, including the risks associated with impersonation and MitM attacks. By intentionally leaving these aspects uncovered, we can gain insights into the potential security challenges that may arise in such scenarios and explore the importance of SND. As an example, a MitM attack is designed in Chapter 4.1.2 for this protocol, exploiting the lack of security features. The attacker can modify the information on the messages and relay them, making it possible to impersonate any other device.

3.3 Authentication with ECDHE

Following the design and development of our first authenticated SND protocol, denoted as *Basic Authentication Protocol* in Chapter 3.2, substantial enhancements are implemented to fortify the protocol's resilience against potential vulnerabilities before using the predetermined time-based, location-based and time-location-based SND protocols.

This protocol includes five major changes compared to the previous one as listed below:

- We change the public key cryptography choice from *RSA* to *ECC* while creating digital signatures and performing the key exchange. Specifically, we use *EdDSA* as a digital signature scheme. As a result of this change, we change the key exchange algorithm from classic DH to ECDH.
- The protocol does not contain any encryption with public keys. In other words, all the encryptions are accomplished by a *SSK*.
- The mode of the encryption algorithm is changed from *CBC* to *GCM*.
- We start to use *HMAC* which provides authenticity protection and integrity protection.
- To ensure perfect forward secrecy (PFS), the utilization of ephemeral public keys is incorporated in the generation of the *SSK*. Consequently, unique random ECDH point multipliers are generated for each key exchange and subsequently discarded. As a result, the key exchange algorithm transitions to ECDH ephemeral (ECDHE).

Figure 3.3 provides an overview of the authentication with ECDHE and the same assumptions for the previous protocol apply here as well. Although the first messages of both protocols seem the same, there is a slight change in the first message where we use *EdDSA* instead of *RSA* while signing the message which provides better security. When Dev_2 receives the message *1* as depicted in Figure 3.3, it performs different steps than the first protocol. First, Dev_2 verifies the signature and then creates a *SSK*. After that, it uses the *SSK* to create a *HMAC* of a custom $hello$ message. As Dev_1, Dev_2 creates a similar message where the message includes the custom word *hejhej*, a public key of Dev_2, and the name of

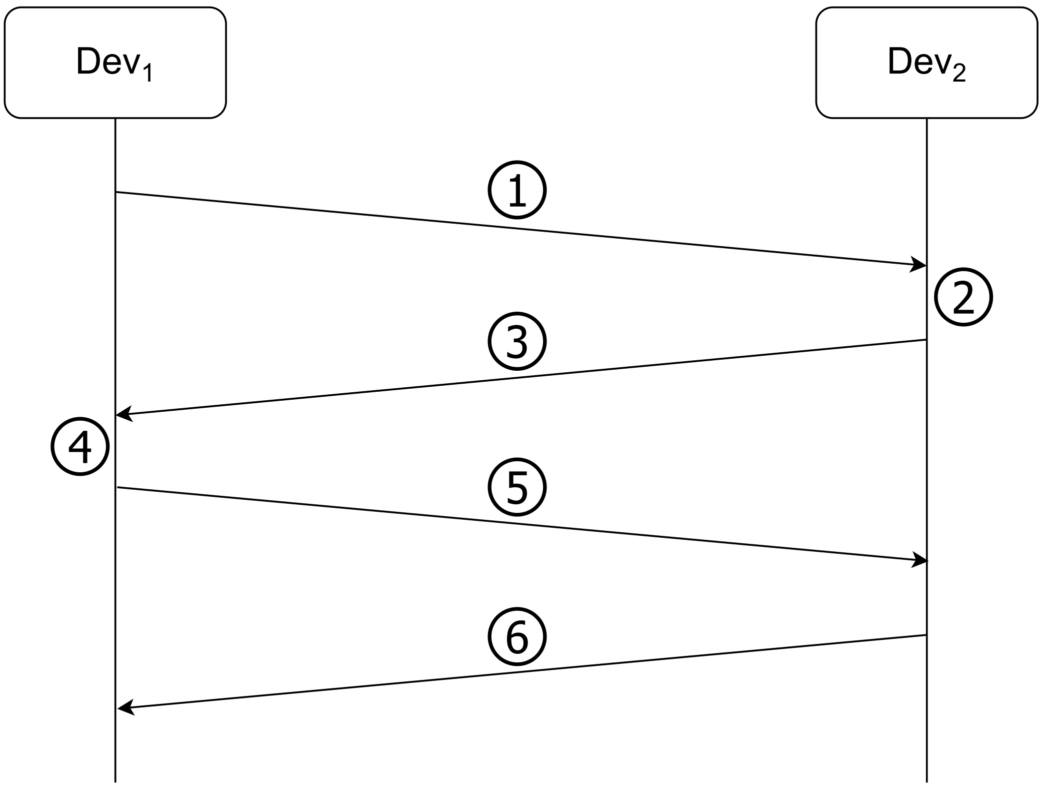


Figure 3.3: Authentication with ECDHE Protocol Diagram

the Dev_2. Thereafter, Dev_2 signs the above-mentioned message along with $HMAC$ and sends it to Dev_1 as message *3*.

The protocol description is as follows:

1. hej, PK_1, Dev_1, EPH_{PK_1}, $sign_{PR1}$(hej, PK_1, Dev_1, EPH_{PK_1})
2. Signature verification
3. hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$, $sign_{PR2}$(hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$)
4. Following operations are performed by Dev_1:
   (a) Signature verification
   (b) $R =$ compare($HMAC_{SSK_1}(hello)$, $HMAC_{SSK_2}(hello)$). If $R == false$, then disconnect.
5. $E_{SSK}(snd\_packet_1)$
6. $E_{SSK}(snd\_packet_2)$

As soon as Dev_1 receives the message *3* from Dev_2, the first operation that Dev_1 does is to verify the signature and then calculate the *HMAC* of the message *hello* by using the *SSK*. Here, we define two different *SSK*s, namely SSK_1 and SSK_2, to emphasize the comparison. Unless there is a miscalculation on both sides, Dev_1 and Dev_2 are able to end up with the same *SSK* value which makes the $comparison(c1,c2)$ function return true. If the calculated *HMAC* and received *HMAC* are not the same, then Dev_1 stops the connection. If they are the same, then Dev_1 uses the *SSK* to encrypt the custom *snd_packet* message. The difference in the *snd_packet* is the inclusion of MAC address as can be seen in Listing 3.2. As previously mentioned, we have transitioned from using the *CBC* mode to *GCM* in our encryption algorithm. One of the primary motivations for this shift is that *GCM* offers authenticated encryption, ensuring both confidentiality and integrity of the data. *GCM* is specified in NIST SP 800-38D and only operates with a 128-bit cipher in the Pycryptodome Python package [25].

Listing 3.2: Example snd_packet message in JSON format

```
{
  "msg": "snd_packet",
  "payload": {
    "msg": "hello, I am Dev1",
    "mac": "00-B0-D0-63-C2-26"
  },
  "node_name": "Dev1",
}
```

hej: custom message sent by Device 1	hejhej: custom message sent by Device 2
PK_1: Dev_1's public key	PK_2: Dev_2's public key
Dev_1: Device 1	Dev_2: Device 2
EPH_{PK_1}: Dev_1's ephemeral public key	EPH_{PK_2}: Dev_2's ephemeral public key
PR_1: Dev_1's private key	PR_2: Dev_2's private key
$sign_{PR1}(...)$: sign with Dev_1's private key	$sign_{PR2}(...)$: sign with Dev_2's private key
msg: any custom message that will be used in HMAC operation	
$HMAC_{KEY}(msg)$: HMAC the *msg* with the *KEY*	
SSK: Shared Secret Key	
$compare(c1,c2)$: compares c1 and c2	
snd_packet_1: Dev_1's custom message	snd_packet_1: Dev_2's custom message
$E_{SSK}(...)$: AES authenticated encryption in GCM mode with *SSK*	

Table 3.2: Parameters in Authentication with ECDHE

When Dev_2 receives the message *5* from Dev_1, it first validates the message by checking the *HMAC* and then decrypts the SND_1 message. Lastly, Dev_1 follows a similar procedure as Dev_2 does in the previous step: Dev_1 receives the message *6* from Dev_2 where Dev_1 first needs to validate the message by checking the *HMAC* and then decrypts the SND_2 message by using the predetermined *SSK*. If the *HMAC* validation and decryption processes proceed without issues, both devices establish mutual neighbor status. All parameter definitions used in describing the protocol can be found in Table 3.2.

Even with the usage of *HMAC*, signatures, ephemeral cryptography and session key, it is still not enough to make the protocol immune to an active MitM attack. In a scenario as described in Figure 4.4 and Figure 4.5, where the messages are modified by the adversary and then relayed, it is possible to break the protocol by modifying the values sent by Dev_1 and using a different set of keys in the message, as described in Chapter 4.1.3.

3.4 Time-based Authentication with ECDHE

Following the elucidation of the five major modifications as described in Chapter 3.3, it is inferred that a sufficiently fortified foundation has been established for the first predetermined protocol, namely time-based SND. By leveraging and extending the groundwork of the preceding protocol, we decide to name this protocol as the *Time-Based Authentication Protocol with ECDHE*. The naming explicitly denotes the protocol's hierarchical evolution and signifies its augmented functionalities derived from its precursor.

Before diving into the significant changes in this protocol, it is essential to mention that there have not been any changes in the choices of cryptographic operations. On the other hand, this protocol exhibits two notable deviations in comparison to the previous one, enumerated as follows:

- Inclusion of Time of Flight (ToF): Both devices calculate ToF by multiplying the speed of light with the time difference between sending message *1* and receiving the message *2*, and dividing the whole result by 2. Following this, the preceding outcome is subjected to multiplication by the Securitas Constant, formulated within the scope of this study. Subsequently, a pre-defined threshold is utilized to conduct a check with ToF.

$$ToF = \frac{(C \times \Delta T)}{2} \times S \tag{3.1}$$

  where,

  - C: speed of light which approximately equals to 3×10^8.

  - ΔT: time difference between sending message *1* and receiving message *2*.
  - S: Securitas Constant which is equal to $6.6 \times 10^{-}6$.

- Usage of local clocks: Both devices use their own local clock to measure ToF for particular messages. Notably, the synchronization of two clocks' accuracy is not imperative for this specific protocol. Each device autonomously manages its ToF calculation which removes the necessity of clock synchronization, and compares ToF with the predetermined threshold.

Figure 3.4 provides an outline of the time-based authentication protocol with ECDHE, with analogous assumptions applying here as in the preceding protocols as well. The first two messages are the same as in the previous protocol but as an additional step, Dev_1 starts its local clock when it sends the message *1* and Dev_2 does the same when it sends the message *3*. As soon as Dev_1 receives the message *4*, it first verifies the signature and then stops the local clock to calculate the ToF for the messages *2* and *4*. Then, Dev_1 compares the calculated and received $HMAC$ values.

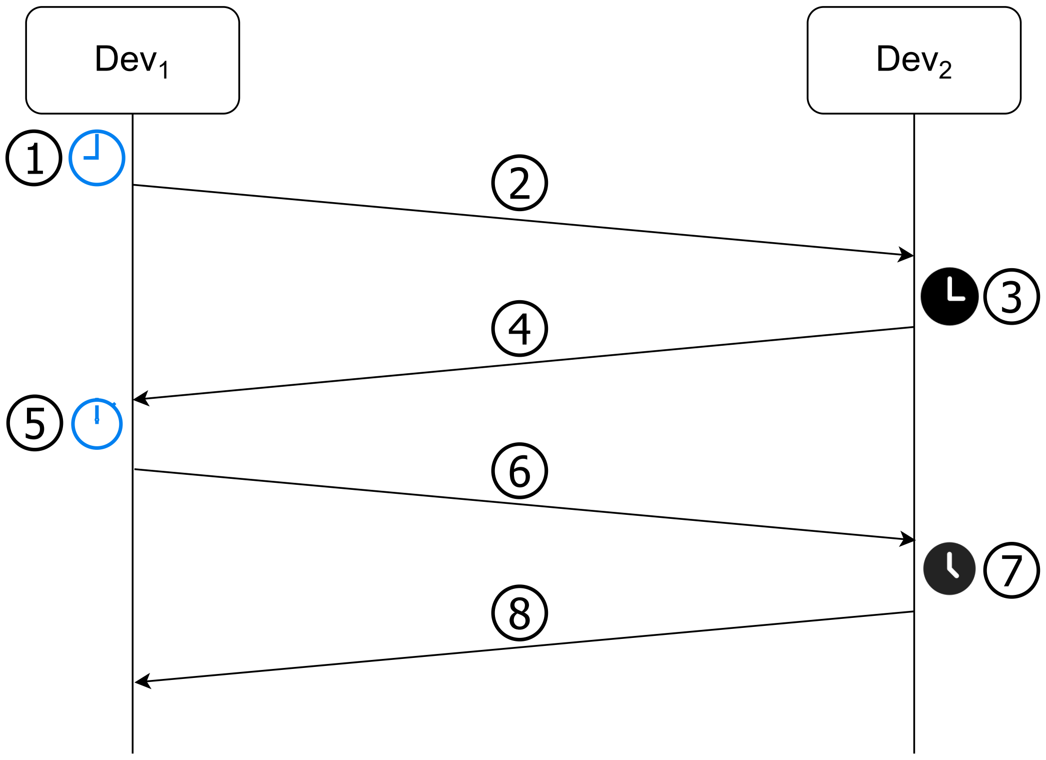


Figure 3.4: Time-based Authentication Protocol with ECDHE Diagram

The protocol description is as follows:

1. Timer initiation
2. hej, PK_1, Dev_1, EPH_{PK_1}, $sign_{PR1}$(hej, PK_1, Dev_1, EPH_{PK_1})
3. Following operations are performed by Dev_2:
   (a) Timer initiation
   (b) Signature verification
4. hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$, $sign_{PR2}$(hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$)
5. Following operations are performed by Dev_1:
   (a) Signature verification
   (b) Timer halt. If $ToF_1 > Range$, then disconnect.
   (c) $R =$ compare($HMAC_{SSK_1}(hello)$, $HMAC_{SSK_2}(hello)$). If $R == false$, then disconnect.
6. $E_{SSK}(SND_1)$
7. Timer halt. If $ToF_2 > Range$, then disconnect.
8. $E_{SSK}(SND_2)$

As the last and also additional operation compared to the previous protocol in step *5*, Dev_1 compares the calculated *ToF* value with the predetermined *Range* value and terminates the connection if the *ToF* value is bigger than the *Range* value. If not, Dev_1 continues the connection by sending message *6*. When Dev_2 receives the message *6*, it stops its local clock and calculates *ToF* for the messages *4* and *6* in step *7*. If the *ToF* value is less than the predetermined *Range* value, the protocol continues and Dev_2 sends the message *8*. Otherwise, Dev_2 terminates the connection since one of its prerequisites is not met. Upon receiving the message *6*, Dev_1 promptly conducts an integrity check by verifying the *HMAC* and endeavors to decrypt the message content to retrieve the snd_packet_2 message. Should no operation encounter a failure at this stage, both devices mutually confirm their status as neighbors. All parameter definitions used to describe the protocol are available in Table 3.3.

Furthermore, it is recognized that assuming a noise-free network is impractical. Consequently, a decision has been made to simulate network noise, introducing two key concepts: noise level and channel reliability level. It is important to note that the noise level and channel reliability level exhibit an inverse relationship, with the channel reliability level reaching its minimum when the noise level is at its maximum. In real-world scenarios, the noise level corresponds to any disruptions in the network that may cause delays in packet transmission between devices.

To execute the noise simulation, two parameters, namely the low and high values of the variable *noise_strength* in milliseconds, are defined. Subsequently, a random number within this range is generated using the *uniform()* function of the *random* module. In the final step, all executions of the corresponding thread are delayed for a duration equal to the aforementioned random number of seconds. This approach ensures a realistic simulation of network noise, contributing to a more accurate representation of network conditions.

hej: custom message sent by Device 1	hejhej: custom message sent by Device 2
PK_1: Dev_1's public key	PK_2: Dev_2's public key
Dev_1: Device 1	Dev_2: Device 2
EPH_{PK_1}: Dev_1's ephemeral public key	EPH_{PK_2}: Dev_2's ephemeral public key
PR_1: Dev_1's private key	PR_2: Dev_2's private key
$sign_{PR1}(...)$: sign with Dev_1's private key	$sign_{PR2}(...)$: sign with Dev_2's private key
msg: any custom message that will be used in HMAC operation	
$HMAC_{KEY}(msg)$: HMAC the msg with the KEY	
SSK: Shared Secret Key	
ToF_1: Time of Flight for Dev_1	ToF_2: Time of Flight for Dev_2
$Range$: A threshold for distance in meters	
$compare(c1, c2)$: compares c1 and c2	
snd_packet_1: Dev_1's custom message	snd_packet_1: Dev_2's custom message
$E_{SSK}(...)$: AES authenticated encryption in GCM mode with SSK	

Table 3.3: Parameters in Time-based Authentication Protocol with ECDHE

The time-based authentication protocol is still not secure, and it is vulnerable to MitM attacks as can be seen in Chapter 4.1.4. The messages can be intercepted and modified to make Dev_1 believe it is communicating with Dev_2 even if that is not the case. Nevertheless, the ToF validation adds another security layer to the protocol and protects against attackers who are trying to impersonate slow-responding devices or who don't have equipment fast enough to complete the attack.

3.5 Location-based Authentication with ECDHE

Upon the successful implementation of the *Time-Based Authentication Protocol with ECDHE*, the subsequent step involves proceeding with the second predetermined protocol: *Location-Based Authentication Protocol with ECDHE*. Unlike the previous protocol, there have been modifications introduced to the message content in this protocol.

It is worth noting that we concentrate solely on outlining the distinctions between this protocol and its predecessor. For example, the first four steps remain unchanged compared to the previous protocol; thus, we omit the detailed description of these steps. As depicted in 3.5, when Dev_2 receives the message *5*, it calculates the Euclidian distance, which is denoted as D_{12} by using the latitude-longitude values that it receives and its own latitude-longitude values. If the value of D_{12} exceeds the predetermined *Range*, the protocol execution is terminated by Dev_2, assuming that Dev_1 is outside the range.

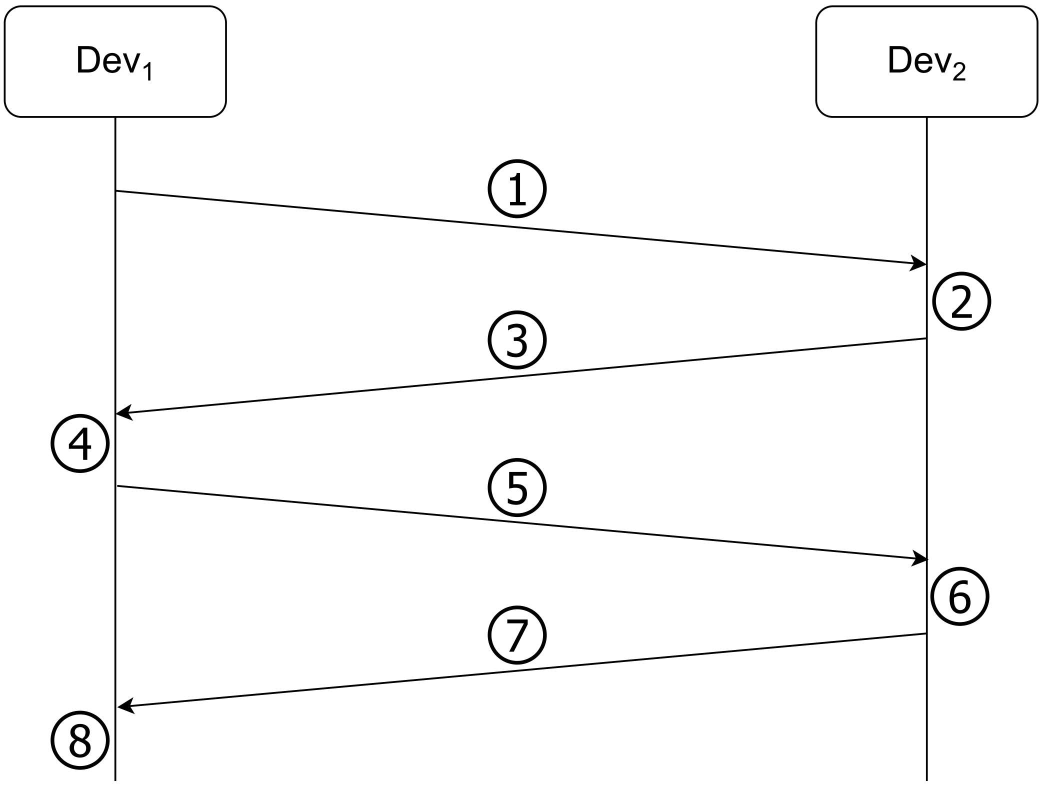


Figure 3.5: Location-based Authentication Protocol with ECDHE Diagram

The protocol description is as follows:

1. hej, PK_1, Dev_1, EPH_{PK_1}, $sign_{PR1}$(hej, PK_1, Dev_1, EPH_{PK_1})
2. Signature verification
3. hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$, $sign_{PR2}$(hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$)

4. Following operations are performed by Dev_1:
   (a) Signature verification
   (b) $R = \text{compare}(HMAC_{SSK_1}(hello), HMAC_{SSK_2}(hello))$. If $R == false$, then disconnect.
5. $E_{SSK}(SND_1)$
6. If $D_{12} > Range$, then disconnect.
7. $E_{SSK}(SND_2)$
8. If $D_{21} > Range$, then disconnect.

If the value of D_{12} does not exceed the predetermined *Range* value, then Dev_2 sends its latitude and longitude information, denoted as lat_2 and lng_2, respectively, as the last message which is message *7*. In step *8*, Dev_1 performs the same operations as Dev_2 and calculates the Euclidian distance, namely D_{21}. We utilize two different representations for the Euclidean distances computed by each device, denoted as D_{12} and D_{21}, respectively. This distinction enables the detection of any potential changes in the locations of both devices during the protocol execution although we are aware of the minimal probability of significant alterations occurring in the locations of both devices between messages *5* and *7*. A minor modification in the *snd_packet* messages from the previous protocol involves the inclusion of latitude and longitude fields as *lat* and *lng* as can be seen in Listing 3.3. As the first four steps mirror those of the previous two protocols, the *Location-Based Authentication Protocol with ECDH* adheres to a similar procedure as its predecessors, determining its neighbor identically. All protocol parameter definitions are listed in Table 3.4.

Listing 3.3: Example SND message in JSON format

```
{
  "msg": "snd_packet",
  "payload": {
    "msg": "hello, I am Dev1",
    "mac": "00-B0-D0-63-C2-26"
  },
  "node_name": "Dev1",
  "lat": "38.8951",
  "lng": "-77.0364"
}
```

hej: custom message sent by Device 1	hejhej: custom message sent by Device 2
PK_1: Dev_1's public key	PK_2: Dev_2's public key
Dev_1: Device 1	Dev_2: Device 2
EPH_{PK_1}: Dev_1's ephemeral public key	EPH_{PK_2}: Dev_2's ephemeral public key
PR_1: Dev_1's private key	PR_2: Dev_2's private key
$sign_{PR1}(...)$: sign with Dev_1's private key	$sign_{PR2}(...)$: sign with Dev_2's private key
msg: any custom message that will be used in HMAC operation	
$HMAC_{KEY}(msg)$: HMAC the msg with the KEY	
SSK: Shared Secret Key	
$compare(c1, c2)$: compares c1 and c2	
snd_packet_1: Dev_1's custom message	snd_packet_1: Dev_2's custom message
$E_{SSK}(...)$: AES authenticated encryption in GCM mode with SSK	
lat_1: Latitude of Dev_1	lat_2: Latitude of Dev_2
lng_1: Longitude of Dev_1	lng_2: Longitude of Dev_2
D_{ab}: Euclidean distance from (lat_a, lng_a) to (lat_b, lng_b)	

Table 3.4: Parameters in Location-based Authentication Protocol with ECDHE

Even with the location-based authentication layer the protocol is still not secure enough, and it is vulnerable to MitM attacks. The messages can be intercepted and modified as described in Chapter 4.1.5. The coordinates can be spoofed and modified to the attacker's convenience. Nevertheless, the *Range* validation adds another security layer to the protocol and protects against attackers who are not aware of the *Range* value and are out of range.

3.6 Time-location-based Authentication with ECDHE

After the successful implementation of the *Location-based Authentication Protocol with ECDHE*, the subsequent phase entails the progression to the last and third predefined protocol: *Time-location-based Authentication Protocol with ECDHE*. This protocol can be perceived as a synthesis of both the time-based and location-based protocols.

As delineated in Section 3.5, our focus is again exclusively directed towards emphasizing the differences between the current protocol and its predecessor. Therefore, we omit the explanation of message components that remain unchanged from the preceding protocol, that is *Location-based Authentication Protocol with ECDHE*. The content within message *2* remains identical to that of the previous protocol. However, the singular distinction arises at the Dev_1 side, initiating its local clock immediately right before sending the message *2*, mirroring the behavior outlined in the *Time-based Authentication Protocol with ECDHE*. Similarly, Dev_2 does the same right before it sends the message *4* which is depicted in Figure 3.6 with blue clock sign for Dev_1 and black for Dev_2. Another distinction manifests in step *5*. Upon reception of message *4*, Dev_1 first verifies the signature, stops its local clock, and subsequently computes ToF. A similar process applies to Dev_2, whereupon it halts its local clock right after receiving the message *6* and proceeds with the calculation of the ToF.

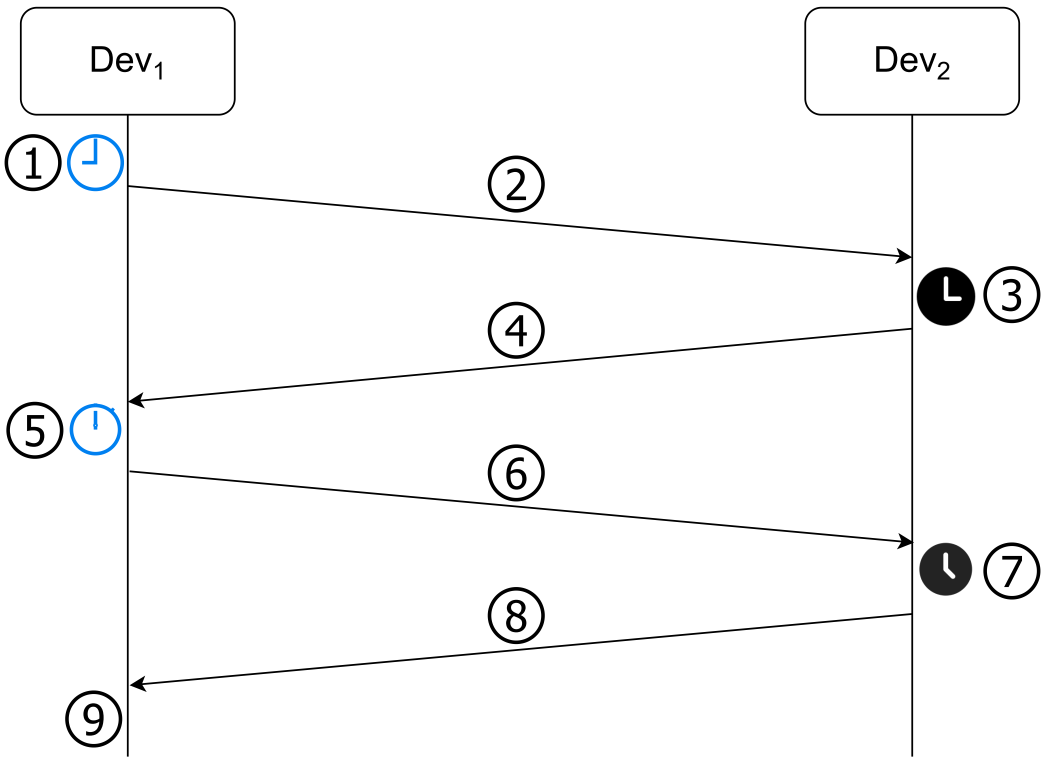


Figure 3.6: Time-location-based Authentication Protocol with ECDHE

The protocol description is as follows:

1. Timer initiation
2. hej, PK_1, Dev_1, EPH_{PK_1}, $sign_{PR1}$(hej, PK_1, Dev_1, EPH_{PK_1})
3. Following operations are performed by Dev_2:
   (a) Timer initiation
   (b) Signature verification
4. hejhej, Dev_2, PK_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$, $sign_{PR2}$(hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$)

5. Following operations are performed by Dev_1:
   (a) Signature verification
   (b) Timer halt. If $ToF_1 > Range$, then disconnect.
   (c) $R = \text{compare}(HMAC_{SSK_1}(hello), HMAC_{SSK_2}(hello))$. If $R == false$, then disconnect.
6. $E_{SSK}(SND_1)$
7. Following operations are performed by Dev_2:
   (a) Timer halt. If $ToF_2 > Range$, then disconnect.
   (b) If $D_{12} > Range$, then disconnect.
8. $E_{SSK}(SND_2)$
9. If $D_{21} > Range$, then disconnect.

All other message contents, verifications, and comparisons not specified earlier remain unchanged compared to the previous protocol. Thus, *Time-location-based Authentication Protocol with ECDHE* adheres to a similar procedure as its predecessor in identifying its neighbors, except for the alterations detailed in the previous paragraph. Regarding the details of the *ToF* calculation, readers are advised to consult Chapter 3.4. Similarly, for the details of the distance calculation, reference to Chapter 3.5 is recommended. All parameter definitions utilized in the protocol description are outlined in Table 3.5.

The time-location-based authentication protocol is more secure than the previous iterations, time-based and location-based authentication, but it is still not secure enough. It is still vulnerable to certain MitM attacks, as described in Chapter 4.1.6. When an attacker uses coordinates that are within the *Range* of the protocol parameters and the responses are within the accepted *ToF*, the messages can be modified and relayed which makes the impersonation of other devices possible.

3.7 GeoTimeCert Authentication with ECDHE

Having bolstered the security of the preceding protocol through the inclusion of time and location data, our objective is to progress toward the development of a highly robust protocol. This enhanced protocol aims to proactively address and mitigate a comprehensive spectrum of attacks, particularly MitM attacks. To achieve this goal, we introduce nonces and certificates. The quest for this protocol is motivated by the imperative to safeguard sensitive data and critical communications in an ever-evolving and increasingly hostile digital landscape.

hej: custom message sent by Device 1	hejhej: custom message sent by Device 2
PK_1: Dev_1's public key	PK_2: Dev_2's public key
Dev_1: Device 1	Dev_2: Device 2
EPH_{PK_1}: Dev_1's ephemeral public key	EPH_{PK_2}: Dev_2's ephemeral public key
PR_1: Dev_1's private key	PR_2: Dev_2's private key
$sign_{PR1}(...)$: sign with Dev_1's private key	$sign_{PR2}(...)$: sign with Dev_2's private key
msg: any custom message that will be used in HMAC operation	
$HMAC_{KEY}(msg)$: HMAC the msg with the KEY	
SSK: Shared Secret Key	
ToF_1: Time of Flight for Dev_1	ToF_2: Time of Flight for Dev_2
$Range$: A threshold for distance in meters	
$compare(c1,c2)$: compares c1 and c2	
snd_packet_1: Dev_1's custom message	snd_packet_1: Dev_2's custom message
$E_{SSK}(...)$: AES authenticated encryption in GCM mode with SSK	
lat_1: Latitude of Dev_1	lat_2: Latitude of Dev_2
lng_1: Longitude of Dev_1	lng_2: Longitude of Dev_2
D_{ab}: Euclidean distance from (lat_a, lng_a) to (lat_b, lng_b)	

Table 3.5: Parameters in Time-location-based Authentication Protocol with ECDHE

3.7.1 Certificates

In our pursuit of adhering to best practices and minimizing vulnerabilities within this robust protocol, we have opted to adhere to the public key infrastructure (PKI) system, which enables secure data exchange among authenticated entities over the Internet. As a team, we face a pivotal decision regarding whether to utilize a well-established third-party Certificate Authority (CA), such as Let's Encrypt, or to develop our own CA for the sake of simplicity and proof-of-concept. Ultimately, we choose to proceed with our own CA implementation, encompassing fundamental CA functionalities such as issuing certificates for Certificate Signing Requests (CSR) and verifying certificates. Consequently, we establish a rudimentary CA HTTP server designed to manage X.509 certificates for the aforementioned basic functionalities. It is noteworthy within the context of PKI that the generation of a CSR is a prerequisite before requesting and obtaining a certificate [26]. Once the CSR is generated, the CA can utilize the information contained within it to issue the corresponding certificate.

Certificate support is conditional upon the use of ECDHE. In other words, certificates can only be utilized when ECDHE is employed; they are not compatible with our current RSA implementation. Before establishing any connections, all devices are required to generate their CSRs if the current certificates are not found. For example, both Dev_1 and Dev_2 generate their respective certificates using our custom CA. Subsequently, Dev_1 transmits its certificate to Dev_2 in message *2*. Upon receiving message *2*, Dev_2 validates the certificate received from Dev_1 using our custom CA and then forwards its own certificate to Dev_1. As the final step concerning certificates, Dev_1 verifies the validity of Dev_2's certificate. Messages containing certificates are denoted by *certificate document sign* in Figure 3.7.

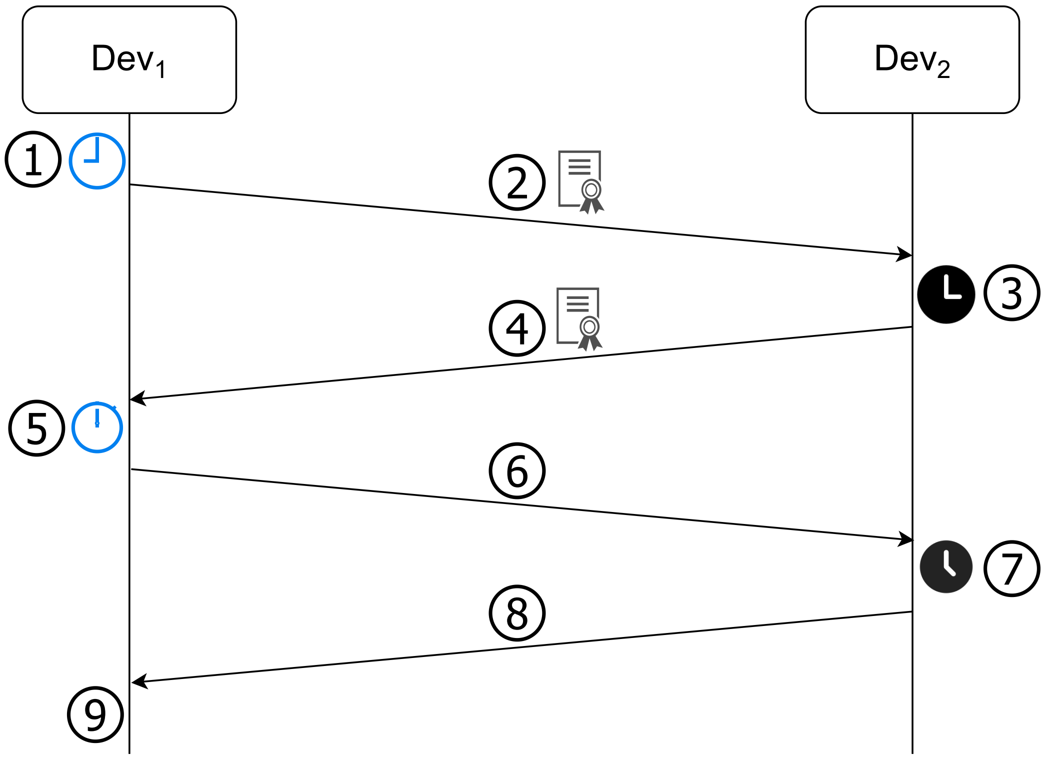


Figure 3.7: GeoTimeCert authentication protocol with ECHDE Diagram

3.7.2 Nonces

As previously discussed, two significant enhancements, namely nonces and certificates, distinguish the current protocol from the preceding three protocols where our focus is solely on these modifications. As the first step, Dev_1 initiates the process by creating a nonce, denoted as N_1, which involves concatenating 16 bytes of a randomly generated hexadecimal universally unique identifier (UUID) with the current time in seconds since the Epoch. Subsequently, Dev_1 incorporates this time-dependent nonce, N_1, into message *2* as depicted in 3.7 along with all other previously introduced information from the last three protocols. In step *3*, Dev_2 employs a similar process to generate its nonce, denoted as N_2, and subsequently incorporates N_2 into message *4* along with all other previously introduced information from the last three protocols. Upon receiving message *4*, Dev_1 conducts an additional check to verify the freshness of the received nonce, N_2, in addition to the existing checks outlined in the preceding protocol. The freshness of the received nonce is done as follows:

1. Calculate the difference between the current time and the time indicated in the nonce. The time in the nonce refers to the segment that begins after the initial 16 bytes and continues until the end of the nonce.

2. If the calculated difference exceeds the predefined expiration time, which is set at 120 seconds, the nonce is considered as expired; otherwise, it is checked in the local database. If not found, it is considered valid and stored in the local database of the corresponding device.

3. A separate thread is responsible for removing nonces from the local database that are older than the current time by the predefined expiration time. This process is essential for managing the local database size and preventing it from becoming excessively large.

If the nonce has expired or it is found in the local database, Dev_1 discontinues its communication and terminates the protocol. Similarly, in step *7*, Dev_2 conducts identical checks and discontinues communication if the nonce provided by Dev_1 has expired. Despite the nonce being dependent on randomly generated numbers and local time, both devices verify the received nonce in their local databases to ascertain whether it has been previously used. Also, both devices use their local clock to measure time difference as we do in the *Time-based Authentication Protocol with ECDHE* to avoid clock synchronization. If the device finds the received nonce in its local database, it terminates the protocol, as a nonce should not be reused, constituting a replay attack. To illustrate practical and demonstration purposes, each device removes nonces from its local database if they are older than 120 seconds.

3.7.3 Protocol

Figure 3.7 offers a comprehensive depiction of the secure authentication protocol using ECDHE. Similar to previous protocols, our assumptions include the presence of two legitimate devices capable of establishing a connection within a reasonable range. Furthermore, we assume that Dev_1 intends to initiate a connection with Dev_2.

The protocol description is as follows:

1. Timer initiation
2. hej, $Cert_1$, Dev_1, EPH_{PK_1}, N_1, $sign_{PR1}$(hej, $Cert_1$, Dev_1, EPH_{PK_1}, N_1)
3. Following operations are performed by Dev_2:
   (a) Timer initiation
   (b) Certificate verification
   (c) Signature verification
4. hejhej, $Cert_2$, Dev_2, EPH_{PK_2}, N_1, N_2, $HMAC_{SSK_2}(hello)$, $sign_{PR2}$(hejhej, $Cert_2$, Dev_2, EPH_{PK_2}, N_1, N_2, $HMAC_{SSK_2}(hello)$)
5. Following checks are conducted by Dev_1:
   (a) Certificate verification
   (b) Signature verification
   (c) Timer halt. If $ToF_1 > Range$, then disconnect.
   (d) Nonce freshness check
   (e) $R =$ compare($HMAC_{SSK_1}(hello)$, $HMAC_{SSK_2}(hello)$). If $R == false$, then disconnect.
6. $E_{SSK}(SND_1)$
7. Following checks are conducted by Dev_1:
   (a) Timer halt. If $ToF_2 > Range$, then disconnect.
   (b) Nonce freshness check
   (c) If $D_{12} > Range$, then disconnect.
8. $E_{SSK}(SND_2)$
9. If $D_{21} > Range$, then disconnect.

All other message contents, verifications, and comparisons that were not previously specified remain unaltered from the previous protocol. Therefore, the *GeoTimeCert Authentication with ECDHE* follows a comparable procedure to its predecessor in identifying its neighbors, except for the modifications outlined in the preceding paragraphs. For a comprehensive understanding of the *ToF* calculation, readers are encouraged to refer to Chapter 3.4. Likewise, for detailed information on the distance calculation, it is recommended to consult Chapter 3.5. Additionally, all protocol parameter definitions are listed in Table 3.6.

Listing 3.4: Example SND message in JSON format

```
{
  "msg": "snd_packet",
  "nonce_a": "3479
     ↪ cc8251314f17bf6ac1512a15c4ea1704184122",
  "payload": {
    "msg": "hello, I am Dev1",
    "mac": "00-B0-D0-63-C2-26"
  },
  "node_name": "Dev1",
  "lat": "38.8951",
  "lng": "-77.0364"
}
```

hej: custom message sent by Device 1	hejhej: custom message sent by Device 2
PK_1: Dev_1's public key	PK_2: Dev_2's public key
Dev_1: Device 1	Dev_2: Device 2
EPH_{PK_1}: Dev_1's ephemeral public key	EPH_{PK_2}: Dev_2's ephemeral public key
N_1: nonce generated by Dev_1	N_2: nonce generated by Dev_2
PR_1: Dev_1's private key	PR_2: Dev_2's private key
$sign_{PR1}(...)$: sign with Dev_1's private key	$sign_{PR2}(...)$: sign with Dev_2's private key
msg: any custom message that will be used in HMAC operation	
$HMAC_{KEY}(msg)$: HMAC the msg with the KEY	
SSK: Shared Secret Key	
ToF_1: Time of Flight for Dev_1	ToF_2: Time of Flight for Dev_2
$Range$: A threshold for distance in meters	
$compare(c1, c2)$: compares c1 and c2	
snd_packet_1: Dev_1's custom message	snd_packet_1: Dev_2's custom message
$E_{SSK}(...)$: AES authenticated encryption in GCM mode with SSK	
lat_1: Latitude of Dev_1	lat_2: Latitude of Dev_2
lng_1: Longitude of Dev_1	lng_2: Longitude of Dev_2
D_{ab}: Euclidean distance from (lat_a, lng_a) to (lat_b, lng_b)	

Table 3.6: Parameters in GeoTimeCert Authentication Protocol with ECDHE

Chapter 4

Scenario and Threat Model

As mentioned in earlier sections, the goal of ND is to connect neighbor devices, this is why the evaluated scenario in this project consists of two devices, within communication range of each other, trying to initiate a connection to become neighbors. One device (Dev_1) initiates the communication by sending the first message to the other device (Dev_2) which responds accordingly to the received message. The described scenario can be seen in Figure 4.1.

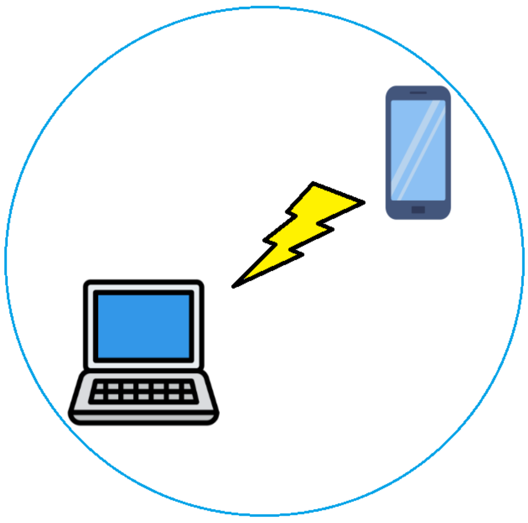


Figure 4.1: ND scenario

4.1 Man-in-the-Middle Attack

When deploying ND protocols in public networks, the main risk comes with potential hacker attacks, such as address spoofing attacks, denial-of-service, and Man-in-the-Middle attacks[5]. To evaluate the security of the designed protocols, a series of MitM attacks are prepared, such an attack occurs when a malicious actor positions themselves between two authentic devices, effectively eavesdropping on, injecting data into, or intercepting the communication between them as depicted in Figure 4.2.

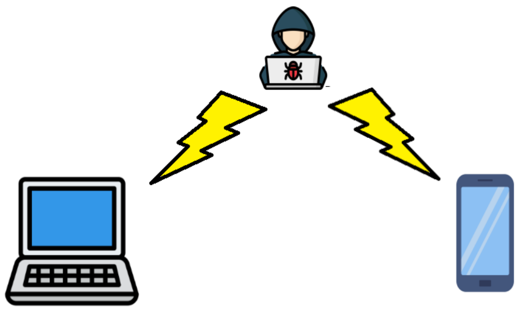


Figure 4.2: Passive MitM attack scenario

The attacker is linked to both devices and acts as an intermediary, facilitating the exchange of data between them. Despite appearances, these legitimate devices seem to communicate directly, but in reality, they are communicating through an intermediary third-party device [27].

4.1.1 Passive MitM Attack

The first attack is designed as a passive MitM attack. This is a common threat to all of the protocols since the messages can always be relayed to other devices. This attack is only effective when all of the devices are within the connectivity range of each other, including time and location when applicable. As Figure 4.3 shows, the passive MitM attack makes no changes in the intercepted communication and only relays the messages to the other device.

The message structure of a passive MitM attack for the basic authentication protocol is as follows:

1. hej, PK_1, Dev_1, sign$_{PR1}$(hej, PK_1, Dev_1)
2. hejhej, PK_2, Dev_2, $E_{PK1}(SSK)$, sign$_{PR2}$(hejhej, PK_2, Dev_2, $E_{PK1}(SSK)$))
3. $E_{SSK}(snd\_packet_1)$
4. $E_{SSK}(snd\_packet_2)$

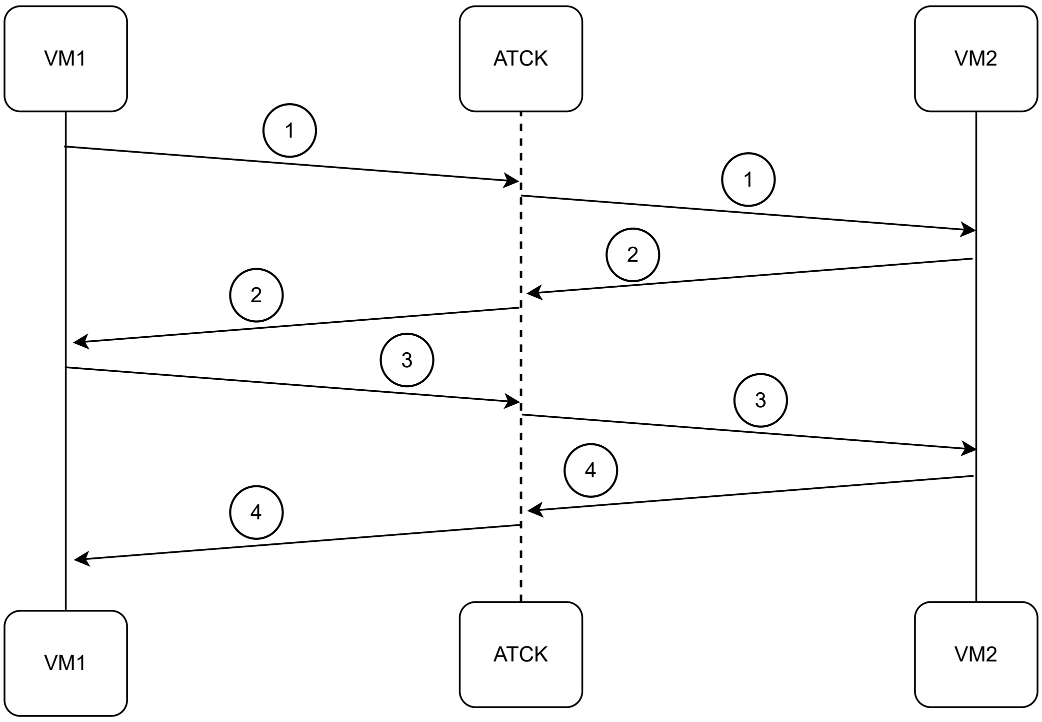


Figure 4.3: Messages relayed in a passive MitM attack

4.1.2 Active MitM Attack: Basic authentication

Figure 4.4 shows an active MitM attack designed against the basic authentication protocol where an adversary not only relays the messages but also opens the messages and changes the information so that the responses from Dev_2 can be decrypted and changed at the adversary's convenience.

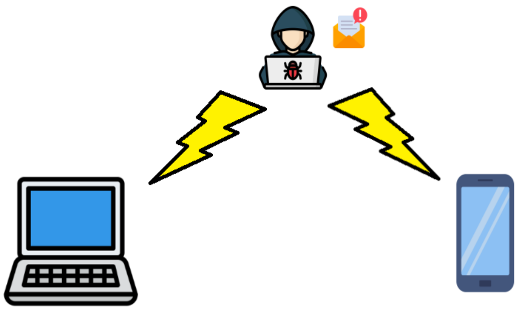


Figure 4.4: Active MitM attack scenario

In this attack, the relayed messages 1 and 2 are injected with the adversary's public key (PK), making it possible for the adversary to decrypt the responses sent by both Dev_1 and Dev_2 and giving unrestricted access to the communication between the devices. Thus, the adversary impersonates Dev_2 as shown in Figure 4.5.

1. hej, PK_1, Dev_1, sign_{PR1}(hej, PK_1, Dev_1)

    a) hej, $PK_{Attacker}$, Dev_1, $\text{sign}_{PR_{Attacker}}$(hej, $PK_{Attacker}$, Dev_1)

2. hejhej, PK_2, Dev_2, $\text{E}_{PK_{Attacker}}$(SSK), sign_{PR2}(hejhej, PK_2, Dev_2, $\text{E}_{PK_{Attacker}}$(SSK))

    a) hejhej, $PK_{Attacker}$, Dev_2, E_{PK_1}(SSK), $\text{sign}_{PR_{Attacker}}$(hejhej, $PK_{Attacker}$, Dev_2, E_{PK_1}(SSK))

3. $\text{E}_{SSK}(snd\_packet_1)$

4. $\text{E}_{SSK}(snd\_packet_2)$

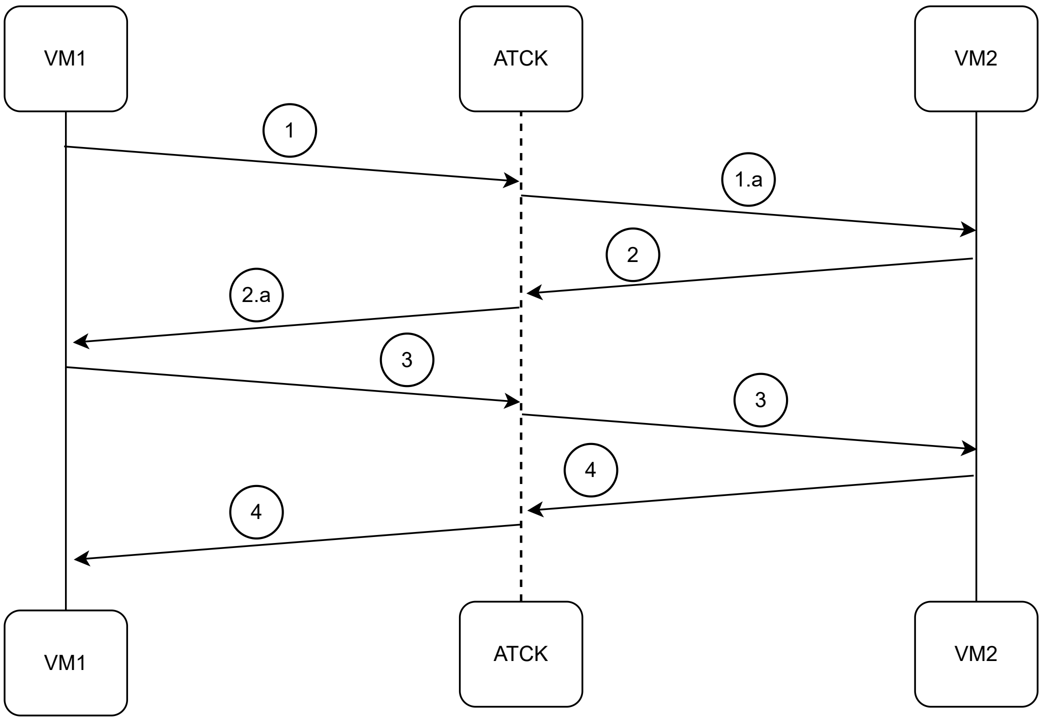


Figure 4.5: Messages relayed in an active MitM attack

4.1.3 Active MitM Attack: Authentication with ECDHE

The next iteration of the protocol uses ECDHE and *SSK*, nevertheless, this protocol is still vulnerable to an active MitM attack. In a scenario as described in Figure 4.4 and Figure 4.5, where the messages are modified and then relayed the sequence of messages would be as follows:

1. hej, PK_1, Dev_1, EPH_{PK_1}, $sign_{PR1}$(hej, PK_1, Dev_1, EPH_{PK_1})
   a) hej, $PK_{Attacker}$, Dev_1, $EPH_{PK_{Attacker}}$, sign$_{PR_{Attacker}}$(hej, $PK_{Attacker}$, Dev_1, $EPH_{PK_{Attacker}}$)
2. hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$, $sign_{PR2}$(hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$)
   a) hejhej, $PK_{Attacker}$, Dev_2, $EPH_{PK_{Attacker}}$, $HMAC_{SSK_{Attacker}}(hello)$, $sign_{PR_{Attacker}}$(hejhej, $PK_{Attacker}$, Dev_2, $EPH_{PK_{Attacker}}$, $HMAC_{SSK_{Attacker}}(hello)$)
3. Operations performed by Dev_1:
   (a) Signature verification
   (b) R = $compare(HMAC_{SSK_1}(hello), HMAC_{SSK_{Attacker}}(hello))$.
   If $R == false$, then disconnect.
4. $E_{SSK}(snd\_packet_1)$
5. $E_{SSK}(snd\_packet_2)$

In this case, the comparison between the *HMAC*s, will not cause a problem since the SSK_1 and $SSK_{Attacker}$ are the same in step 3. In other words, the process of these keys is not altered as if they are legitimate keys.

4.1.4 Active MitM Attack: Time-based Authentication with ECDHE

The time-based authentication with ECDHE protocol relies on the assumption that whatever message is delivered within the accepted time range is a valid message. Adversaries can perform a MitM attack whenever the environmental conditions and equipment allow them to relay the messages within the time range. To perform an active MitM attack, it is necessary to relay the message from the victim, get a response from the impersonated device, make the appropriate changes to the message, as shown below, and send back the message to the victim.

1. hej, PK_1, Dev_1, EPH_{PK_1}, $sign_{PR1}$(hej, PK_1, Dev_1, EPH_{PK_1})

   a) hej, $PK_{Attacker}$, $EPH_{PK_{Attacker}}$, Dev_1, sign$_{PR_{Attacker}}$(hej, $PK_{Attacker}$, Dev_1, $EPH_{PK_{Attacker}}$)

2. hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$, $sign_{PR2}$(hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$)

   a) hejhej, $PK_{Attacker}$, $EPH_{PK_{Attacker}}$, Dev_2, $HMAC_{SSK_{Attacker}}(hello)$, sign$_{PR_{Attacker}}$(hejhej, $PK_{Attacker}$, Dev_2, $EPH_{PK_{Attacker}}$, $HMAC_{SSK_{Attacker}}(hello)$)

3. Operations performed by Dev_1:

   (a) Signature verification
   (b) If $ToF_1 > Range$, then disconnect.
   (c) R = $compare(HMAC_{SSK_1}\ (hello), HMAC_{SSK_{Attacker}}(hello))$.
   If $R == false$, then disconnect.

4. $E_{SSK}(snd\_packet_1)$

5. If $ToF_1 > Range$, then disconnect.

6. $E_{SSK}(snd\_packet_2)$

4.1.5 Active MitM Attack: Location-based Authentication with ECDHE

In the protocol that implements location, it can be vulnerable when all of the devices are within the location range or when only the attacker is within the location range of both devices as shown in Figure 4.6.

The sequence of messages for the attack would be the same as in section 4.1.3. The protocol difference in this attack is the *SND* packets which contain the *longitude* and *latitude* coordinates. These coordinates would be accessed by the attacker and modified to simulate an acceptable location.

1. hej, PK_1, Dev_1, EPH_{PK_1}, $sign_{PR1}$(hej, PK_1, Dev_1, EPH_{PK_1})

   a) hej, $PK_{Attacker}$, Dev_1, sign$_{PR_{Attacker}}$(hej, $PK_{Attacker}$, Dev_1)

2. hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$, $sign_{PR2}$(hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$)

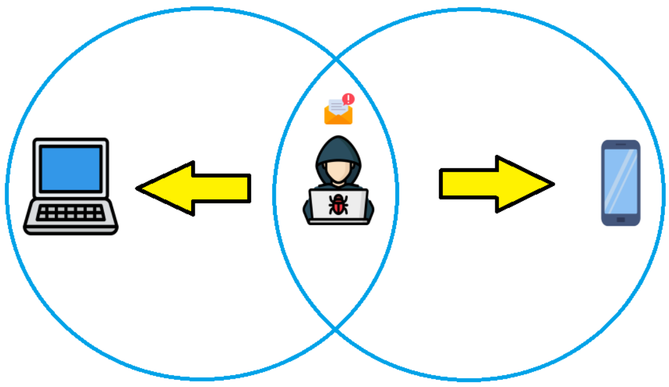


Figure 4.6: Active MitM Attack scenario for location-based

a) hejhej, $PK_{Attacker}$, Dev_2, $EPH_{PK_{Attacker}}$, $HMAC_{SSK_{Attacker}}(hello)$, $sign_{PR_{Attacker}}$(hejhej, $PK_{Attacker}$, Dev_2, $EPH_{PK_{Attacker}}$, $HMAC_{SSK_{Attacker}}(hello)$)

3. Operations performed by Dev_1:
   (a) Signature verification
   (b) R = $compare(HMAC_{SSK_1}(hello), HMAC_{SSK_{Attacker}}(hello))$. If $R == false$, then disconnect.
4. $E_{SSK}(snd\_packet_1)$
   (a) $E_{SSK}(snd\_packet_{Attacker})$
5. If $D_{12} > Range$, then disconnect
6. $E_{SSK}(snd\_packet_2)$
   (a) $E_{SSK}(snd\_packet_{Attacker})$
7. If $D_{21} > Range$, then disconnect

4.1.6 Active MitM Attack: Time-location-based Authentication with ECDHE

The attack for time-location-based protocol is a combination of the attacks for location-based and time-based protocols, where the attack is successful only when the communication time does not exceed nor is the distance range. The message-sending sequence would be the same as in section 4.1.5, and using the following messages:

1. hej, PK_1, Dev_1, EPH_{PK_1}, $sign_{PR1}$(hej, PK_1, Dev_1, EPH_{PK_1})

   a) hej, $PK_{Attacker}$, Dev_1, $EPH_{PK_{Attacker}}$, sign$_{PR_{Attacker}}$(hej, $PK_{Attacker}$, Dev_1, $EPH_{PK_{Attacker}}$)

2. hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$, $sign_{PR2}$(hejhej, PK_2, Dev_2, EPH_{PK_2}, $HMAC_{SSK_2}(hello)$)

   a) hejhej, $PK_{Attacker}$, Dev_2, $EPH_{PK_{Attacker}}$, $HMAC_{SSK_{Attacker}}(hello)$, sign$_{PR_{Attacker}}$(hejhej, $PK_{Attacker}$, Dev_2, $EPH_{PK_{Attacker}}$, $HMAC_{SSK_{Attacker}}(hello)$)

3. Operations performed by Dev_1:

   (a) Signature verification

   (b) Timer halt. If $ToF_1 > Range$, then disconnect.

   (c) $R =$ compare($HMAC_{SSK_1}(hello)$, $HMAC_{SSK_{Attacker}}(hello)$).
   If $R == false$, then disconnect.

4. $E_{SSK}(snd\_packet_1)$

   (a) $E_{SSK}(snd\_packet_{Attacker})$

5. Following checks are conducted by Dev_2:

   (a) If $ToF_2 > Range$, then disconnect.

   (b) If $D_{12} > Range$, then disconnect

6. $E_{SSK}(snd\_packet_2)$

   (a) $E_{SSK}(snd\_packet_{Attacker})$

7. If $D_{21} > Range$, then disconnect

4.1.7 Active MitM Attack: GeoTimeCert Authentication with ECDHE

A MitM attack over the GeoTimeCert Authentication with ECDHE protocol is not possible because of the implementation of nonces and certificates. The nonces used in the protocol have an expiration time, making it impossible to resend the same nonce twice. The involvement of a third-party authority makes it impossible to impersonate another device without also compromising the authentication authority.

Chapter 5

Performance Evaluation

5.1 Cryptographic Functions

In the report, the authentication process starts with the utilization of the RSA followed by the application of EdDSA which enables us to conduct a performance comparison of both cryptographic functions. By implementing RSA and EdDSA, we can assess and contrast their efficiency, speed, and overall performance in an authentication scenario. Table 5.1 outlines a cryptographic comparison of the performance between RSA and EdDSA involving several key aspects related to their security, compatibility, performance, and speed [28, 29]. This chapter examines various performance tests conducted on basic authentication protocol, authentication with ECDHE protocol, time-based authentication with ECDHE protocol, location-based authentication with ECDHE protocol, time-location-based authentication with ECDHE protocol, and GeoTimeCert authentication with ECDHE protocol. The testing is done at the 2.4 GHz band as it provides better coverage over longer distances and better penetration through walls and obstacles compared to the 5 GHz band. The error bars in the plots represent the margin of error at a 95% confidence interval.

Aspect	**RSA**	**EdDSA**
Security	RSA's security level depends on the key size and its resistance to integer factorization attacks.	EdDSA, based on Edwards curves, provides a high level of security, and depending on the curve and key length used, offers robust security with shorter key lengths.

Key Size	Typically requires longer key sizes (2048 bits or more) for security. In the evaluation, the key size is 256 bytes.	Uses shorter key sizes (around 256 bits) for equivalent security, which is more efficient. In the evaluation, the key size is 32 bytes.
Signature Size	Same as the modulus size (larger), leading to larger signatures.	Produces shorter signature sizes, which are more efficient for bandwidth and storage.
Performance: Signature Generation	Slower due to complex mathematical operations with large key sizes.	Faster signature generation, especially with smaller key sizes.
Performance: Signature Verification	Slower, as it involves exponentiation with large numbers.	Faster signature verification, with a significant speed advantage.
Implementation	RSA libraries are widely available in major languages.	EdDSA is relatively newer and might have limited library support in certain languages.
Compatibility	RSA's compatibility might be an issue with SHA-1 usage or public keys under 2048 bits.	EdDSA, particularly Ed25519, has gained significant adoption and compatibility with newer clients, with Ed448 also proposed by NIST.
Performance Considerations	Signature generation and verification can be slow with larger keys; and significant computational costs.	Faster signature operations, especially with smaller key sizes, which reduce computational overhead.

Table 5.1: Comparison between RSA and EdDSA

5.1.1 Basic Authentication Protocol

Figure 5.1 illustrates the runtime of the basic authentication protocol in AES 16, 24, and 32 which corresponds to 128, 192, and 256 bits for different RSA key lengths of 1024, 2048, and 3072 bits. It is evident from the Figure 5.1 that the total time increases as the RSA key size is increased.

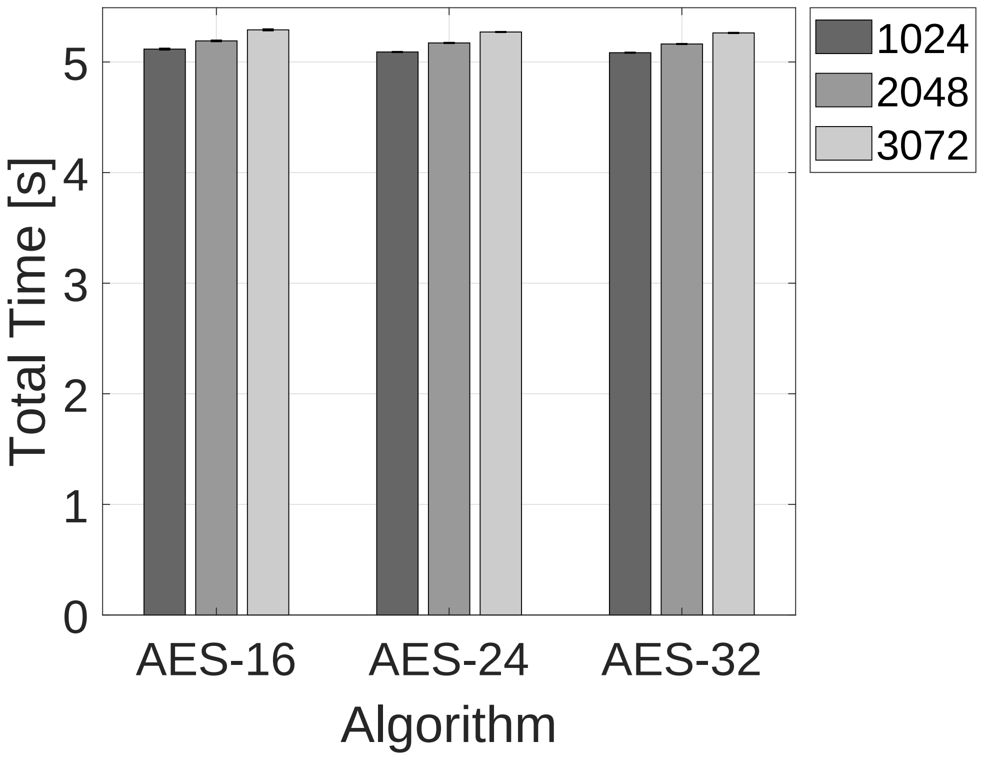
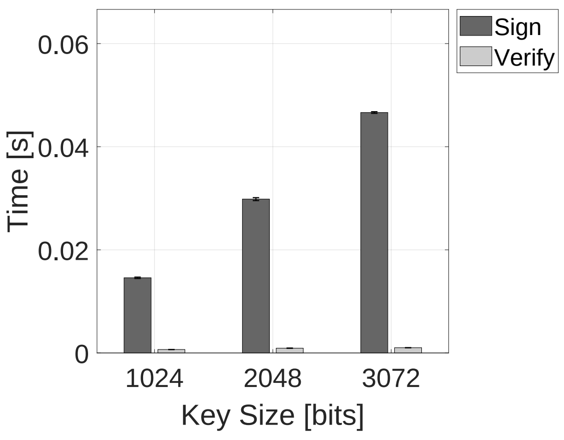
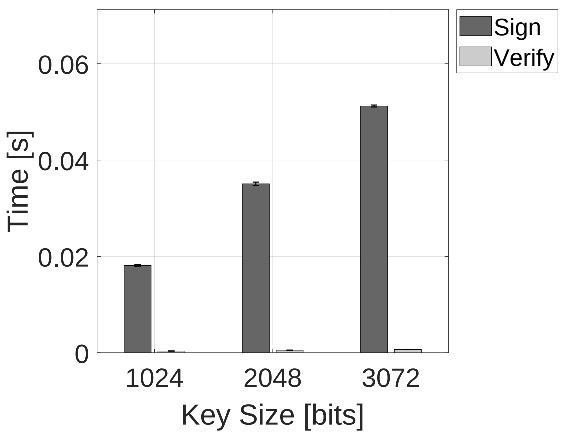


Figure 5.1: Comparison of total time in the basic authentication protocol in AES 16, 24, and 32 for different RSA key lengths of 1024, 2048, and 3072 bits

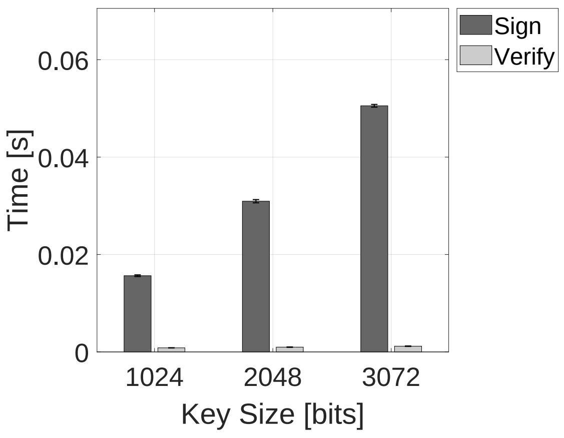
Figure 5.2 compares the signing and verifying time of a message at the receiver and transmitter sites. Additionally, the execution times for signing a message using RSA and usage of AES with different key lengths are analyzed. The results show that increasing RSA key length generally results in longer processing times for both signing and verification. The impact of AES key length on verification time is observed to be minimal. These findings provide insights into the performance trade-offs associated with different RSA and AES key lengths during message signing and verification processes on both the transmitter and receiver sides.



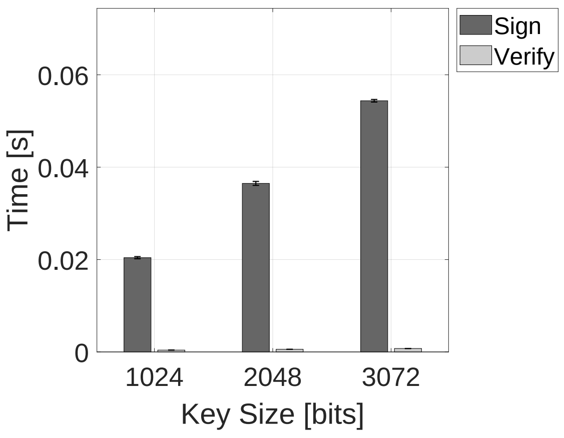
(a)



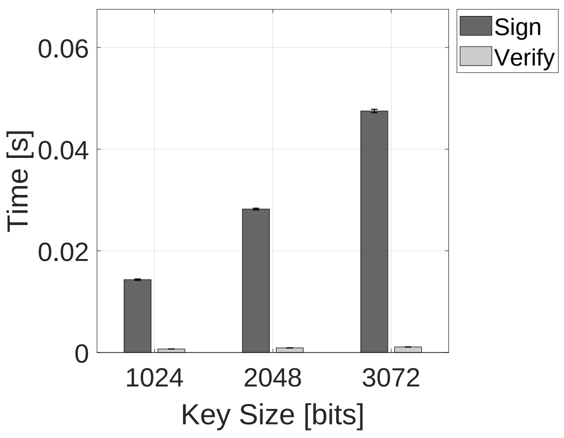
(b)



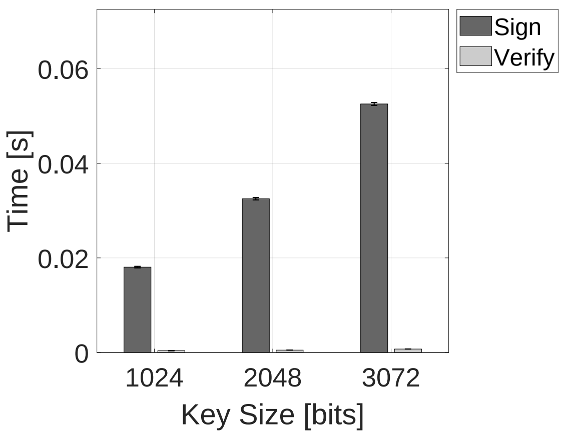
(c)



(d)

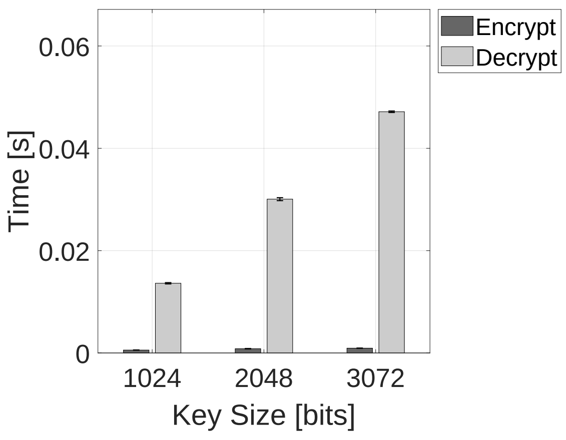


(e)

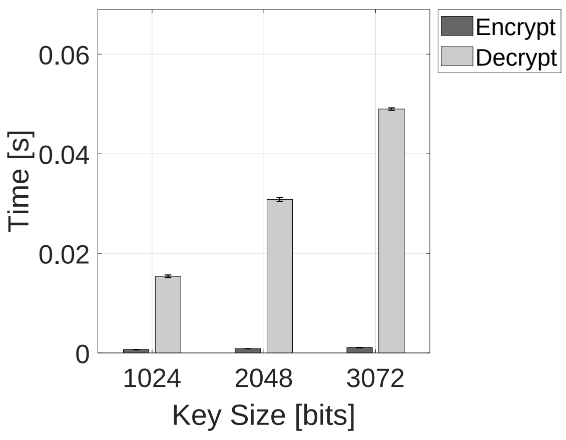


(f)

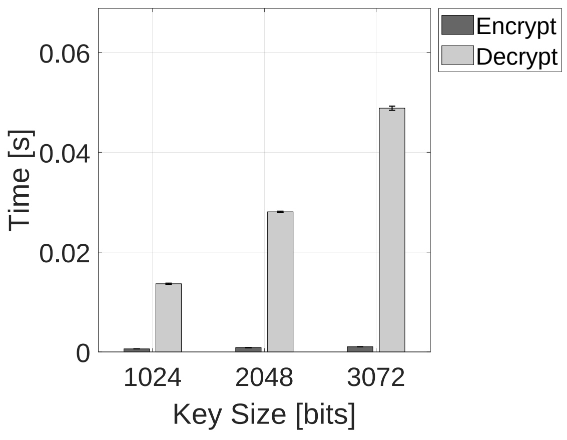
Figure 5.2: Comparison of signing and verifying time at the receiver (a,c,e) and transmitter (b,d,f) site in AES 16, 24, and 32, respectively, for RSA key lengths of 1024, 2048, and 3072 bits



(a)



(b)



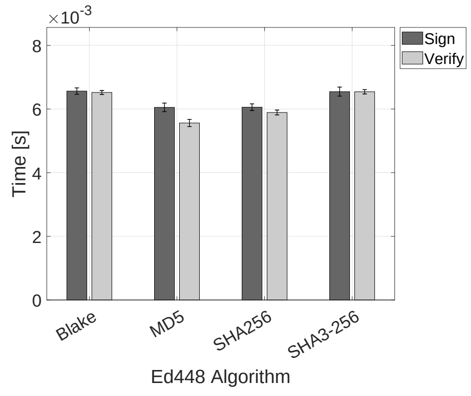
(c)

Figure 5.3: Comparison of public key encryption and decryption time for AES 16 (a), 24 (b), and 32 (c) for RSA key lengths of 1024, 2048, and 3072 bits

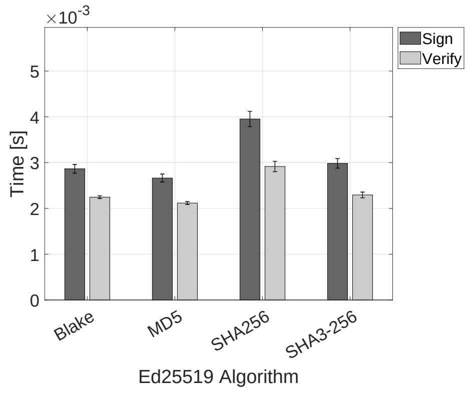
Generally, increasing RSA key size leads to longer encryption and decryption times as shown in Figure 5.3. For each AES key size, decryption times are consistently higher than encryption times. The impact of RSA key size on decryption time is more pronounced than on encryption time. These observations provide insights into the performance characteristics of AES encryption and decryption with varying RSA key sizes, emphasizing the trade-offs associated with different key lengths in terms of processing times.

5.1.2 Authentication with ECDHE Protocol

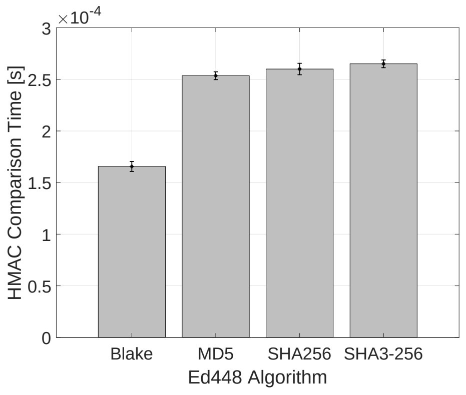
Figure 5.4 compares the signing and verifying time, HMAC computation time, AES-GCM encryption time, and total time for Ed448 and Ed25519 elliptic curves based on different digest modes such as Blake, MD5, SHA256 and SHA3-256. In this protocol, we use Ed25519 with SHA256 digest mode. Although, by running the performance evaluation tests between Ed25519 and Ed448 with different digest modes, we observe that there exist better and less time-consuming alternatives, such as Ed25519 with Blake, Ed448 with Blake, etc.



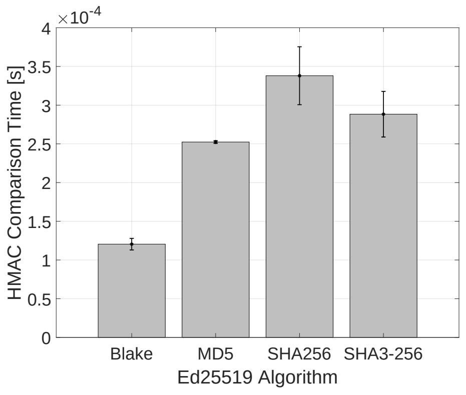
(a) Signing and verifying in Ed448



(b) Signing and verifying in Ed25519



(c) HMAC computation in Ed448



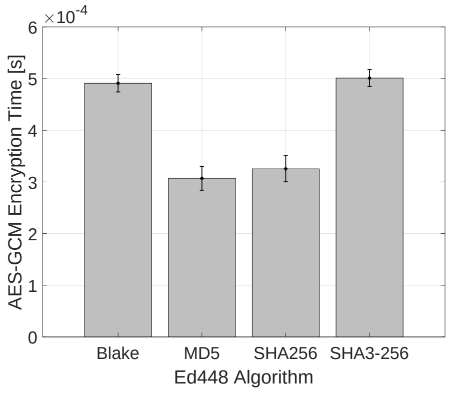
(d) HMAC computation in Ed25519

Figure 5.4: Comparison of Times: Signing and verifying time, HMAC computation time, and encryption time, and total time in Ed448 and Ed25519 curves with different digest modes such as Blake, MD5, SHA256, and SHA3-256

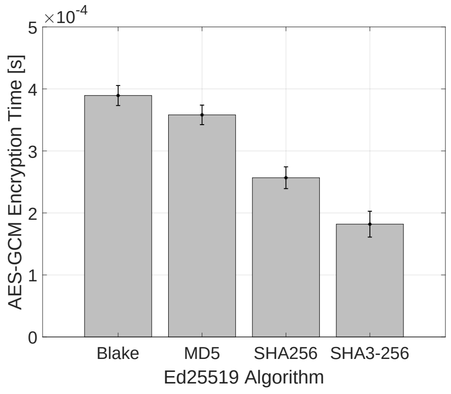
However, in this project, we opt to go with Ed25519 over Ed448 because of the following reasons:

- Ed25519 has a key size of 256-bit, which is secure enough against most of the attacks and is used widely. Meanwhile, Ed448 can also provide a great security advantage due to its larger key size i.e., 456-bit, but it also impacts the performance due to its larger key size in resource-constrained environment [30].

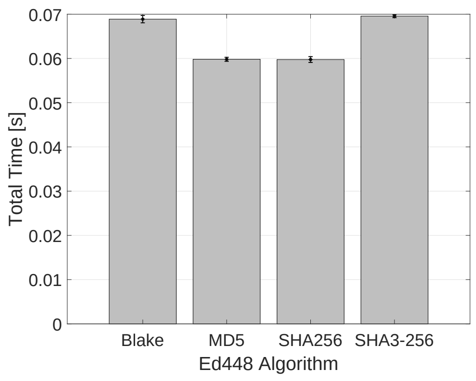
- In contrast to Ed448, Ed25519 is widely supported by various cryptographic libraries and protocols, such as TLS and SSH.



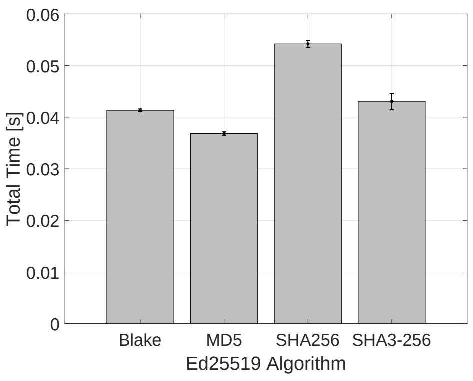
(e) Encryption in Ed448



(f) Encryption in Ed25519



(g) Total time in Ed448



(h) Total time in Ed25519

Figure 5.4: Comparison of Times: Signing and verifying time, HMAC computation time, and encryption time, and total time in Ed448 and Ed25519 curves with different digest modes such as Blake, MD5, SHA256, and SHA3-256 (continued)

We use SHA-256 over Blake because:

- SHA-256 ensures better compatibility and support as it is widely adopted and standardized in various cryptographic protocols.

- SHA-256 is a highly used hash function that has gone through extensive cryptanalysis. It belongs to the SHA-2 family which has proven to be secure for many applications.

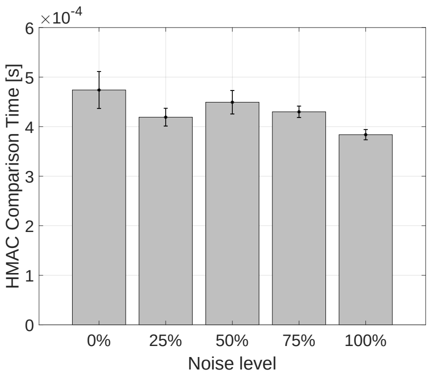
However, both Ed448 and ED5519 with other digest modes also exhibit better features in certain domains than the one that we choose in this project, and it is evident by these performance evaluation runs. Overall, it can be observed that the signing and verification times in Ed448 with different digest modes are similar, while in Ed5519, SHA-256 has proven to be taking the most time among all others. The total time of SHA-256 is also greater than most of the options as can be seen in Figure 5.4h.

The combination of Ed25519 for digital signatures and SHA256 as the digest mode is chosen to optimize the performance of the protocol without compromising on security. This selection ensures that the protocol benefits from efficient cryptographic operations while maintaining a strong level of protection against potential vulnerabilities.

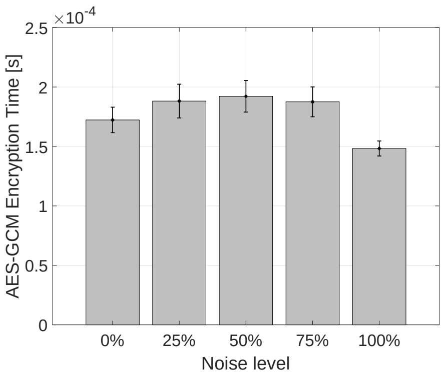
5.1.3 Time-based Authentication with ECDHE Protocol

Figure 5.5 illustrates the comparison of HMAC computation time, AES-GCM encryption time, signing and verifying time, and total time in the presence of random channel noise and varying channel reliability levels. Here, noise level means any kind of interruption in the network that may delay the packet transmission between the devices, hence affecting the channel reliability. This type of evaluation is important to assess the robustness and reliability of your authentication system under different conditions. In this evaluation, we simulate random channel noise at different levels (0%, 25%, 50%, 75%, 100%) to mimic real-world conditions.

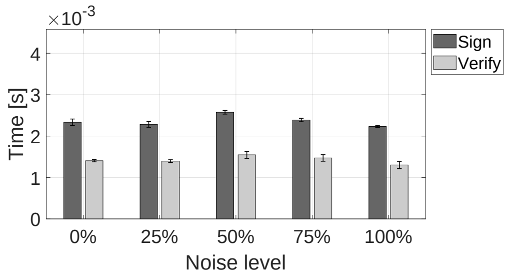
The following graphs of the performance evaluation show that the HMAC computation, encryption, and signing and verification times at different noise levels are almost similar. However, the total time depicted in Figure 5.5d clearly shows that the time increases with an increase in the noise level. This means that whenever we increase the noise in a channel, the reliability of that channel ultimately declines and hence the transmission time gets a significant increase.



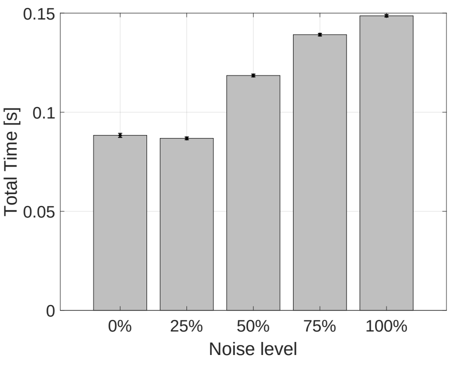
(a) HMAC computation



(b) AES-GCM encryption



(c) Signing and verifying



(d) Total time

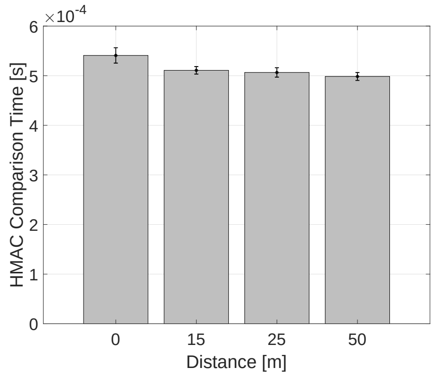
Figure 5.5: Comparison of Times: HMAC computation time, and AES-GCM encryption time, signing and verifying time, and total time at 0%, 25%, 50%, 75% and 100% noise level and 100%, 75%, 50%, 25%, and 0% channel reliability level in Time-based authentication with ECDHE

This evaluation helps us measure how often the authentication succeeds and how quickly it responds in situations where the communication isn't perfect. By doing this, we can figure out if the system is robust (can handle challenges) and reliable (works well under different conditions). The goal is to ensure that the system not only keeps things secure but also performs effectively in the real world.

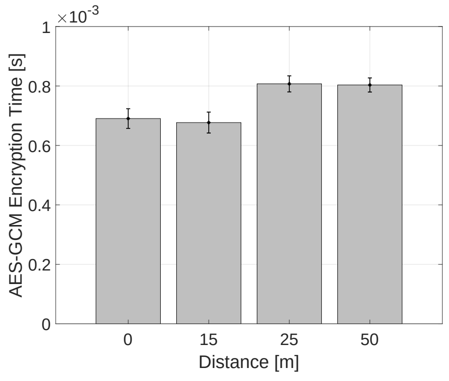
5.1.4 Location-based Authentication with ECDHE Protocol

The performance evaluation of location-based ECDHE at different distances, specifically 0, 15, 25, and 50 meters, is crucial to assess the system's efficacy in varying physical proximity scenarios. In this evaluation, the goal is to measure and analyze the key aspects of ECDHE-based cryptographic operations as the distance between communicating entities changes. Parameters such as total time, sign and verify time, HMAC computation time, and GCM encryption time are examined to understand the impact of distance on the system's performance. This comparison in Figure 5.6 helped us to understand how the devices behave under different distances, which is crucial for scenarios where secure communication or authentication is required across varying physical distances.

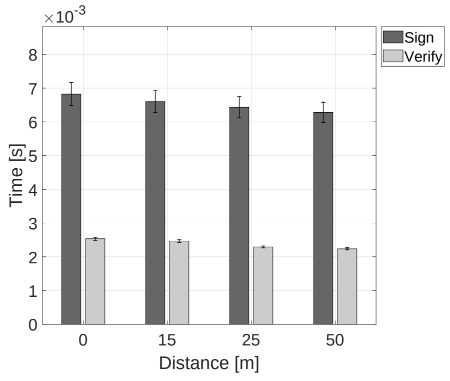
Similar to time-based protocol, the performance evaluation graphs of location-based variants of SND also show that the HMAC computation, encryption, and signing and verification times at different physical distances are almost similar. However, the total time depicted in Figure 5.6d clearly shows that the transmission time significantly increases each time the physical distance increases. This means that whenever we increase physical distance between the two communicating nodes it will take longer for an SND packet to travel between the devices.



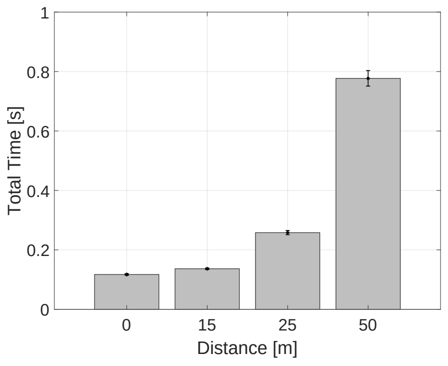
(a) HMAC computation



(b) AES-GCM encryption



(c) Signing and verifying

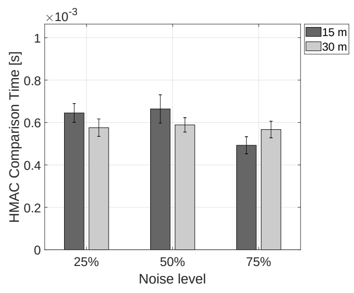


(d) Total time

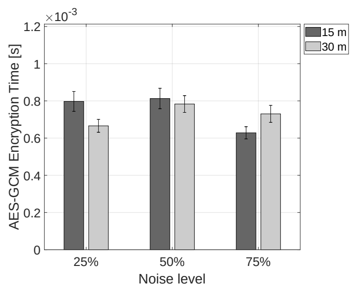
Figure 5.6: Comparison of Times: HMAC computation time, and AES-GCM encryption time, signing and verifying time, and total time at 0, 15, 25 and 50 m distance in Location-based authentication with ECDHE

5.1.5 Time-location-based Authentication with ECDHE Protocol

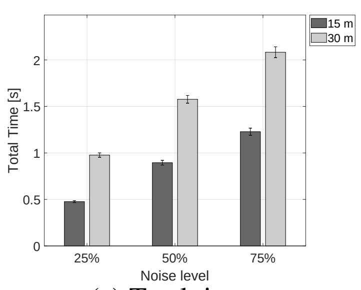
Figure 5.7 depicts the performance evaluation of the third variant of the SND protocol which is a time-location-based protocol. In addition to time-based and location-based protocols, we combined both the time and location aspects in this protocol's implementation.



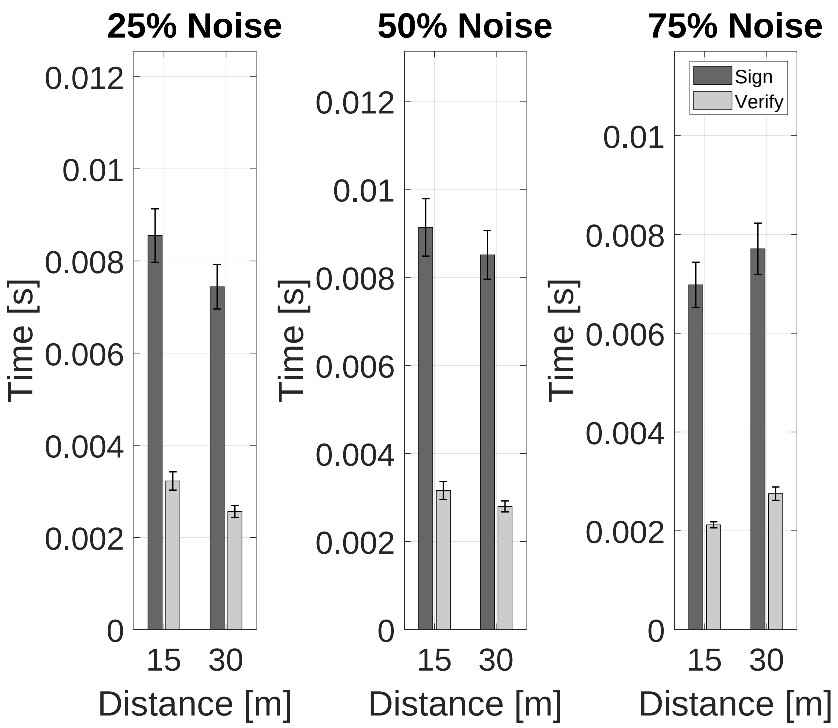
(a) HMAC computation



(b) AES-GCM encryption



(c) Total time



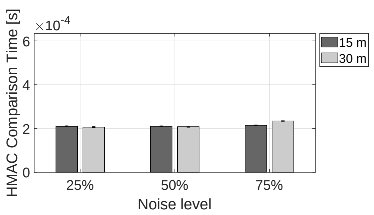
(d) Sign and Verify

Figure 5.7: Comparison of Times: HMAC Computation Time, and AES-GCM Encryption Time, Sign and Verify Time, and Total Time at 15 and 30 m distance at 25%, 50% and 75% noise level in time-location-based Authentication with ECDHE

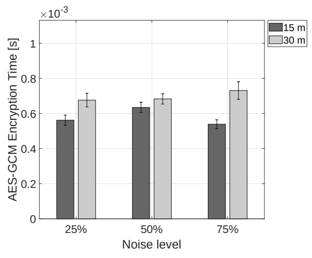
To evaluate the performance of this protocol we tested it on multiple levels of noise i.e., 25%, 50%, and 75%, as well as at different physical distance ranges i.e., 15 meters and 30 meters. We evaluated different aspects like HMAC computation time, AES-GCM encryption time, signing and verification time, and total time. Overall, it can be observed from the plots that the HMAC computation, encryption, signing, and verifying time do not exhibit significant differences at different noise and distance levels. However, the total time in Figure 5.7c depicted a consistent increase with the increase in noise level and physical devices.

5.1.6 GeoTimeCert Authentication with ECDHE Protocol

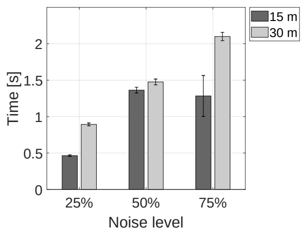
The GeoTimeCert authentication protocol extends the capabilities of the time-location-based protocol by introducing nonces and a certificate authority (CA). Building upon the fundamental principles of time and location attributes in authentication, this protocol enhances security through the generation of unique nonces for each session, thereby fortifying defenses against replay attacks. The integration of a Certificate Authority establishes a trusted third party that issues digital certificates, validating the legitimacy of communicating devices. In evaluating performance, the protocol undergoes tests based on metrics similar to the time-location-based protocol. The protocol's resilience is assessed under adverse conditions, including heightened levels of noise and the proliferation of physical devices in the communication environment. Notably, an increase in noise levels and the presence of additional devices correlates with an increase in total authentication time, emphasizing the need for adaptive measures to ensure the protocol's effectiveness in diverse real-world scenarios. In conc;usion, the total time increase with the increase in noise level and physical devices.



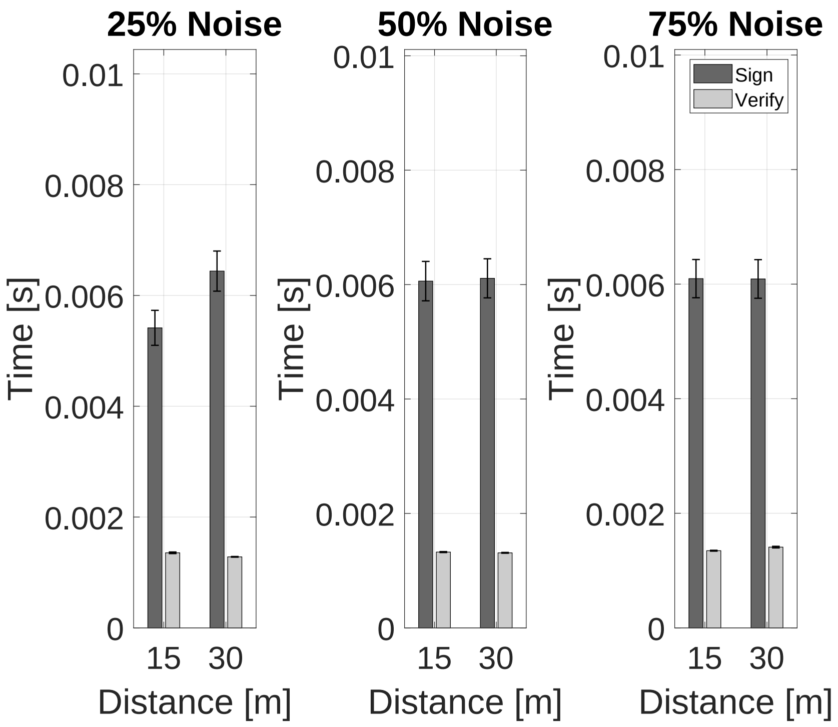
(a) HMAC Computation



(b) AES-GCM Encryption



(c) Total Time



(d) Sign and Verify

Figure 5.8: Comparison of Times: HMAC Computation Time, and AES-GCM Encryption Time, Sign and Verify Time, and Total Time at 15 and 30 m distance at 25%, 50% and 75% noise level in GeoTimeCert Authentication with ECDHE

Chapter 6

Conclusions

In this concluding section, we summarize the project work and clarify the key takeaways. Section 6.1 offers the conclusion remark, followed by Section 6.2 which presents the limitations of the project. Section 6.3 highlights potential future work for further exploration. Section 6.4 brings the paper to a close with a reflective analysis.

6.1 Conclusion

In this project, the team has explored methods to make neighbor discovery protocols more secure. We implemented various methods, mainly time-based, location-based, time-location-based, and GeoTimeCert authentication protocols, and compared the results obtained. Additionally, we have implemented attacker scenarios against these protocols and measured their performance and efficiency. We demonstrated that the GeoTimeCert authentication approach is the most efficient in mitigating active attacks.

To investigate wireless communication, the research study uses Alfa devices (AWUS036ACHM). There are three devices in use: one for active attacks, two for transmission and reception. Because of the device range specifications, Alfa devices are utilized close to one another. The devices are then linked to virtual machines. Due to the restricted number of devices, the communication is done as direct communication.

6.2 Limitations

The implementation of the project has been carried out in a virtual environment utilizing two Alfa devices. Consequently, the results and the assessments presented in the paper might deviate from real-world scenarios. We characterize the direct communication between the two devices as peer-to-peer. However, we recognize that the cluster of devices in the real world tends to be rather complex.

The implementation mainly considers protection mechanisms against relay attacks, which create false communication between non-neighboring devices. It is reasonable to expect other kinds of active attacks even against legit communication between neighboring devices.

In real-world scenarios, it would be more practical to employ beacon broadcasting for secure network communication instead of relying on direct connections. Due to the constraints of time limitations, we use a simplified approach by sharing IP addresses among devices instead of beacons, sending broadcast discovery messages.

Using the *ToF* method, we add a constant delay for the time of flight to account for some uncertainty and provide more realistic results.

We employ Euclidian distance, which measures the short and straight line distance between two nodes, to calculate the distance between the two devices to determine whether or not they are in range. Nonetheless, we are aware that Haversine distance, which takes angular distances and obstacles into account, offers a more accurate distance calculation.

We implement a simplified certification generation, capable of handling fundamental tasks, including certificate issuance and verification. We require a more robust CA with additional features, including certificate revocation and certificate rotation. To simplify things and facilitate learning, the certificates are self-signed and only have one layer rather than utilizing certificate chains.

Additionally, in our last *GeoTimeCert Authentication with ECDHE* protocol, the implementation of a separate thread for cleaning up expired nonces has been omitted, citing considerations related to concurrent handling and time constraints.

Overall, the application is not optimized for deployment at the production level; rather, it is meant to serve as a proof of concept.

6.3 Future work

In this project, we manage to design and implement the fundamental and authenticated ND, time-based, location-based, time-location-based, and GeoTimeCert authentication protocols, along with active MitM attacks.

Based on the results obtained, we demonstrate that the GeoTimeCert authentication approach is the most effective one in mitigating relay attacks. However, further research has to be done on the investigation of vulnerabilities of the protocol. It is left to future research to examine different types of attacks against the protocol and defense strategies while considering the performance.

Since Alfa devices serve as the testing ground for the protocols, they exhibit specific resource limitations. One of the major constraints is the limited capacity of the devices to form connections, which makes them vulnerable to Denial of Service (DDoS) attacks. To prevent this attack, It is recommended to implement a maximum connection setting. Subsequent research could further explore strategies to reinforce the devices against DDoS attacks.

While verifying the validity of the nonces, both communicating devices check the expiration time. If the nonces didn't expire, it will then be checked against a database of nonces to determine its freshness. If it is fresh it will be stored in the database and the device proceeds with the communication. However, the size of the database could rapidly grow if expired nonces are not removed from it. To employ this, we recommend future studies use a separate thread to clean up the local database to optimize performance.

Once a trusted communication is established between two devices, future communications should be simpler. For the reconnection, initial handshakes are not necessary. This might significantly reduce processing overload and improve the performance of the protocol. Nevertheless, these strategies are not addressed in the study, it is reserved for future explorations.

6.4 Reflections

Establishing connections between wireless devices heavily relies on neighbor discovery. Using secure neighbor discovery techniques is critical in preventing miscommunication. Encryption and authentication techniques are essential for guaranteeing confidentiality and authenticity. Utilizing additional strategies, such as time- and location-based approaches, is also beneficial. Time-based protocols measure the difference between the transmission and delivery times to determine if the other device is within range.

On the other hand, location-based protocols use a distance measurement between the two nodes to compare it to an accepted range to assess whether or not the other node is a legitimate neighbor. Although these two methods are a fine place to start, they are not sufficiently protected from man-in-the-middle attacks. The attacker could deceive the nodes by manipulating the output using a variety of techniques.

The combined time-location-based protocol provides enhanced security for neighbor discovery protocols. Even though it is more secure than the previous protocols, it is still vulnerable to MitM attacks.

Conversely, the GeoTimeCert authentication protocol provides highly robust security. Certificates and nonces are utilized to achieve this. The certificate authenticates the participating entities, reducing vulnerabilities in the protocol. We implemented our own CA with the basic functionalities of issuing and verifying the certificates. Nonces, on the other hand, ensure the freshness of the messages, which is an efficient method to provide forward secrecy and also prevent reply attacks. However, it should be recognized that there is no absolute or universal security. Better approaches might be explored in the future to achieve a more solid secure neighbor discovery.

Bibliography

[1] R. Hassan, A. S. Ahmed, and N. E. Osman, "Enhancing security for ipv6 neighbor discovery protocol using cryptography," *American Journal of Applied Sciences*, vol. 11, pp. 1472–1479, Jul. 2014. doi: 10.3844/ajassp.2014.1472.1479. [Online]. Available: https://thescipub.com/abstract/ajassp.2014.1472.1479

[2] J. Ioannidis and G. Q. M. Jr., "Coherent file distribution protocol," *Internet Request for Comments*, vol. RFC 1235 (Experimental), Jun. 1991. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1235.txt

[3] J. Postel, "Internet protocol," *Internet Request for Comments*, vol. RFC 791 (Standard), Sep. 1981, updated by RFC 1349. [Online]. Available: http://www.rfc-editor.org/rfc/rfc791.txt

[4] S. Deering and R. Hinden, "Internet protocol, version 6 (IPv6) specification," *Internet Request for Comments*, vol. RFC 2460 (Draft Standard), Dec. 1998, updated by RFCs 5095, 5722, 5871. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2460.txt

[5] P. Papadimitratos, M. Poturalski, P. Schaller, P. Lafourcade, D. Basin, S. Capkun, and J.-P. Hubaux, "Secure neighborhood discovery: a fundamental element for mobile ad hoc networking," *IEEE Communications Magazine*, vol. 46, no. 2, pp. 132–139, 2008. doi: 10.1109/MCOM.2008.4473095

[6] M. Poturalski, P. Papadimitratos, and J.-P. Hubaux, "Formal analysis of secure neighbor discovery in wireless networks," *IEEE transactions on dependable and secure computing*, vol. 10, no. 6, pp. 355–367, 2013.

[7] R. Shokri, M. Poturalski, G. Ravot, P. Papadimitratos, and J.-P. Hubaux, "A practical secure neighbor verification protocol for wireless sensor networks," in *Proceedings of the second ACM conference on Wireless network security*, 2009, pp. 193–200.

[8] Z. Jian-zhao, Z. Hang-sheng, and Y. Fu-qiang, “A fast neighbor discovery algorithm for cognitive radio ad hoc networks,” in *2010 IEEE 12th International Conference on Communication Technology*, 2010. doi: 10.1109/ICCT.2010.5688847 pp. 446–449.

[9] W. Sun, Z. Yang, X. Zhang, and Y. Liu, “Energy-efficient neighbor discovery in mobile ad hoc and wireless sensor networks: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1448–1459, 2014. doi: 10.1109/SURV.2013.012414.00164

[10] A. A. Khan, M. H. Rehmani, and Y. Saleem, “Neighbor discovery in traditional wireless networks and cognitive radio networks: Basics, taxonomy, challenges and future research directions,” *Journal of Network and Computer Applications*, vol. 52, pp. 173–190, 2015. doi: https://doi.org/10.1016/j.jnca.2015.03.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804515000569

[11] M. Bagnulo and J. Arkko, “Cryptographically generated addresses (cga) extension field format,” *IEEE Communications Magazine*, 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4581

[12] M. Cobb, “What is the rsa algorithm? definition from searchsecurity,” Nov. 2021. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/RSA

[13] “Pkcs#1 v1.5 (rsa) - pycryptodome 3.19.0 documentation.” [Online]. Available: https://www.pycryptodome.org/src/signature/pkcs1_v1_5

[14] “Pkcs#1 oaep (rsa) - pycryptodome 3.19.0 documentation.” [Online]. Available: https://www.pycryptodome.org/src/cipher/oaep

[15] I. Kariyawasam, “Selecting the best aes block cipher mode (aes-gcm vs aes-cbc),” May 2021. [Online]. Available: https://isuruka.medium.com/selecting-the-best-aes-block-cipher-mode-aes-gcm-vs-aes-cbc-ee3ebae173c

[16] “Cryptosys pki pro manual,” Jan. 2023. [Online]. Available: https://www.cryptosys.net/pki/manpki/pki_aesgcmauthencryption.html

[17] J. Lake, “What is the diffie–hellman key exchange and how does it work?” Aug. 2023. [Online]. Available: https://www.comparitech.com/blog/information-security/diffie-hellman-key-exchange/

[18] Apr. 2023. [Online]. Available: https://www.geeksforgeeks.org/implementation-diffie-hellman-algorithm/

[19] "Eddsa and ed25519 - practical cryptography for developers." [Online]. Available: https://cryptobook.nakov.com/digital-signatures/eddsa-and-ed25519

[20] "Hmac (hash-based message authentication codes) definition." [Online]. Available: https://www.okta.com/identity-101/hmac/#:~:text=Hash%2Dbased%20message%20authentication%20code,function%20and%20a%20secret%20key.

[21] R. Awati and P. Loshin, "What is a certificate authority (ca)?" Sep 2021. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/certificate-authority#:~:text=A%20certificate%20authority%20(CA)%20is,trust%20in%20content%20delivered%20online.

[22] Admin, "Euclidean distance - definition, formula, derivation & examples," Dec 2021. [Online]. Available: https://byjus.com/maths/euclidean-distance/

[23] "Haversine formula to find distance between two points on a sphere," Sep 2022. [Online]. Available: https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/

[24] G. Singh, "Cqi (channel-quality indicator)," Mar 2023. [Online]. Available: https://www.telecomtrainer.com/cqi-channel-quality-indicator/

[25] "Modern modes of operation for symmetric block ciphers - pycryptodome 3.19.1 documentation." [Online]. Available: https://pycryptodome.readthedocs.io/en/latest/src/cipher/modern.html

[26] S. Shea, "What is csr (certificate signing request)?: Definition from techtarget," Mar 2023. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/CSR-Certificate-Signing-Request

[27] Z. E. Mrabet, N. Kaabouch, H. E. Ghazi, and H. E. Ghazi, "Cyber-security in smart grid: Survey and challenges," *Computers & Electrical Engineering*, vol. 67, pp. 469–482, 2018. doi: https://doi.org/10.1016/j.compeleceng.2018.01.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790617313423

[28] E. Kontsevoy, "Comparing ssh keys - rsa, dsa, ecdsa, or eddsa?" Apr. 2022. [Online]. Available: https://goteleport.com/blog/comparing-ssh-keys/

[29] A. Kardi, R. Zagrouba, and M. Alq, "Performance evaluation of rsa and elliptic curve cryptography in wireless sensor networks," in *2018 21st Saudi Computer Society National Computer Conference (NCC)*, Apr. 2018. doi: 10.1109/NCG.2018.8592963 pp. 1–6.

[30] S. Josefsson and I. Liusvaara, “Rfc 8032: Edwards-curve digital signature algorithm (eddsa).” [Online]. Available: https://datatracker.ietf.org/doc/rfc8032/