A Framework for Secured Computation over the Untrusted Cloud

Abdullah-Al-Tariq
144608

A Dissertation
Presented to the Faculty
of Islamic University of Technology
in Partial Fulfillment of the Degree
of Master of Science
in Computer Science and Engineering

Adviser
Dr. Abu Raihan Mostofa Kamal
Associate Professor
Department of Computer Science and Engineering
Islamic University of Technology

June 2018

Declaration

I hereby declare that this dissertation entitled **A Framework for Secured Computation over the Untrusted Cloud** was carried out by me for the degree of **Master of Science in Computer Science and Engineering**, M.Sc. (Engg.) in CSE, under the guidance and supervision of Dr. Abu Raihan Mostofa Kamal, Islamic University of Techology, Gazipur.

The findings put forth in this work are based on my research and understanding of the original works and they are not published anywhere in the form of books, monographs or articles. The other books, articles and websites, which I have made use of are acknowledged at the respective place in this thesis.

For the present thesis, which I am submitting to the University, no degree or diploma or distinction has been conferred on me before, either in this or in any other University.

Author

Abdullah-Al-Tariq
Student ID: 144608

Date: ____________________

Abstract

In an era where virtually every electronic device is built with the capability to connect to the internet, and thus to the cloud, end users now are getting more adjoined with numerous IoT devices which are part of a myriad of Smart Systems. Moreover, these heterogeneous IoT environments, with newly boosted security features - thanks to exponential growth in security solutions for IoT devices - possess unique behavioral patterns in unique environments. Additionally, it is highly unlikely that in such an environment, with plethora of device-types,every device will leak information at once. Thus, in this work, we propose a security framework to establish secured communication between end-user and the cloud using the behavioral patterns of the IoT devices which are accessed by both the communicating parties following some proper authorization. To implement our proposal, we have used Sensorscope sensor network's weather sensor data. After training an *Long Short Term Memory* network model using time series of sensor data, we predicted session keys between the cloud and the user using noisy data. The goals we attained from this work are Twofold. First, we achieved forward secrecy using session keys which are generated using noisy environment data. Second, it is observed that when we decrypted the messages using noisy-data-generated session keys, the accuracy of decryption varied according to the proportion of the added noise in the sensor data. For keys generated with normalized noise with 3% standard deviation, we found out decryption accuracy to be as high as 96%. On the other hand, communicating parties from two different environments can only decrypt only 50% of the message bits accurately. Finally we argued, since the noise in sensor data is reflected in the decryption accuracy, successful decryption of messages with narrow margin of error verifies that the communicating parties are part of the same environment and thus any intruder with information of the partial environment cannot communicate without decryption accuracy falling drastically.

Acknowledgements

I would first like to thank my thesis advisor Dr. Abu Raihan Mostofa Kamal, Associate Professor of the the department of Computer Science and Engineering at Islamic University of Technology. The door to Dr. Kamal's office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this work to be my own, but steered me in the right the direction whenever he thought I needed it. I would also like to extend my cordial gratitude towards Mr. Md. Sakhawat Hossen, Assistant Professor of CSE department. Mr. Hossen's theoretical prowess regarding cryptography, which I was introduced while taking part in CSE 6251 of my M.Sc., greatly assisted my endeavor.

I would also like to thank the expert committee members who were involved in the validation this research work. Without their passionate participation and input, criticism and productive feedback, the thesis could not have been successfully conducted. Moreover, I would like to pass my ardent gratitude and appreciation to all the faculty and staff members of department of CSE, IUT for their kind support.

Finally, I would also like to acknowledge Prof. Dr. Mahbub Alam, Head of the department, Computer Science and Engineering, IUT, who despite all the workloads of the department gave me breathing space to finish on my research work and this dissertation. Without his tremendous support, this thesis would never have finished.

To Carl Sagan

"The nitrogen in our DNA, the calcium in our teeth, the iron in our blood, the carbon in our apple pies were made in the interiors of collapsing stars. We are made of star stuff"

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Internet of Things (IoT) spectacularly emerged in the last decade, riding the all-engulfing phenomenon of connecting every single device to the internet. Though a formal definition of *IoT* is yet to be conceived, devices now are ever increasingly sharing information with each other and with the internet. Consequently, new so-called Smart Systems are coming to life. Cyber-Physical Systems in smart vehicles, smart buildings, health monitoring, energy managements, construction management, environmental monitoring, production and assembly line management, food supply chain management are just some names [37] in an ocean of such systems. As IoT devices are getting closer to our everyday life through various smart systems, ensuring security for them are becoming increasingly challenging. If we look at the CISCO Proposed seven layered architecture of IoT [13], we can broadly segregate them in three major layers [37] - Edge-side Layer, Server/Cloud-side Layer and User-side Layer. Most state-of-the-art security countermeasures for the vulnerabilities of the IoT architectures constrict their scope within the Edge-side Layer of the aforementioned architecture, while ignoring the other two layers as if they had no business with the Edge-nodes. There is no denying the fact that it is absolutely imperative to ensure the security of devices at the edges of any IoT architecture. However, our proposed work here ensures secured communication between Server/Cloud-side and User-side Layers utilizing the heterogeneity of the devices in Edge-side Layer.

An example of such secured communication is depicted in Figure 1.1. In our example, sensors of different types in a smart car may connect to a remote cloud to receive instructions. However, the instructions from the remote cloud must be passed to the sensors through an on-board user-application. If all the brake-pedal pressure sensors were compromised on a fateful day, it is highly unlikely that others would walk the same path too. Thus an attacker may receive data from the car's specific sensors but not from the whole environment. Whereas, both the server and the on-board user application can access the whole sensor array. Thus, for that particular vehicle, both the app and the server coming to agree on the same session key to encrypt communication, through the inputs from the heterogeneous sensors of the car, implicitly results in *Forward Secrecy* and *Group Membership Verification.* Our goal in this work is to use the heterogeneity of the IoT devices present in different environments to come up with a security solution that rely neither on a particular

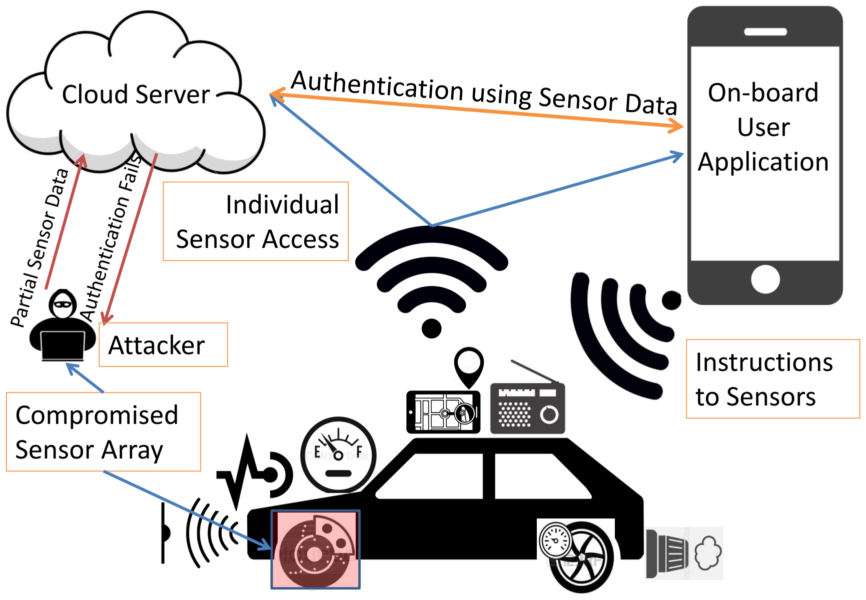


Figure 1.1: Example of Secured Communication in a Smart Car

device which can be exploited for its specific vulnerabilities, nor on the user credentials which can be set as default or lost.

In this work, we propose a LWE based public key encryption scheme that achieves *Forward-Secrecy* by generating temporary session keys between the cloud and user application. Our protocol achieves a common session key by predicting the outputs of heterogeneous IoT devices in a particular environment using *Long Short Term Memory* network [50] generated time-series [22]. We tried to ensure that the behavior of communicating parties in a certain environment is considered as an implicit part of our forward secrecy scheme, so that the integrity of the messages remains intact, should the long-term private key is compromised. Moreover, we ensured, to some extend, anonymous verification of group-membership for the communicating parties. This is achieved by assuming that observations of the same environment variables, spanning over an extended period of time, can only be accessible to both parties of the same environment. Consequently, such observations are hardly impeccable to reconstruct without substantial error for anyone outside of that particular environment. As the communicating parties need to observe and/or predict all the interacting variables in a particular environment, we propose a protocol to construct a common session key to send encrypted messages and successfully decrypt them. Moreover, communicating parties attaining such session key proves that both the parties are part of the same environment with access to the aforementioned variables. This, as a consequence, verifies their *Group Membership* anonymously.

We demonstrated our scheme by collecting data from SensorScope sensor networks deployed in Le Genepi[52] and Grand St. Bernerd[51] for over a period of more than a month. We then generated session keys from predicting the time series of noise-introduced sensor data using LSTM-network model. Later we used these session keys to encrypt messages following *Learning With Error* based public key encryption method. Our best results show that after introducing 7.03% *Normalized Root-Mean-Squared Error (NRMSE)* to the test sensor data, average decryption accuracy can get as high as 96%.

1.1 Problem Statement

On October 21, 2016, many major internet sites like Twitter, Amazon, Tumblr, Reddit, Spotify and Netflix faced unprecedented congestion due to an attack on Krebs on Security, OVH and Dyn, restraining them from providimg critical internet infrastructure services to the mentioned destinations [4]. Though this is not particularly a new phenomenon in the recent years, what unique about this particular attack by *Mirai Botnet* was the devices used - devices that were pervasive in nature. Hackers scoured the internet in search for devices with factory set usernames and passwords. They targeted devices from a specific manufacturer and exploited them to muster an attack so immense that it affected millions of legitimate users and sustained for a significant amount of time. These pervasive devices with internet connection - commonly known as IoT - seemingly harmless while working alone can have earth shattering effects if victimized in large numbers.

We understood from this security fallout that two major faults - one of them being devices with default username and password - played the most decisive role in this attack. But we cannot rely only on the users to be vigilant enough to always take proper measures in changing default username and password. Moreover, the second major flaw of such system is the staggering number of similar devices throughout the internet. If one particular type of IoT devices show vulnerability to an attack, all of these devices require revamped security to overcome such flaw. Yet, IoT devices tend to work in collaboration with one another where such collaborative environments remain heterogeneous. Additionally, IoT devices in a particular environment tend to behave is specific ways.

From our perception of the attack by *Mirai Botnet*, we, here, have proposed a solution which attempts to answer two **Research Questions**.

RQ1: Can we propose an encryption scheme that not only depends on long-term secrets but collaborative behavior of a particular environment?

To answer **RQ1**, we have implemented forward secrecy in public key encryption. The session key for the forward secrecy is generated from the behaviors of the IoT devices. The session keys, coupled with long-term secret form the secret for encryption of messages.

RQ2: Can we reflect the noise in behavior of an environment while decrypting an encrypted message?

To answer **RQ2**, we have implemented our proposed protocol using weather sensor data gathered from SensorScope Sensor Network spread through St. Bernerd and Le Genepi. We introduced noise in data gathered from these sensors and emulated them as behavior from a particular environment. Our result analysis shows that as we introduce more noise in our behavior data, we find that larger percentage of encrypted message bits are decrypted inaccurately.

1.2 Security Model

Lattice based encryption is relatively recent addition in cryptography, ushered in by Ajtai [1], which can withstand even quantum algorithm based cryptanalysis. Regrettably, the same cannot be said for other popular cryptographic tools, such as, Discrete Logarithm problem or Integer Factorization, which can easily be broken by Shor's Algorithm for Quantum Computers [54]. As of today, we do not have any algorithm to solve Lattice based *Closest Vector Problem (CVP)* in polynomial time [33], which makes CVP a perfect candidate for building foundation of Public Key Cryptography. It can also be shown that a certain machine learning problem called *Learning With Error (LWE)* is as hard to solve as CVP [48]. Moreover, Chris Peikert proposes a *non-quantum reduction* from the variants of the shortest vector problem to corresponding versions of the LWE problem which has given rise to efficient public-key cryptosystems [43] and our proposed protocol is built upon the foundation of his work. However, no matter how strong a public key encryption mechanism may be, once implemented, it always runs the risk of being compromised by merely leaking the private key or by using a default one. *Forward Secrecy* in public key cryptography mitigates the risk of a secret key being leaked when the same secret key is being used for a long time, by generating temporary session keys from the long-term secrets using some one-way function. Our main focal point was to formulate a protocol that dose not allow an attacker to 'Break' the scheme before the session expires. Moreover, two communicating parties can verify their membership to a common group by sharing a common shared key as can be seen in [2] where the membership verification to a group is done by Bloom Filters. In our proposed protocol to verify group membership, we used the shared secret key that is generated by accessing the same environment variables.

In this work, our goal is to propose a Public Key scheme with forward secrecy and group membership verification for pervasive devices where we can ensure three goals. **First**, our target is to make sure that the sender and the receiver of the messages must be from the same environment. **Second**, we aspire to generate session keys for both communicating parties at the start of each session of communication by observing the environment variables. This means, the receiver can only decipher a message successfully when she has access to the same session key as the sender. **Third**, we allocated slim margin of error while observing data from the environment so that such noise do not hinder encryption or decryption of the messages.

We considered a scenario where the long-term secret key is compromised and we came up with a solution where we generated session keys from the data provided by pervasive devices. Such data are used to encrypt and share the session key between the communicating parties. We implemented our customized version of encrypting the key with *One-Time Pads* to allow noisy data from the pervasive devices to be used in encryption and decryption of the session key at sender's and receiver's end respectively. Later, the session keys are used as the secret key to create a One-Way hash of the public key using *Short Integer Solutions* [34] by both the communicating parties. Finally, the hash of the Public Key is used to encrypt and decrypt messages using *Learning With Error* based Public Key encryption.

1.3 Our Contributions

The protocol we proposed in this work focuses on communication between two parties where both of them have access to same environment variables. These variables may be generated from private sensor cloud like the array of sensors found in a smart car, weather stations or even in today's cellphones. Our assumptions in this work are that these sensors are heterogeneous in nature and replicating the sensor data is difficult, even if some of the sensors in a particular environment is compromised. To the best of our knowledge, achieving forward secrecy and verifying group membership using heterogeneous pervasive device data as session keys in Lattice-based encryption mechanism have never been done before. It should also be realized that, in our proposed protocol, the observation of sensor variables may be accompanied with significant amount of noise which, we have shown, has proportionally reflected on the decryption accuracy of the messages encrypted following our protocol. After the session key is shared, it is used to calculate the public key by both parties of communication. We used Lattice-based encryption mechanism since lattices have shown resilience to any contemporary cryptanalysis and their hardness can be proven mathematically. Moreover, lattices give us flexibility in working with matrices, which, in our case, are predicted and/or generated by acquired data originating from array of sensors. We described the whole process elaborately in section 4.5.

1.3.1 Implementation of the Protocol

We implemented our protocol using weather sensor data of Sensorscope from 39 different locations. After acquiring weather sensor data of one month from [52] and [51], we used them to train LSTM-Network model to later predict time series with noisy data. As we wanted to use the data to generate session keys, waiting for more than one month to get all the data for the next session key was not feasible. Thus, to demonstrate our protocol, we took small portion of acquired data and added little noise to them. We used our trained LSTM-network model to predict the rest of the sensor data. Furthermore, as our weather data spanned for more than one month, and for each day, we had several observations, we considered, data from each day belong to a separate classes. To acquire maximized interclass distance and minimized intra-class distance, we changed the axes of original data using LDA. The Eigenvectors of the axes are used to change the axes of the predicted dataset. We had 9 features from each dataset - the different weather sensors. Upon applying cross-correlation between the features, we ended up with 45 features. We used these 45 features from both original data and predicted data to generate the session keys for communication between two parties. We elaborated the implementation procedure thoroughly in chapter 5. The decryption accuracy of our protocol is discussed in chapter 6.

1.4 Thesis Organization

In this dissertation we contemplated on proposing a framework to secure communication over the cloud using the diversity of IoT devices. Our work in this thesis is organized as follows. In chapter 2, we discuss the preliminary tools we used throughout our work. The research works done to secure different layers of IoT architecture are discussed in chapter 3. Our proposed framework and its details are elaborated in chapter 4. The implementations we have done to experiment on our proposed framework is detailed in chapter 5 whereas the results of such experiments by varying different parameters are described in chapter 6. Finally, we conclude this dissertation with chapter 7 where we have shed light on future directions the researchers might follow to progress this work.

Chapter 2

Preliminaries

This chapter includes the tools and techniques that we have gone through while seeking a comprehensive security solution for IoT architecture. Some of these topics may seem irrelevant to this dissertation but present the readers an entry point to the world of cryptography and artificial intelligence. In this chapter, we constraint our discussion in two parts. We first discuss the basis of *Asymmetric Key Cryptography* and its mathematical background. Later we shift our focus to present a background study on *Artificial Neural Network*.

2.1 Asymmetric Key Cryptography

The general notion of Asymmetric Key Cryptography and the distinctive feature that separates it from its Symmetric counterpart is that, in the former, the key to encrypt messages is different from the one to decrypt it. This is not the case for Symmetric Key Cryptography where both the encryption and decryption keys are the same. This distinctive feature of Asymmetric Key Cryptography has given rise to *Public Key Cryptography*, where owner of the key-pair shares one key in the pair with everyone - the *Public Key* - and keeping the other part secret - the *Private Key*. Either of the keys can be used to encrypt a message; the opposite key from the one used to encrypt the message is used for decryption. In this section we focus our discussion on how these key-pairs can be generated and what purpose they can serve as the cryptographic primitives.

We start our discussion by describing three NP-hard problems - *Integer Factorization Problem* [30], *Discrete Logarithmic Problem* [27] and *Closest Vector Problem* [33]. All these three have no algorithm to solve an average instance of the problem efficiently in polynomial time, which means these problems have been used to generate key-pairs and cryptographic primitives in different public key cryptographic schemes.

2.1.1 Integer Factorization Problem

To decompose a sufficiently large composite integer into a product of smaller integers is not a problem that can be solved by any already known efficient algorithm in polynomial times. Yet this has not been proven either that there cannot be any algorithm that can solve this problem in polynomial time. This perceived hardness of decomposition of a large integer into its constituent factors is known as *Integer Factorization Problem*. When the smaller constituent smaller integers are primes, the problem is known as *Prime Factorization*. In [29], Kleinjung et al. has shown that even with hundreds of machines, a 768 bit number takes at least two years to be decomposed.

Example. We can easily calculate the product of two prime numbers: p_1 and p_2. The Prime Factorization Problem states that given $large_{number} = p_1 \times p_2$, decomposing $large_{number}$ into its prime factors is not solvable in polynomial times when none of p_1 and p_2 is known. However, if someone knows any of the primes, the other prime factor of $large_{number}$ can easily be found using trivial division operation.

Application: RSA. In RSA, this asymmetry is based on the practical difficulty of the factorization of the product of two large prime numbers. The acronym RSA is made of the initial letters of the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who first publicly described the algorithm in 1978 [35]. Here we have presented a worked example of RSA algorithm. The algorithm involves four steps:

- **Key Generation** This steps to generate the *Public Key* and the *Private Key* is given below:

  1. Select two prime numbers,p and q, that are sufficiently large. For this example we have selected very small primes compared to the ones used in practical implementation.

  $$p = 61; q = 53$$

  2. Compute $n = pq$ giving

  $$n = 61 \times 53 = 3233$$

  3. Compute the totient of the product as $\lambda(n) = lcm(p-1, q-1)$ giving

  $$\lambda(3233) = lcm(60, 52) = 780$$

  4. Choose any number $1 < e < 780$ that is coprime to 780. Choosing a prime number for e leaves us only to check that e is not a divisor of 780. $let, e = 17$

  5. Compute d, the modular multiplicative inverse of $e \pmod{\lambda(n)}$ yielding, $d = 413$. The process is described in Appendix A.

  The *Public Key* is $(n = 3233, e = 17)$ whereas the *Private Key* is $(n = 3233, d = 413)$.

- **Key Distribution** Suppose that Bob wants to send information to Alice. If they decide to use RSA, Bob must know Alice's public key to encrypt the message and Alice must use her private key to decrypt the message. To enable Bob to send his encrypted messages, Alice transmits her public key (n, e) to Bob via a reliable, but not necessarily secret, route. Alice's private key (d) is never distributed.

- **Encryption** For a padded plaintext message m, where $m = 65$, the encryption function is

$$\begin{aligned} c(m) &= m^e \pmod{n} \\ &= 65^{17} \pmod{3233} \\ &= 2790 \end{aligned} \tag{2.1}$$

  Thus, the encryption of 65 is 2790.

- **Decryption** The decryption function for cipher text, $c = 2790$ is

$$\begin{aligned} m(c) &= c^d \pmod{n} \\ &= 2790^{413} \pmod{3233} \\ &= 65 \end{aligned} \tag{2.2}$$

2.1.2 Discrete Logarithm Problem

To introduce the *Discrete Logarithm Problem*, we start with its first application in cryptography. Whit Diffie and Martin Hellman in 1976 first applied this problem in sharing keys secretly among two parties [17]. This is a cryptographic method to exchange keys between two parties without having to actually share it on an unsecure line. In this method two parties who want to share a symmetric key generate a common public key for both of them along with a private key for each of them. Now the public and private parts of the key are mixed together generating two separate keys for two parties and the newly generated key in exchanged. The received key is again mixed with the private part of keys of each individuals and it results in a symmetric key, which is only known by the sharing parties and no one else. Here is an example of the protocol, where Alice and Bob shares a common secret.

1. Alice and Bob agree to use a modulus $p = 23$ and base $g = 5$ (5 is a primitive root of modulo 23. The process of finding primitive root is described later).

2. Alice chooses a secret integer $a = 6$, then sends Bob $A = g^a \pmod{p}$

$$A = 56 \pmod{23} = 8$$

3. Bob chooses a secret integer$b = 15$, then sends Alice $B = g^b \pmod{p}$

$$B = 515 \pmod{23} = 19$$

4. Alice computes $s = B^a \pmod{p}$

$$s = 196 \pmod{23} = 2$$

5. Bob computes $s = A^b \pmod{p}$

$$s = 815 \pmod{23} = 2$$

6. Alice and Bob now share a secret (the number 2).

Both Alice and Bob have arrived at the same value, because,

$$\begin{aligned} A^b \pmod{p} &= (g^a \pmod{p})^b \pmod{p} \\ &= (g^a)^b \pmod{p} \\ &= (g^b)^a \pmod{p} \\ &= (g^b \pmod{p})^a \pmod{p} \\ &= B^a \pmod{p} \end{aligned} \tag{2.3}$$

To understand the underlying mechanism and mathematical background behind this secret sharing protocol, we have to get ourselves accustomed to the following concepts.

- **Groups:** A Group is an abstract mathematical concept which is defined by a set of elements and one group operation. The operation must be similar to ordinary integer arithmetic but has to be defined on pairs of elements of the Group. The result of the operation must also be an element of the same Group.

  For example, let the set, integer modulo n forms a group under the operation of addition modulo n. If $\mathbb{Z}_n$ is the set of integers modulo n, then,

  $$\mathbb{Z}_n = \{0, 1, ..., n-1\}$$

  If we add any two elements in the group and get the modulo of n, we will get an element in the group. However, this is not the only property that a Group has to hold. If G° is a group which is defined over the operation $\circ$, then it must hold the following properties.

  1. **Closeness:** If, $a, b \in G^\circ$, then, $a \circ b = c \in G^\circ$
  2. **Associativity:** If, $a, b, c \in G^\circ$, then, $(a \circ b) \circ c = a \circ (b \circ c)$
  3. **Neutral Element:** There exists a neutral element, e, for which $a \circ e = a$. In the previous example, 0 is the neutral element in $\mathbb{Z}_n$.
  4. **Inverse Element:** There exists an inverse element, $a - 1$, for each element $a \in G^\circ$, for which $a \circ a^{-1} = e$. In the previous example, for each element in $\mathbb{Z}_n$, there can be found another element in the same group for

which the addition modulo n of those two elements will be the neutral element, 0. For example,

$$\begin{aligned} 0+0 \pmod n &= 0 \\ 1+(n-1) \pmod n &= 0 \\ 2+(n-2) \pmod n &= 0 \\ &. \\ &. \\ &. \\ (n-1)+1 \pmod n &= 0 \end{aligned} \tag{2.4}$$

- **Abelian Group:** An *Abelian Group* has all the properties of a *Finite Group* along with the property of Commutativity. That is, for a group G° to be Abelian, elements $a, b \in G^\circ$ has to show the commutative property: $a \circ b = b \circ a$.

- **Multiplicative Group:** One of the group operations - multiplication modulo n- has found rather importance in cryptography because of it can be related with prime numbers. From Appendix A, we have seen that, Modular Multiplicative Inverse of a modulo n exists only if $gcd(a, n) = 1$. Now, for a set of elements can only be a group under the operation of multiplication modulo n when there exists a Modular Multiplicative Inverse of each element of the set.

  For example, for the set $\mathbb{Z}_9$ to become a multiplicative group $\mathbb{Z}_9^*$, it has to get rid of the elements $a \in \mathbb{Z}_9$ for which $gcd(a, 9) \neq 1$. Thus,

  $$\mathbb{Z}_9^* = \{1, 2, 4, 5, 7, 8\}$$

  Now, if we select an group operation that is multiplication modulo *prime integer, p*, we can take all the integers less than p and greater than 0 to form a multiplicative group. Such as,

  $$\mathbb{Z}_p^* = \{1, 2, 3, ..., p-1\}$$

- **Finite Group:** If the number of elements of a Group is finite, then the group is called a Finite Group. For example $\mathbb{Z}_9^*$ is a finite group.

- **Cardinality:** The Cardinality or the **Order** of a group is the number of elements present in that group. For example, the Cardinality of $\mathbb{Z}_9^* = ||\mathbb{Z}_9^*|| = 6$.

- **Order of an Element:** Each element in a *Multiplicative Group* can generate certain number of elements of the same group by multiplying with itself . For example, if we consider the group $\mathbb{Z}_{11}^*$, where,

  $$\mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

Now, the element $a \in \mathbb{Z}_{11}^*$ can generate certain elements in the group $\mathbb{Z}_{11}^*$. if $a = 3$,

$$\begin{array}{llll} a^1 \pmod{11} & \equiv 3; & a^4 \pmod{11} & \equiv 4; \\ a^2 \pmod{11} & \equiv 9; & a^5 \pmod{11} & \equiv 1; \\ a^3 \pmod{11} & \equiv 5; & a^6 \pmod{11} & \equiv 3; \end{array}$$

As we can see, element $a = 3 \in \mathbb{Z}_{11}^*$ generates $\{3, 9, 5, 4, 1\}$ by multiplying with itself. After that the process repeats itself by generating the same elements. The minimum number of operations that it took for $a = 3$ to generate the neutral element $e = 1$, is called the order of the element.

$$ord(a) = ord(3) = 5$$

- **Generator:** If the order of an element in a Group is same as the Cardinality of that Group, then the element is called a *Generator* of the Group. A Generator is also known as the **Primitive Element** or **Primitive Root**.

  For example, the element $a = 2$ is a Generator for Group, $\mathbb{Z}_{11}^*$, because

$$\begin{array}{llllll} a^1 \pmod{11} & \equiv 2; & a^4 \pmod{11} & \equiv 5; & a^7 \pmod{11} & \equiv 7; \\ a^2 \pmod{11} & \equiv 4; & a^5 \pmod{11} & \equiv 10; & a^8 \pmod{11} & \equiv 3; \\ a^3 \pmod{11} & \equiv 8; & a^6 \pmod{11} & \equiv 9; & a^9 \pmod{11} & \equiv 6; \\ & & & & a^{10} \pmod{11} & \equiv 1; \end{array}$$

- **Cyclic Group:** A Group, G, is called *Cyclic* if it contains an element (α) with maximum order, such that $ord(\alpha) = ||G||$. For every prime p, $\mathbb{Z}_p^*$ is a *Finite, Abelian, Cyclic* group.

Now as we go back to our previous example of Diffie-Hellman Key exchange protocol, where we took $g = 5$ and $p = 23$, since 5 is a *Generator* or *Primitive Root* of the group $\mathbb{Z}_{23}^*$. But to relate Discrete Logarithm Problem with this key exchange protocol, let us consider, $\beta \in \mathbb{Z}_p^*$. Now, β can be expressed as,

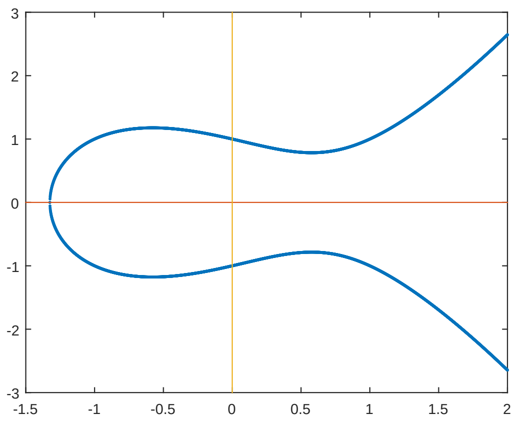
$$g^x \equiv \beta \pmod{p} \tag{2.5}$$

Discrete Logarithm Problem in $\mathbb{Z}_p$

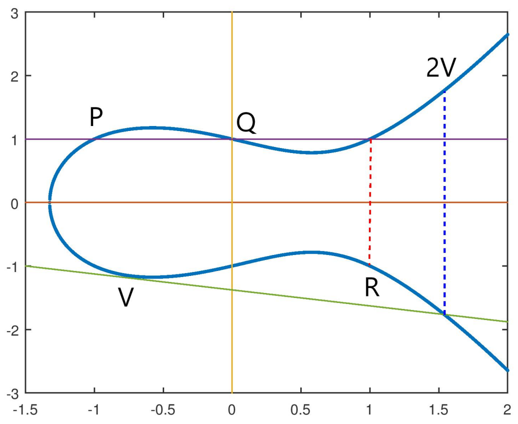
If Eve is paying attention to the communication between Alice and Bob, Eve will be given $p, \beta \in \mathbb{Z}_p^*$ and g and will need to find x such that equation 2.5 holds. Such an integer x is the discrete logarithm of β to the base g. Thus, $x = ind_g^\beta \pmod{p}$ (another word for Discrete Logarithm is *index*) [27]. Unfortunately for Eve, finding such x is a computationally hard problem given a large enough prime p. And this is the basis of Discrete Logarithm Problem.

Elliptic Curves as Cyclic Groups

Similar to multiplicative groups, Elliptic curves are potent candidates as cyclic groups to be utilized in cryptography.



(a) Elliptic Curve created by $y^2 = x^3 - x + 1$



(b) Group Operation on Elliptic Curve

Figure 2.1: Elliptic Curve as a Cyclic Group

Elliptic Curve Background. The main motivation behind Elliptic Curves is the search for a cyclic group other than $\mathbb{Z}_p^*$ for which Discrete Logarithm Problem is difficult and the key is shorter than the multiplicative groups. We could consider polynomials like $x^2+y^2 = r^2$ or $ax^2+by^2 = r^2$ which provide specific shapes. However for using in cryptography, we need to consider polynomial over $\mathbb{Z}_p$. Similarly, elliptic curves are also some polynomial defined over $\mathbb{Z}_p$ and show cyclic group properties. Why Elliptic curves and not some other polynomial shows such properties are beyond the scope of this work. Readers should look into the origin of Elliptic Curves in cryptography from [36].

Definition. The elliptic curve over $\mathbb{Z}_p$ is the set of all pairs $(x, y) \in \mathbb{Z}_p$ such that

$$\begin{aligned} & y^2 \equiv x^3 + ax + b \pmod{p} \\ & \text{where, } \ p > 3 \\ & \text{and, } \ (a, b) \in \mathbb{Z}_p \\ & \text{and, } \ 4a^3 + 27b^2 \not\equiv 0 \pmod{p} \end{aligned} \tag{2.6}$$

The elliptic curve is defined together with a point θ at *infinity* along y-axis. Moreover, from figure 2.1a, we can see that elliptic curves are symmetric along x-axis. If we connect any point on the curve to the point at *infinity* θ with a line, the connecting line will be parallel to the y-axis.

Group Operations on Elliptic Curve. For a Discrete Logarithm Problem, we need a Cyclic Group with a set of elements and a group operation. Equation 2.6 gives us the set of elements. Two group operations are defined over the Elliptic curves, namely *Point Addition* and *Point Doubling*. In figure 2.1b, we show examples of both the operations.

- **Point Addition:** As we can see from the figure, points P and Q on the curve are joined with a line which intersects the curve on a third point. The point is reflected on the other side of the x-axis, since the curve is symmetric along that axis. Thus we get a point R, by adding two points, P and Q, on the curve.

- **Point Doubling:** A tangent, drawn along the point V on the elliptic curve in figure 2.1b, intersects the curve on another point. The intersection point is reflected along the x-axis and we get the point $2V$. The process done on the elliptic curve is known as point doubling.

Analytical Expressions for Group Operations. For an elliptic curve $E : y^2 \equiv x^3 + ax + b \pmod{p}$ and points $P_1, P_2 \in E$, where, $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, we calculate the analytical expressions for the group operation in the following way.

As the equation of the curve, E is given as:

$$y^2 = x^3 + ax + b \tag{2.7}$$

We get the values of x_3 and y_3 from equation 2.8.

$$\begin{array}{rl} x_3 = & m^2 - x_1 - x_2 \pmod{p} \\ y_3 = & m(x_1 - x_3) - y_1 \pmod{p} \\ \text{where,} & \\ m = & \left\{ \begin{array}{lll} \frac{y_2-y_1}{x_2-x_1} \pmod{p} & ;\text{ if } P_1 \neq P_2 & [\text{Point Addition}] \\ & & \\ \frac{3x_1^2+a}{2y_1} \pmod{p} & ;\text{ if } P_1 = P_2 & [\text{Point Doubling}] \end{array} \right\} \end{array} \tag{2.8}$$

The derivation of the equations are given in Appendix B.

Example. Here we present an example on how we calculate point addition and point doubling on Elliptic curves. Let, $y^2 = x^3 + 2x + 2 \pmod{17}$. For correspondence with the equation 2.8, here we have, $a = 2$, $b = 2$ and $p = 17$. Let a point be $Q(5, 1)$.

- **Point Doubling:** In point doubling, the slope,

$$m = \frac{3 \times (5^2) + 2}{2 \times 1} = (2)^{-1} \times 77 \pmod{17}$$

The calculation of *Modular Multiplicative Inverse* of 2 (mod 17) is shown in Appendix A. From the calculation, we see,

$$2^{-1} \pmod{17} \equiv 9$$

Thus,

$$\begin{aligned} m &\equiv 9 \times 77 \pmod{17} \\ &\equiv 9 \times 9 \pmod{17} \\ &\equiv 81 \pmod{17} \\ &\equiv 13 \pmod{17} \end{aligned} \tag{2.9}$$

Thus, after point doubling, the point $2Q(x_3, y_3)$ is calculated by:

$$\begin{aligned} x_3 &\equiv m^2 - x_1 - x_2 \pmod{17} \\ &\equiv 13^2 - 10 \pmod{17} \\ &\equiv 6 \pmod{17} \\ \\ y_3 &\equiv m(x_1 - x_3) - y_1 \pmod{17} \\ &\equiv 13 \times (5 - 6) - 1 \pmod{17} \\ &\equiv 13 \times 16 - 1 \pmod{17} \\ &\equiv 3 \pmod{17} \end{aligned} \tag{2.10}$$

Thus, $2Q \equiv (6, 3) \pmod{17}$

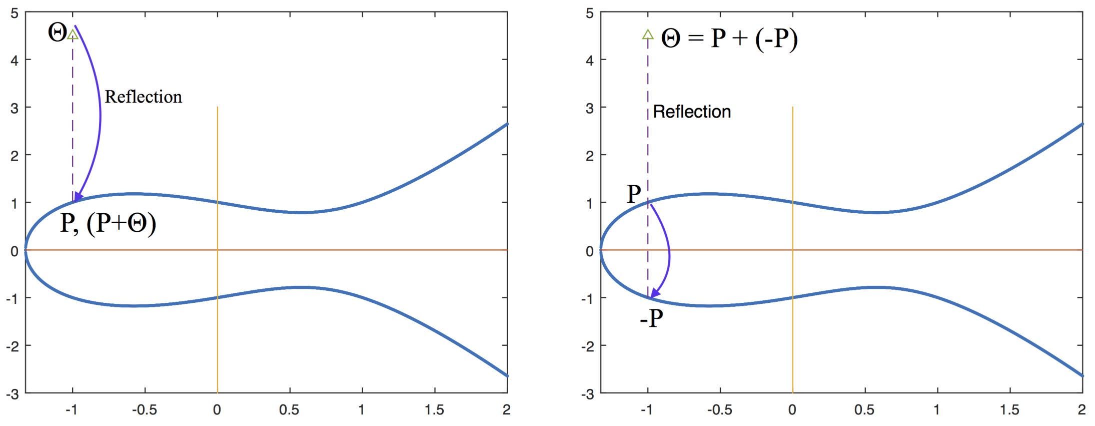
- **Point Addition:** If we want to add $2Q(6, 3)$ from the previous example to $Q(5, 1)$, this will require point addition on Elliptic Curve. For point addition, slope m is calculated by

$$\begin{aligned} m &\equiv \frac{y_2 - y_1}{x_2 - x_1} \pmod{17} \\ &\equiv \frac{1 - 3}{5 - 6} \pmod{17} \\ &\equiv \frac{2}{1} \pmod{17} \\ &\equiv 1^{-1} \times 2 \pmod{17} \\ &\equiv 2 \pmod{17} \end{aligned} \tag{2.11}$$

Now let $R = Q + 2Q$ and $R \equiv (x_3, y_3) \pmod{17}$. So,

$$\begin{aligned} x_3 &\equiv 2^2 - 6 - 5 \pmod{17} \\ &\equiv -7 \pmod{17} \\ &\equiv 10 \pmod{17} \\ y_3 &\equiv m(x_3 - x_1) - y_1 \pmod{17} \\ &\equiv 2(10 - 5) - 1 \pmod{17} \\ &\equiv 9 \pmod{17} \end{aligned} \tag{2.12}$$

Thus, $R \equiv (10, 9) \pmod{17}$



(a) Elliptic Curve Neutral Element (b) Elliptic Curve Inverse Element

Figure 2.2: Group Properties of Elliptic Curve

Elliptic Curve Group Properties. Elliptic curves hold all properties to be considered as groups. Herein, we will discuss two vital properties of groups that are held by elliptic curves.

- **Neutral Element:** For a set of elements in a group E, there should be a neutral element θ for which for all elements $P \in E$, $P + \theta = P$.

  As we can see from equation 2.6, elliptic curve defines an element at infinity, θ along y-axis and the line, intersecting any point on the curve and θ will be parallel to y-axis. Thus, if we perform *Point Addition* on the neutral element θ and any point on the curve, P, the reflection will be along y-axis on P. Figure 2.2a depicts the point addition operation between P and θ.

- **Inverse Element:** In a group E, for each element P, there is an inverse element $(-P)$, for which $P + (-P) = \theta$, where θ is the neutral element.

  Since Elliptic curves are symmetric along x-axis, if $P \in E$ and $P = (x, y)$, then $(-P) \in E = (x, -y)$. As we can see from the figure in 2.2b, the point addition operation of P and $(-P)$ will be reflected along y-axis back to the neutral element θ at infinity. This is because the line connecting P and $(-P)$ is parallel to y-axis and such line connects θ at infinity.

Group Cardinality of Elliptic Curves. According to Hasse's theorem, the group cardinality, $\#E$ of the elliptic curve, described by equation 2.6, can be estimated by the approximation given by equation 2.13.

$$||\#E - (p+1)|| \leq 2\sqrt{p} \tag{2.13}$$

For a large enough prime number, equation 2.13 can be approximated as

$$\#E \approx p$$

Elliptic Curve Discrete Logarithm Problem

Given an elliptic curve E, we consider a primitive element P and another element T on the curve. The *Discrete Logarithm* problem is to find the number of group operations it takes to go from P to T. In other words, if $P + P + ... + P = dP = T$, then the Discrete Logarithm problem is to find the integer d, where P and T are given.

If the elliptic curve is chosen carefully (i.e., NIST proposed elliptic curves [20]), the best known algorithm to compute the EC-DLP requires approximately $\sqrt{p}$ steps, where p is the finite field over which the curve is defined (i.e., a large prime). All Elliptic Curve protocols rely on this hardness of the EC-DLP.

Elliptic Curve Diffie-Hellman Key Exchange

We show here a straight forward adoption of Diffie-Hellman Key Exchange protocol in $\mathbb{Z}_p$ using elliptic curves. Let, Alice and Bob want to exchange keys and they decide on a particular elliptic curve, E and primitive element $P = (x, y)$. The following steps describe the key exchange protocol.

1. Alice and Bob both selects their secret keys: a and b respectively, where

$$a, b \in \{2, 3, ..., \#E - 1\}$$

2. Alice calculates A by hopping around the elliptic curve E, a times. Similarly, Bob calculates B by b times hopping around the same elliptic curve. Both of them start their hop from the primitive element P. A and B are the public keys of Alice and Bob respectively. Thus,

$$A = aP = \text{ point on the curve, } E = (x_A, y_A)$$

$$B = bP = \text{ point on the curve, } E = (x_B, y_B)$$

3. Alice and Bob exchange their public keys with each other.

4. Upon receiving Bob's public key,B, Alice computes $a.B = (x_{AB}, y_{AB})$, by hopping along the elliptic curve E, a times, starting from point B. At the same time, after Bob receives Alice's public key, A, Bob computes $b.A = (x_{AB}, y_{AB})$ by hopping along the curve b times, starting from point A.

5. Since both Alice and Bob have come up with the shared secret $T = (x_{AB}, y_{AB})$, they can use either of these two values to encrypt and decrypt messages.

Discussion. The hardness of Discrete Logarithm Problem depends on calculating the number of hops it takes to get from $P = (x, y)$ to $T = (x_{AB}, y_{AB})$. It is computationally expensive to calculate these values. If we look at the described key exchange protocol both Alice and Bob should also face the same problem, since they will also have to calculate the hops sequentially - a times for Alice and b times for Bob. To alleviate the computational complexity of calculating points on the curve, given the hop count, [23] includes a list of algorithms i.e., *Double-and-Add* method, *Windowed* method, *Sliding Window* method, etc.

In the following section, we discuss the *Double-and-Add* method as an example to explain the efficiency of calculating a point of a curve, if the number of hops are already determined.

Algorithm 1 Double-and-Add Method

```
Require: Scalar Multiple $2 \leq a < \#E$, Primitive Element $P \in E$
1: $b \in \mathbb{Z}_2^{len} \leftarrow$ ConvertToBinary($a$) {Convert $a$ into binary string $b$ of length $len$}
2: $Q \leftarrow P$
3: for $i = 2$ to $len$ do
4:     $Q \leftarrow Q + Q$ {Point Doubling}
5:     if $b[i] ==' 1'$ then
6:         $Q \leftarrow Q + P$ {Point Addition}
7:     end if
8: end for
9: return $Q$
```

Double-and-Add Method

In this part, we describe the *Double-and-Add* method to compute the scalar multiple of P, where P is a *Primitive Element* on an Elliptic Curve. The method works by converting the scalar multiple, a, of P into a binary string and then traversing the string from left to right. For each 0 in the string, the method doubles the point on the curve and for each 1 the point is doubled at first and added with the primitive element P. Finally, we receive the point on the curve equivalent to aP.

Algorithm 1 describes the process in brief. As the binary string is traversed from left to right, this method is also known as *Left-to-Right* method.

Table 2.1: Example of Double-and-Add Method

Step	**Operation**	**Comment**	**Decimal Scalar Multiple**	**Binary Scalar Multiple**
ϕ	$Q \leftarrow P$	Initial Step	$1_{10}P$	1_2P
$1a$	$Q \leftarrow Q + Q$	Point Doubling	$2_{10}P$	10_2P
$1b$	$Q \leftarrow Q + P$	Point Addition	$3_{10}P$	11_2P
$2a$	$Q \leftarrow Q + Q$	Point Doubling	$6_{10}P$	110_2P
$3a$	$Q \leftarrow Q + Q$	Point Doubling	$12_{10}P$	1100_2P
$3b$	$Q \leftarrow Q + P$	Point Addition	$13_{10}P$	1101_2P
$4a$	$Q \leftarrow Q + Q$	Point Doubling	$26_{10}P$	11010_2P

Example. Let us calculate the value of $26P$. The scalar multiple here is 26_{10} given as a decimal value. If we convert 26 into binary string, we get 11010_2. Thus calculating the value of $26_{10}P$ is equivalent to calculating 11010_2P. Now we describe the steps of calculating the value of $26_{10}P$ in table 2.1.

Discussion. As we can see in the example given in table 2.1, calculating $26P$ took only 6 steps to be computed. A brute-force method still takes 26 steps to compute the value. This results in computational efficiency by reducing the complexity from $O(n)$ to $O(\log_2 n)$ where the finite field of the Elliptic Curve is defined by a n-bit prime number.

2.1.3 Closest Vector Problem

Previously, we have discussed two computationally hard problems based on which many cryptographic tools have prevailed and elaborately used. One major issue regarding *Integer Factorization Problem* and *Discrete Logarithm Problem* is that no one can prove that there *can not* be an algorithm that solves the problems in polynomial times. As Quantum Computers are increasingly becoming a reality, the risk of quantum algorithms solving such problems are starting to be realized throughout the cryptographic community. Moreover, these computational problems do not ensure every test-cases to be as hard as the next one. For example, Integer Factorization problem can be hard to solve given very large prime numbers but the same cannot be said for any randomly selected prime.

A ground-breaking solution to such problems was established on foundation laid by Ajtai in [1] where he proved the hardness of lattice problems in average cases whose details, however, is not the focus of our discussion in this work. To understand the *Closest Vector Problem*, we need to understand the basics of Lattices.

Lattice. Let $\mathbb{R}^m$ be an m-dimension Euclidean space. A *Lattice* in $\mathbb{R}^m$ is the set

$$\mathcal{L}(\mathbb{b}_1, ..., \mathbb{b}_n) = \left\{ \sum_{i=1}^{n} x_i \mathbb{b}_i : x_i \in \mathbb{Z} \right\} \tag{2.14}$$

of all integer combinations of n linearly independent vectors $\mathbb{b}_1, ..., \mathbb{b}_n$ in $\mathbb{R}_m$ where $m \geq n$. That is, n linearly independent vectors, each with m dimensions and there integral combinations, form a Lattice. Each point in the lattice is described by the equation 2.14, from where we can see that the lattice points are spread throughout the $\mathbb{R}^m$ plane. The integers n and m are called the *rank* and *dimension* of the Lattice $\mathcal{L}$, respectively. The sequence of vectors $\mathbb{b}_1, ..., \mathbb{b}_n$ is called a *Lattice Basis* and can be conveniently represented as a matrix. For example,

$$B = [\mathbb{b}_1, ..., \mathbb{b}_n] \in \mathbb{R}^{m \times n}$$

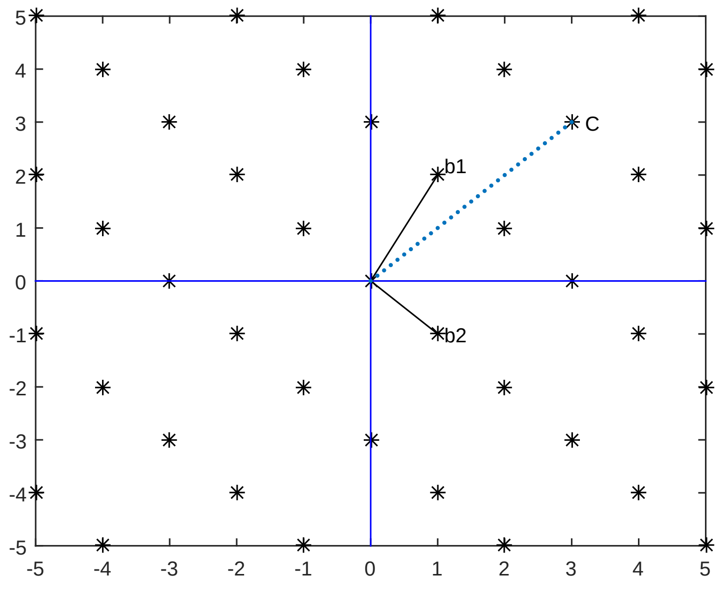


Figure 2.3: Example of a Lattice in $\mathbb{R}^2$

The basis vectors are the columns of matrix B which are linearly independent. Thus, the lattice notation $\mathcal{L}$ can be written more compactly as

$$\mathcal{L}(B) = \{B\mathbb{x} : \mathbb{x} \in \mathbb{Z}^n\}$$

We can calculate $\mathcal{L}$ by regular matrix-vector multiplication. Graphically, a lattice can be described as the set of intersection points of an infinite, regular n-dimensional grid. It should be mentioned here that a lattice n independent vectors each with m-dimensions will form an infinite n-dimensional grid.

A 2-dimensional example is shown in Figure 2.3. The lattice points are spread through the whole $\mathbb{R}^2$ plane. We are just depicting a small portion of the lattice points. Each point on the figure is part of the lattice. There, the basis vectors are

$$\mathbb{b}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \; ; \quad \mathbb{b}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Moreover, each point on the figure 2.3 is integral combination of $\mathbb{b}_1$ and $\mathbb{b}_2$. For example, the vector, c is obtained by $2\mathbb{b}_1 + \mathbb{b}_2$.

Definition of CVP. *Closest Vector Problem* or *CVP* is one of the computationally hard problems formulated on Lattices. Given a lattice $\mathcal{L} \subset \mathbb{R}^n$, a target point $t \in \mathbb{R}^n$ and a distance bound d, the *CVP* asks for a lattice point $v \in \mathcal{L}$ at distance $||t-v|| \leq d$, provided such lattice point exists. In the "Exact" version of CVP, the distance bound

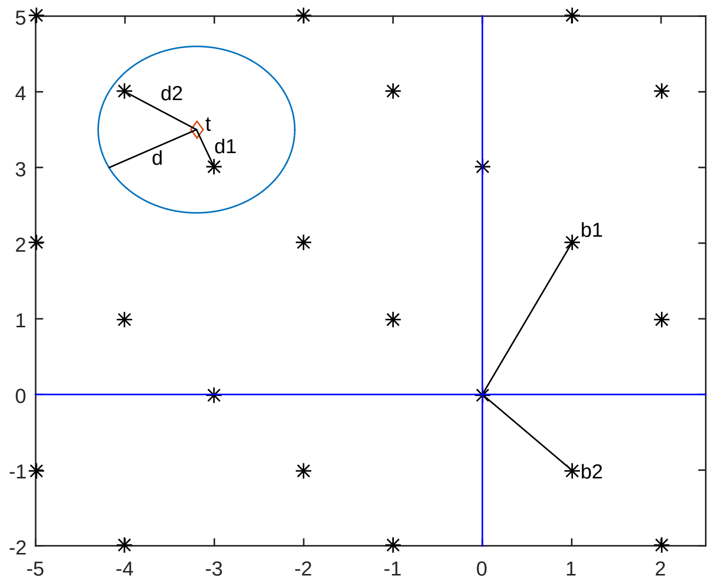


Figure 2.4: Example of Closest Vector Problem in $\mathbb{R}^2$

is considered to be the distance $d = \mu(t, \mathcal{L}) = min_{v \in \mathcal{L}} ||t - v||$. In other words, in exact CVP, the task is to find the lattice point which is the closest to a given point. In the "Approximate" version, CVP_γ, the distance bound is set as $d = \gamma.\mu(t, \mathcal{L})$, which means to find any lattice point within a n-dimensional sphere centering the point t having a radius of d. Here $\gamma \geq 1$ is a coefficient that determines the length of the radius of the sphere.

In figure 2.4, we have given an example of a 2-dimensional lattice generated by two basis vectors $\mathbb{b}_1$ and $\mathbb{b}_2$. We also considered an arbitrary point $t \in \mathbb{R}^2$ space. As we can see from the figure, the closest lattice point is at d_1 distance from t. Let, the point is v_1. In the *Exact* version of CVP, point v_1 should be the only solution to the problem. However, in the *Approximate* version, the distance bound is given by d. As we can see from the figure, point v_1 is inside the 2-dimensional sphere with radius d. Interestingly, the lattice point, with distance d_2 from t, is also inside the sphere. Let, the new point be v_2. Thus, in CVP_γ, both v_1 and v_2 will be a solution to the Closest Vector Problem.

Discussion. As per the example given in figure 2.4, we can solve the problem very easily since the lattice given there has only two dimensions. However, increasing the number of dimensions of basis vectors, the CVP problem becomes more and more difficult to solve within polynomial times. Thus, to solve a lattice based Closest Vector Problem is computationally difficult with increased *ranks* and *dimensions*.

Application of *CVP* in Cryptography

With the introduction of one-way fuction based on *Short Integer Solution* problem by Ajtai[1] in 1996, lattice based cryptography took off. Later Oded Regev's seminal work on *Learning with Error* problem[48] formulated a cryptographic tool based on this machine learning problem. In this section, we are going to discuss the application of these problems in cryptography and why both of them fall in the family of Closest Vector Problem.

Short Integer Solution

Let, $m, n, q \in \mathbb{Z}$ and a matrix A has n rows and m columns and each of the values of A are between 0 and $q-1$, that is $A \in \mathbb{Z}_q^{n \times m}$. Moreover, let $\mathbb{x}$ be a short column vector in terms of Euclidean distance and has m dimensions, that is, $\mathbb{x} \in \{0,1\}^m$. Furthermore, let $m \geq n \log_2 q$ and $||\mathbb{x}|| \leq \sqrt{n \log_2 q}$. The Short Integer Solution asks to find the short vector $\mathbb{x}$, given the matrix A and vector $b = A.\mathbb{x} \in \mathbb{Z}_q^n$.

Example. Let $m = 4, n = 3, q = 10$. Thus, $A \in \mathbb{Z}_{10}^{3\times4}$. Let,

$$A = \begin{bmatrix} 1 & 2 & 9 & 4 \\ 5 & 7 & 4 & 8 \\ 9 & 5 & 7 & 6 \end{bmatrix} \text{ and } \quad \mathbb{x} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}. \text{ Thus, } \quad \mathbb{b} = A.\mathbb{x} = \begin{bmatrix} 14 \\ 17 \\ 22 \end{bmatrix}$$

Now, to solve $\mathbb{b} = A.\mathbb{x}$ where we have 3 equations and 4 unknowns, such as,

$$\begin{aligned} 1.x_1 + 2.x_2 + 9.x_3 + 4.x_4 &= 14 \\ 5.x_1 + 7.x_2 + 4.x_3 + 8.x_4 &= 17 \\ 9.x_1 + 5.x_2 + 7.x_3 + 6.x_4 &= 22 \end{aligned} \tag{2.15}$$

Discussion. Since, equation 2.15 is a non-homogeneous linear system, the general solution to such system consists of a specific solution to the system ($A\mathbb{x} = \mathbb{b}$) and a generic solution to the homogeneous system ($A\mathbb{x} = 0$). However, since the number of equation is less than the number of unknowns in equation 2.15, there can be arbitrary number of solution to the specific solution to the inhomogeneous system. Thus, the solution vector $[x_1, x_2, x_3, x_4]^T$ can be arbitrarily long. As an example, if we consider it as a Closest Vector Problem, the solution vector will be far from the target point t, as shown in figure 2.4. Thus, the solution vector cannot be solved by any known algorithm in polynomial time.

Compression using SIS. As, $A \in \mathbb{Z}_q^{n \times m}$ and $\mathbb{x} \in \{0,1\}^m$, $f_A(\mathbb{x}) = A\mathbb{x} \pmod{q} \in \mathbb{Z}_q^{n \times 1}$. The main security parameter here is n : the number of rows in the matrix A. This defines the difficulty of inverting a matrix which implicitly means that we can make m as large as we want. The function $f_A(x)$ maps m bits of vector $\mathbb{x}$ to $n \log_2 q$

bits since input $\mathbb{x}$ is binary but output is integer $\pmod q$. Hence, we introduce $log_2 q$ with n. When $m > n\log_2 q$, $f_A(\mathbb{x})$ maps larger domain of inputs to smaller $n\log_2 q$ output. Thus, $f_A(\mathbb{x})$ acts as a compression function. Equation 2.16 shows an example of such compression. Here we can see that for $m = 2n\log_2 q$, m-bit input is compressed into $\frac{m}{2}$-bit output by compression function $f_A(\mathbb{x})$.

$$\begin{gathered}\text{Let, } m = 2n\log_2 q \\ \text{Thus, } f_A : \{0,1\}^m \to \{0,1\}^{n\log_2 q} \\ \implies f_A : \{0,1\}^m \to \{0,1\}^{\frac{m}{2}}\end{gathered} \tag{2.16}$$

Collision Resistant Hash Function. A collision resistant hash function compresses the input from set $\mathbb{X}$ to some output set $\mathbb{Y}$, where $||\mathbb{X}|| > ||\mathbb{Y}||$. Since, the size of input set $\mathbb{X}$ is less than that of output set $\mathbb{Y}$, there are collisions generated by such functions. However, finding these collisions are computationally hard problems. As for *Short Integer Solution* problem, the reasons behind SIS produces Collision Resistant Hash Function is described below.

- **One-way Function.** Given a SIS function $f_A : \{0,1\}^m \to \mathbb{Z}_q^n$, given a output vector $\mathbb{t} \in \mathbb{Z}_q^n$, finding the input short vector $\mathbb{x} \in \{0,1\}^m$ is computationally hard. This makes function f_A a *One-way* function.

- **Regularity.** A function $f : \mathbb{X} \to \mathbb{Y}$ is regular if all $y \in \mathbb{Y}$ have same $|f^{-1}(y)|$. That is, if $f(x \in \mathbb{X}) = y \in \mathbb{Y}$, then the inverse function $f^{-1}(y \in \mathbb{Y})$ will produce $|x \in \mathbb{X}|$ as output. In other words, A function is regular, if uniform input distribution to the function generates uniform output distribution.

  Now if we fix two short vectors $\mathbb{x}_1$ and $\mathbb{x}_2$ for our SIS function while randomly select the key A_1 and A_2, the outputs of $f_{A_1}(\mathbb{x}_1)$ and $f_{A_2}(\mathbb{x}_2)$ are pairwise independent. Moreover, since SIS functions map larger domain of inputs to smaller outputs, both of t $f_{A_1}(\mathbb{x}_1)$ and $f_{A_2}(\mathbb{x}_2)$ are pairwise independent compression functions. From *Leftover Hash Lemma* [24], we know that a compression function is regular if it is pairwise independent. This makes SIS a regular function.

From the discussion above, we understand that SIS is One-way and Regular. This makes the outputs from the SIS function to be distributed randomly and the output cannot be traced back to the input. Thus, even if collision happens they are random and computationally hard to be reproduced. Thus, *Short Integer Solution* creates Collision Resistant Hash function. Moreover, we can see that the randomness of the hash function depends on the size of output set. As a result, the security of the hash function $f_A : \{0,1\}^m \to \mathbb{Z}_q^n$ depends on the size of n whereas m can be increased as much as needed.

Perfectly Hiding Commitment. SIS functions can be used to perfectly hide commitment. The **analogy** behind such cryptographic tool is the following:

- Lock message in a box

- Give the locked box to a receiver but keep the key
- Give the key at a later time.

For example, Person A may have found an algorithm to predict the stock market with absolute precision. However, to prove such result to Person B, Person A may want to expose the prediction to Person B before the stock market opens for a particular date. However, to disclose such result to Person B may give Person B the leverage to invest in stock market and thus creating an uneven playing field for the other investors. Person A, thus, locks his prediction in a box and sends it to Person B. Once the actual status of the stock market is out, Person A can give Person B the key to the box. Person B can open the box and check the results of Person A's prediction by cross-referencing it with the actual stock market data. Such process dose not provide any knowledge to Person B yet proves a point by Person A. Such cryptographic tool is known as *Zero Knowledge Proof.* In order to execute such action, one may need to perfectly hide the message until a desired time come when the message cannot be changed, rather the message is decrypted. The **implementation** of such process can be described as the following.

- **Randomized Function.** Let msg be the message. A randomizer r is implemented on the message to create a commitment. So the randomized function becomes $c = Commit(msg, r)$
- $Commit(msg, r)$**.** The randomized message c is given to the verifier who will cross-check the message m once she receives the key to the committed randomized message.
- **Open.** When the right time comes, the sender reveals both msg and r such that the verifier can calculate $Commit(msg, r) = c'$ and checks that the previously sent commitment c is in fact same as c'.

Now that we understand the process of Zero Knowledge Proof, we should look at the reasons why such process is called *Commitment.*

1. The message sender cannot change her message and get the same cipher text. Let $c = Commit(msg, r)$ is given to the verifier. After a while, the sender changes her mind and tries to find another randomizer r' such that she would generate the same cipher text $c = Commit(msg', r')$. However, $Commit(msg, r) = Commit(msg', r')$ is hard to find.

   Thus, once the sender commits to a message it is computationally hard for her to prove that she sent a different message.

2. As $Commit(msg, r)$ is randomized, it is hard for the verifier to guess msg from $c = Commit(msg, r)$ without any knowledge of r.

Thus, from the above discussion , we can conclude that perfectly hiding commitments need to have the following properties:

1. **Hiding:** $c = Commit(msg, r)$ is independent of msg . r randomizes the function.
2. **Binding:** For $msg \neq msg'$, $Commit(msg, r) = Commit(msg', r')$ is hard to find.

Now that we understand the basic mechanism of *Perfectly Hiding Commitment*, let us look at how we can implement such protocol using lattices.

- Choose $A_1, A_2 \in \mathbb{Z}_q^{n\times m}$ where A_1, A_2 are random.
- Message $msg \in \{0,1\}^m$ and randomness $r \in \{0,1\}^m$.
- Calculate commitment:

$$\begin{aligned} Commit(msg, r) &= f_{[A_1,A_2]}(msg, r) \\ &= A_1.msg + A_2.r \\ &= c \end{aligned} \tag{2.17}$$

- Send cipher text c to verifier. Both Sender and Verifier know about A_1 and A_2.
- Later, reveal msg, r to verifier to cross-check the message.

Finally, let us look at the properties of the commitment based on SIS.

1. **Hiding Property:** It can be noticed that $A_2.r = f_{A_2}(r)$ which is uniform at random over $\mathbb{Z}_q^n$. When we add $A_2.r$ to $A_1.msg$, we get a completely random value.
2. **Binding Property:** Finding $Commit(msg, r) = Commit(msg', r')$ where $(msg, r) \neq (msg', r')$ gives a collision in $f_{[A_1,A_2]}$. But as we discussed earlier, such collision is hard to find in SIS.

Digital Signature. To understand the procedure to produce digital signature using Short Integer Solution, we need to at first look at the homomorphism property of SIS. A function $f : \mathbb{X} \to \mathbb{Y}$ is called homomorphic if $f(x_1 * x_2) = f(x_1) * f(x_2)$ where $x_1, x_2 \in \mathbb{X}$ and $*$ is an arbitrary binary operation. SIS shows the following homoporphic properties.

1. Let, $A \in \mathbb{Z}_q^{n\times m}$ and $\mathbb{x}_1, \mathbb{x}_2 \in \{0,1\}^m$ are two short vectors. Consequently, $(\mathbb{x}_1 + \mathbb{x}_2)$ is also short. Thus, $f_A(\mathbb{x}_1 + \mathbb{x}_2) \approx f_A(\mathbb{x}_1) + f_A(\mathbb{x}_2)$ is approximately homomorphic.
2. SIS is also key-homomorphic. For example, let, $A_1, A_2 \in \mathbb{Z}_q^{n\times m}$ and $x \in \{0,1\}^m$ where A_1, A_2 are the keys.

$$\begin{aligned} f_{[A_1+A_2]}(\mathbb{x}) &= f_{A_1}(\mathbb{x}) + f_{A_2}(\mathbb{x}) \\ &= A_1.\mathbb{x} + A_2.\mathbb{x} \end{aligned} \tag{2.18}$$

Now let us look at a common digital signature scheme.

- Using a Key Generation algorithm, generate a public key, secret key pair (P_k, S_k).
- Sign a message msg with the secret key S_k to generate the signature σ.

$$Sign(S_k, msg) = \sigma$$

- Verify the signature using Public Key P_k, message msg and signature σ.

Finally, let us look at the implementation details of digital signatures using *Short Integer Solution*. Let $\mathbb{X} = [\mathbb{x}_1, \mathbb{x}_2, ..., \mathbb{x}_\ell] \in \{0,1\}^{m\times\ell}$. Thus,

$$\begin{aligned} f_A(\mathbb{X}) &= [f_A(\mathbb{x}_1), f_A(\mathbb{x}_2), ..., f_A(\mathbb{x}_\ell)] \\ &= [A\mathbb{x}_1, A\mathbb{x}_2, ..., A\mathbb{x}_\ell] \\ &= AX \pmod{q} \in \mathbb{Z}_q^{n\times\ell} \end{aligned} \tag{2.19}$$

Now, let us discuss the scheme using SIS.

- **Key Generation.** The key to SIS, A is the publicly available parameter. The secret key, $S_k = (\mathbb{X}, \mathbb{x})$ where $\mathbb{X} \in \{0,1\}^{m\times\ell}$ and $\mathbb{x} \in \{0,1\}^m$. The public key, $P_k = (\mathbb{Y} = f_A(\mathbb{X}), \mathbb{y} = f_A(\mathbb{x}))$. The public key is the image of S_k under f_A.
- **Message.** The message is a short vector $msg \in \{0,1\}^\ell$.
- **Sign Message.** The message is signed with S_k and msg.

$$sign(S_k, msg) = (\mathbb{X}.msg + \mathbb{x}) = \sigma$$

- **Verify.** Once the receiver receives the message and signature, she can verify the signature with the message, signature and public key of the message sender.

$$\begin{aligned} f_A(\sigma) &= f_A(\mathbb{X}.msg + \mathbb{x}) \\ &= f_A(\mathbb{X}.msg) + f_A(\mathbb{x}) \\ &= A\mathbb{X}.msg + A\mathbb{x} \\ &= \mathbb{Y}.msg + \mathbb{y} \end{aligned} \tag{2.20}$$

  As we can see from equation 2.20, $\mathbb{Y}.msg + \mathbb{y}$ can be calculated from the received message and public key, P_k.

Learning with Errors Problem

Learning with errors (LWE) is a problem in machine learning that is conjectured to be hard to solve. Introduced by Oded Regev in 2005 [48], it is a generalization of the parity learning problem. Regev showed, furthermore, that the LWE problem is as hard to solve as several worst-case lattice problems. The LWE problem has recently[44] been used as a hardness assumption to create public-key cryptosystems, such as the ring learning with errors key exchange by Peikert. In this section the will demonstrate how to design a public key cryptosystem from LWE.

Public Key Cryptography using LWE

As opposed to what we have understood from SIS problem where the input to the SIS function is compressed, LWE expands the input and perturbs the input with error. This error perturbation is used to generate the keys for public key encryption.

Let $m, n, q \in \mathbb{Z}$, a matrix $A \in \mathbb{Z}_q^{n\times m}$, a short vector $\mathbb{s} \in \mathbb{Z}_q^n$ and an m-dimensional error vector $e \in \alpha$ where $\alpha << 1 \approx 0.01$ is the error rate of Gaussian error distribution and $\alpha.q > \sqrt{n}$. Given $(A, (\mathbb{s}^T.A+e^T))$, LWE asks to find a short vector $\mathbb{s}$. For example,

$$\text{Let, } A = \begin{bmatrix} 1 & 2 & 1 & 9 \\ 3 & 7 & 2 & 2 \\ 9 & 5 & 4 & 3 \end{bmatrix}; \quad \mathbb{s} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} \text{ and, } \quad e = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix}$$

$$\begin{aligned} \text{Now, } \mathbb{b} &= \mathbb{s}^T.A \\ &= \begin{bmatrix} 1 & 3 & 2 \end{bmatrix} . \begin{bmatrix} 1 & 2 & 1 & 9 \\ 3 & 7 & 2 & 2 \\ 9 & 5 & 4 & 3 \end{bmatrix} \\ &= \begin{bmatrix} 28 & 33 & 15 & 21 \end{bmatrix} \end{aligned} \tag{2.21}$$

$$\begin{aligned} \text{Thus, } \mathbb{b}' &= \mathbb{b} + e^T \\ &= \begin{bmatrix} 28 & 33 & 15 & 21 \end{bmatrix} + \begin{bmatrix} e_1 & e_2 & e_3 & e_4 \end{bmatrix} \end{aligned}$$

Public Key Scheme. Let Bob wants to send a message-bit to Alice. The procedure to use LWE based public key encryption scheme is described below.

1. Let, matrix $A \in \mathbb{Z}_q^{n\times m}$ is publicly available for both Alice and Bob.
2. Alice's secret key is a vector, $\mathbb{s} \in \mathbb{Z}_q^n$. The public key of alice is $\mathbb{b}' = \mathbb{s}^T.A + e^T$. Alice sends her public key to Bob.
3. Bob generates a m-bit vector $\mathbb{x} \in \{0,1\}^m$. This vector $\mathbb{x}$ is used to generate *Cipher Text Preamble* $\mathbb{u} = A.\mathbb{x}$ and sends it to Alice.
4. Let, the message bit that Bob wants to send to Alice in $msg_{bit} \in \{0,1\}$. Bob computes $u' = \mathbb{b}'.\mathbb{x} + msg_{bit}.\frac{q}{2}$ and sends it to Alice.
5. Alice receives both $\mathbb{u}$ and u' and computes $u' - \mathbb{s}^T.\mathbb{u} \approx msg_{bit}.\frac{q}{2}$. For a large enough prime q, Alice gets a value that is very close to $\frac{q}{2}$ or something that is very far from $\frac{q}{2}$.

The proof of accuracy of this public key scheme is given in Appendix D.

2.2 Artificial Neural Network

In this section, we constrain our focus on *Feed Forward Artificial Neural Network*. A feedforward neural network is an artificial neural network wherein connections between the units do not form a cycle. As such, it is different from recurrent neural networks.

The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

2.2.1 Single-Layer Perceptron

The simplest kind of neural network is a single-layer perceptron network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. In this way it can be considered the simplest kind of feed-forward network. The sum of the products of the weights and the inputs is calculated in each node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1). Neurons with this kind of activation function are also called artificial neurons or linear threshold units. In the literature the term perceptron often refers to networks consisting of just one of these units. A similar neuron was described by Warren McCulloch and Walter Pitts in the 1940s.

A perceptron can be created using any values for the activated and deactivated states as long as the threshold value lies between the two.

Perceptrons can be trained by a simple learning algorithm that is usually called the delta rule. It calculates the errors between calculated output and sample output data, and uses this to create an adjustment to the weights, thus implementing a form of gradient descent.

Single-unit perceptrons are only capable of learning linearly separable patterns; in 1969 in a famous monograph entitled Perceptrons, Marvin Minsky and Seymour Papert showed that it was impossible for a single-layer perceptron network to learn an XOR function (nonetheless, it was known that multi-layer perceptrons are capable of producing any possible boolean function).

Although a single threshold unit is quite limited in its computational power, it has been shown that networks of parallel threshold units can approximate any continuous function from a compact interval of the real numbers into the interval [-1,1]. This result can be found in Peter Auer, Harald Burgsteiner and Wolfgang Maass "A learning rule for very simple universal approximators consisting of a single layer of perceptrons".

A multi-layer neural network can compute a continuous output instead of a step function. A common choice is the so-called logistic function:

$$f(x) = \frac{1}{1 + e^{-x}}$$

With this choice, the single-layer network is identical to the logistic regression model, widely used in statistical modeling. The logistic function is also known as the sigmoid function. It has a continuous derivative, which allows it to be used in backpropagation. This function is also preferred because its derivative is easily calculated:

$$f'(x) = f(x)(1 - f(x))$$

2.2.2 Multi-Layer Perceptron

This class of networks consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function.

The universal approximation theorem for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer. This result holds for a wide range of activation functions, e.g. for the sigmoidal functions.

Multi-layer networks use a variety of learning techniques, the most popular being back-propagation. Here, the output values are compared with the correct answer to compute the value of some predefined error-function. By various techniques, the error is then fed back through the network. Using this information, the algorithm adjusts the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error of the calculations is small. In this case, one would say that the network has learned a certain target function. To adjust weights properly, one applies a general method for non-linear optimization that is called gradient descent. For this, the network calculates the derivative of the error function with respect to the network weights, and changes the weights such that the error decreases (thus going downhill on the surface of the error function). For this reason, back-propagation can only be applied on networks with differentiable activation functions.

In general, the problem of teaching a network to perform well, even on samples that were not used as training samples, is a quite subtle issue that requires additional techniques. This is especially important for cases where only very limited numbers of training samples are available. The danger is that the network overfits the training data and fails to capture the true statistical process generating the data. Computational learning theory is concerned with training classifiers on a limited amount of data. In the context of neural networks a simple heuristic, called early stopping, often ensures that the network will generalize well to examples not in the training set.

Other typical problems of the back-propagation algorithm are the speed of convergence and the possibility of ending up in a local minimum of the error function. Today there are practical methods that make back-propagation in multi-layer perceptrons the tool of choice for many machine learning tasks.

Chapter 3

Related Works

As concerns regarding the security of IoT devices have been rising sharply throughout the last decade, scholarly articles tackling such challenges and proposing countermeasures are increasing exponentially [32]. However, these security solutions generally concern themselves within any of these three major layers of IoT architecture - Edge-side layer, Cloud/Server-side layer or User-side layer. Moreover, the current trend to hook every device with the internet has resulted in numerous heterogeneous IoT devices from Edge-side layer working together for billions of use-cases and smart systems with the help of Cloud/Server-side layer along with providing services to the User-side layer. Before we go into details of our proposed method to use the heterogeneity of these IoT devices and vast spectrum of their use-cases to come up a solution for secured communication among different layers of IoT architecture, a brief discussion on the current security solutions for these interacting layers are presented in this section.

3.1 Edge-side Layer

According to [37], at the computing nodes of Edge-side layer, the *edge computing nodes* are vulnerable to Hardware Trojans for integrated circuits, Non-network side channel attack, Denial-of-Service attacks, Node Replication Attack, Camouflage etc. Attacks against *RFID*-tags include tracking, inventorying, tag cloning, Eavesdropping, Side-Channel attack, etc. Side-Channel Analysis, Policy-based mechanisms and intrusion detection systems (IDSs), Circuit modification are few viable countermeasures to tackle the security risks in the computing nodes.

Hardware Trojans, as described in [53, 56], are modified integrated circuits which are placed with malicious intents during the fabrication of devices. In [41], it is proposed to use *Side Channel Analysis* of temperature and power consumption of IoT devices to detect Hardware Trojans. Even malicious firmware or software can be detected by without intruding devices using Side Channel Analysis as proposed in *WattsUpDoc* [14]. Unfortunately, this very approach can be used to detect the behavior of a device. For example, in [38], researchers have demonstrated how Electromagnetic and acoustic signals from medical devices can leak information about patients. More interestingly, in [11], it is demonstrated that how acoustic noise from cyber-physical systems can leak information about the manufactured product. The authors here used acoustic noise and vibration from 3D printers to steal geometric information about the products that are being created.

Denial of Service attacks on edge nodes can affect both the edge computing devices (i.e., sleep deprivation in [55], battery draining in [8], etc.) and communication [40] among them. Some IoT devices are also prone to attack from Malware. To tackle these problems, many works have proposed protocols to protect devices autonomously while others have worked on integrating security measures in a distributed fashion. In [19], researchers have proposed a lightweight cryptographic protocol to authenticate RFID cards. [10] describes a model based security mechanism for computing devices that preemptively predicts and reacts to cyber attacks without any human interventions. On the other hand, works like [49] uses artificial intelligence to calculate trust in a *Vehicular Area Network (VAN)* by collaborating with other vehicles in a particular location. Additionally, collaborative effort to ensure security is also discussed in [18] where, based on computation and power level of IoT devices, the devices work together to ensure certain level of security.

IoT devices with constrained resources can sometimes be devoid of a memory management unit, protection rings, hypervisors, or other security mechanisms. An active attacker can take control of software modules and even the operating system of such vulnerable devices. Moreover, the attackers can take control of the communication networks connecting these devices. In [39], authors propose an architecture to deliver software securely to the edge nodes with limited resources along with confining vulnerabilities in software or hardware with their own modules.

3.2 Cloud/Server-side Layer

No matter how secured the devices producing data and the communication channel sending data to the cloud may be, all efforts can be undone by malicious entity in an untrusted cloud. Groundbreaking work by Dan Boneh, Craig Gentry et al. in [5], has introduced an encryption scheme - *Somewhat Homomorphic Encryption* - that enables secured query execution over encrypted data in the cloud. Later Craig Gentry went on to propose a Fully Homomorphic Encryption [21] that led the way to secured computation over cloud. Results from these two schemes have been applied in searching over encrypted data [45], classifying data using various machine learning process in [6], confidential query processing [46], etc. Moreover, Brandenburger et al. in [7] proposes a protocol for verification of integrity and consistency *VICOS* for cloud where the trusted clients can check the integrity of their shared data on the cloud. Given a *Byzantine* server, *VIOCS* detects irregularities in data based on the operations performed by the clients. This, however, relies on the secured communication and confidentiality between clients and server.

3.3 User-side Layer

Protecting User Identity and access pattern along with confidentiality of the data over the cloud are the major security challenges in the IoT architecture. In [47], researchers present a list of cryptographic libraries to protect the user-data in an untrusted cloud. However the work did not concern itself with the user identity and access patterns in the cloud. One solution was provided in [2] that implements forward secrecy to hide user identity and utilizes bloom filter to cloak user-access patterns. In terms of integrity of information, [26] proposes an end-to-end integrity protection web platform to ensure users with unadulterated data from the cloud.

Chapter 4

Proposed Protocol

In this chapter, we propose a protocol to ensure a public key mechanism where the communicating parties need to behave certain way or be part of the same environment. The work-flow of this protocol is presented in section 4.5. But at first let us go through some definitions and notations used in this work.

4.1 Definitions

Following are the definitions that are necessary to describe our proposed protocol.

4.1.1 Communicating Parties

Our protocol intends to facilitate two different parties to communicate using public key encryption method. However, to corroborate the strength of the encryption, we proposed interim methods that requires both parties to communicate at the start of a communication session. Thus, in this work, to determine the parties in communication, we denoted the party who ultimately wants to send message bits as *Message-Sending Party.* On the other hand, the one who finally expects to receive the message-bits is denoted as *Message-Receiving Party.*

4.1.2 Environment Variables

Environment Variables can be anything that both communicating parties have access to and are difficult to replicate. This environment can be a face, a finger print, sensor array in a smart car - any pattern-generating behavior that has multiple features or variables in it and allows multiple observations of those features. Being part of the same environment translates into having access to the variables of the environment which in turn means that observations can be made of those variables with some degree of freedom.

In our work here, we assumed, the parties that want to communicate, are part of the same environment or behave in similar ways and thus, they have access to same environment variables. The observations made by different parties of these variables, should be similar enough to consider the Message-Sending and Message-Receiving party as part of the same environment. When we demonstrate our work in chapter 5, we assume that multiple weather stations may have access to same SensorScope sensor network data and can use these sensor data to generate session keys. Thus, the sensor data from the stations located in the same location makes the stations to be part of the same environment, whereas stations from different locations belong to different environments. As a result, the environment variables from the same environment, in this case, sensor data from same location, will be similar with minimal noise. On the contrary, SensorScope sensor networks in different locations will generate data so dissimilar that the weather stations in different locations will be considered as if they belonged to different environments.

4.1.3 Intermediate Keys

Our main goal in the proposed protocol is to ensure that the environment variable data are part of the public key. So as to achieve that we introduced Intermediate Keys which would be generated by both communicating parties individually from the environment variable data. The Intermediate keys are matrices, generated from environment variable data, are the main ingredients to generate session keys later in the protocol. After environment variable data is acquired, both parties who want to communicate, take the moving average of observations from each variables and come up with the intermediate key. The window of the moving average is fixed for both the parties and thus, the number of rows in the intermediate keys are less than the observations of the environment variables. However, the number of rows in the intermediate keys of both parties remains the same because of the same window length. Moreover, the intermediate keys' number of columns also remains the same as the number of variables in the environment. The environment variable data are modified by taking their moving average in order to smooth out the outliers and spikes in the observed data and generate similar intermediate keys for both parties

4.1.4 Session Keys

Session keys are generated after both communicating parties have already acquired Environment Variables and calculated their own Intermediate Keys. For initiating communication, a session demands a session key which is generated by the collaboration of both parties. However, what will determine a new session and how long a session will exist completely depends on the implementation. In this work, we demonstrated each session as a certain length of message bits. When the message-receiving-party receives that certain number of message-bits, a new session is initiated which in turn requires a new session key to be generated.

Generating the Session Key is a three-step process, the message-sending party initiates it and the message-receiving party completes the process. Moreover, we have defined four matrices for the whole process to finish: ***Initial Session Key***, ***Session Key Seed***, ***Final Session Key*** and ***Session Key Index Matrix***. The processes to generate these session matrices are described below:

1. The *Message-Sending* party picks a certain number of unique elements (i.e., prime numbers) and create a matrix wish dimensions same as the *Intermediate Keys*. The unique elements are distributed randomly across the new matrix. As the number of elements in the new matrix is far greater than the number of unique elements, there will definitely be repetitions of the unique elements, but they will be placed at random.

   These unique elements are selected and distributed optimally based on the constraints described in section 4.5.3. We named this newly created matrix as ***Initial Session Key***.

2. In the next step, the Intermediate Key of the Message-Sending party is utilized as the key to One-Time pad for encrypting *Initial Session Key*. We termed this encrypted interim key as ***Session Key Seed***. This *Session Key Seed* is transmitted to the *Message-Receiving party* for decryption and generation of ***Final Session Key***. The number of unique keys is also sent to the *Message-Sending party*.

3. In the last step, the *Message-Receiving Party* receives this *Session Key Seed* and decrypts it to get the *Initial Session Key* using their own *Intermediate Key*. The emphInitial Session Key of the *Message-Receiving party* may be different from the one generated by the *Message-Sending party*. However, the *Initial Session Key*, generated at the *Message-Receiving Party*'s end, needs to be same for *Message-Sending party* too. We bolstered the probability of *Initial Session Key*s of the two parties to be same by randomly picking a small portion of elements from the *Initial Session Keys* and creating *Final Session Key*s.

   Thus, to enhance the chance of the two *Final Session Keys* to be same, *Message-Receiver party* initially selects a list of indices at random from their own *Initial Session Key* matrix. The said party then permute the random indices as many times as necessary to fill a matrix with the same dimensions as the *Initial Session Key*. This ***Session Key Index Matrix*** is then sent to *Message-Sending Party* and both the parties pick the elements pointed by the indices in *Session Key Index Matrix* from their own *Initial Session Keys* and form the *Final Session Key*.

4.1.5 Unique Elements

When *Message-Sending Party* generates her own Initial Session Key matrix, she needs to select the elements from a list of non-zero positive integers. The values in the list are non repeating and the *Message-Sending Party* randomly chooses the elements to form the Initial Session Matrix. In our work, these non-repeating non-zero integer values are called Unique Elements.

4.1.6 Public Keys

The Message-Receiving Party in any public key encryption method requires to have a public key and a corresponding private key. We, too, in our work have incorporated the working mechanism of a generic public key as ***Primary Public Key***. However, in our endeavor to make the Environment Variables as a part of the public key, we proposed a method to use the Final Session Key as the key to generate one-way hash of Primary Public Key using *Short Integer Solution.* Both Message-Receiving-Party and Message-Sending-Party calculate their hash of Primary Public Key using their own Final Session Key. In this literature, we call the hashed Primary Public Key as ***Session Public Key***. The Primary Public Key can remain the same throughout the lifetime of a Message-Receiving Party while the Session Public Key needs to be generated at the start of each session by both the Message-Sending and Message-Receiving parties. However, the private key of the Message-Receiving party can remain same as the Primary Public Key or it can be changed at the start of each session

4.2 Notations

Throughout this work, we have assumed the *Message-Receiving Party* - one who expects to receive message as $Party_A$, whereas the *Message-Sending Party* is denoted as $Party_B$.

Uppercase letters and Uppercase Double Struck letters have been used to denote Matrices and Sets respectively. For example, K, K_1, K_2 are matrices whereas $\mathbb{X}, \mathbb{Y}$ are sets of elements. It should also be noted that terms written in Camel Case (i.e., *EnvData*) are also considered to be Matrices. Again, scalar values and vectors have been represented using lowercase letters (i.e., m, n) and double struck lowercase letters, (i.e., $\mathbb{x}, \mathbb{y}$), respectively. We have represented the dimensions of a matrix, K, as $m \times n$, which means that matrix K has m number of rows and n number of columns. The transpose of a matrix is denoted by an uppercase 'T' as a super-script, i.e., K_1^T. Moreover, an element in matrix has been described as $K(x, y)$, where, $(x, y) \in \{\mathbb{N} - 0\}$ and points to the element at row x and column y. Additionally, when we describe two sets, i.e., $\mathbb{X} = \{1, 2, ..., m\}$ and $\mathbb{Y} = \{1, 2, ..., n\}$, the Cartesian Product of $\mathbb{X}$ and $\mathbb{Y}$ can be described by $\mathbb{X} \times \mathbb{Y} = \{(x, y) \mid x \in \mathbb{X} \land y \in \mathbb{Y}\}$. On the other hand, if K_1 and K_2 are two matrices with dimensions $m_1 \times n$ and $n \times m_2$ respectively, then the Matrix Product of these two matrices has been denoted as $K_1 \times K_2$ which would have the dimension of $m_1 \times m_2$. However, scalar-scalar multiplication and scalar-vector multiplications are denotes as $m * n$ and $m * \mathbb{x}$ respectively, where $m\ and\ n$ are scalars and $\mathbb{x}$ is a vector.

4.3 Assumptions and Threat Models

In this section we focus on the threats that our work aims to neutralize. At first we discuss how our protocol implements forward secrecy in case of leaked *Long-term*

Secrets. Later we concentrate on an approach to tackle *Distributed Denial-of-Service* attack using IoT devices by implementing our proposed protocol.

4.3.1 Leaked Secret Key

- **Threat:** Using a secret key for long time to decrypt messages in a public key encryption system can give the secret away to a passive attacker. The problem is more crucial when the long-term secret is easy to guess and this can be the case for IoT devices, since it may not always be easy for numerous internet-connected devices to have separate secret keys and to track all those secrets are more difficult still. We can hardly rely on system administrators to always use separate secrets for every IoT device working under certain environment.

- **Guarantees:** Our protocol ensures forward secrecy that not only relies on the long-term secret keys but also utilizes the behavioral patterns and interactions among heterogeneous IoT devices in a system, to generate temporary keys for each session of communication. Thus to decrypt a message sent to a specific environment successfully, the receiver needs to hold the long-term secret key along with the the behavioral patterns of that environment that comprises of heterogeneous IoT devices.

- **Assumptions:** Instead of allowing each device of an environment to receive individual messages, we assumed that the communicating parties have access to an array of common IoT devices and the messages need to be sent from $Party_B$ to $Party_A$ using session keys. $Party_A$ decrypts the messages and can even redirect the decrypted messages to some specific IoT device.

4.3.2 D-DoS Attack

- **Threat:** As more pervasive devices are getting connected to the internet each day, to exploit certain genre of such devices and muster an attack on large-scale digital infrastructure, that may deny services to legitimate clients, are getting more probable. [4, 16] provide us with such an attack and unfortunately many more are to follow unless some constructive steps are taken. One obstacle in avoiding such *Distributed Denial-of-Service* attack by IoT-enabled devices is the sheer number of similar devices with security flaws. Though many security protocols aim to prevent D-DoS attacks, initiated by infected IoT devices [15, 16], hardly such solutions incorporates the heterogeneity and collaboration among such devices. As a result, such solutions again revolve around specific devices with specific characteristics.

- **Guarantees:** In our solution to verify group membership, we propose to include the behavior of IoT devices from wide spectrum of categories, that work under certain environment and show specific characteristics, in order to generate a session key between a *message-sender* and a *message-receiver*. To achieve a common session key is the proof of both communicating parties being part of the same group. Hence, a legitimate communicating party, upon failing repeatedly to generate a common session key with the other one, would be encouraged to block the other party from communicating, though such blocking process and when a legitimate party should do so are beyond the scope of this work.

- **Assumptions:** We, in our proposal, assumed that the characteristics of the pervasive devices in a certain environment are unique enough to be *synthetically* generated by an attacker. Thus, the parties communicating can only generate the session key after having observed the environment variables.

4.4 Protocol Overview

We restrict our discussion in this section to focus on how session keys are generated and used to encrypt messages and decrypt them. As $Party_B$ wants to send message-bits to $Party_A$, $Party_B$ is the *Message-Sending* party, whereas $Party_A$ is the *Message-Receiving* one. Figure 4.1 shows how $Party_A$ and $Party_B$ comes up with the same session keys by acquiring data from their observations of the environment variables.

As a new session starts, both $Party_A$ and $Party_B$ observes their corresponding environment and comes up with Intermediate Keys K_{itm_A} and K_{itm_B} respectively. While $Party_A$ waits for Session Key Seed, K_{SS} to arrive from $Party_B$, $Party_B$ comes up with an Initial Session Key, K_{SI_B}, for itself along with the number of unique elements, el_{uniq} and a window size for calculating moving average, w_{OTP}. K_{itm_B} is used as the key for the One-Time pad to encrypt K_{SI_B} and this encrypted key is called Session Key Seed K_{SS}. The process to encrypt K_{SI_B} to generate K_{SS} is described thoroughly in section 4.5.3.

Upon receiving K_{SS}, el_{uniq} and w_{OTP}, $Party_A$ decrypts K_{SS} using her own Intermediate Key K_{itm_A}. As K_{itm_A} and K_{itm_B} are supposed to be similar because of being generated by observations from same environment variables, decryption of K_{SS} is to relate closely to K_{SI_B}, which is called K_{SI_A}. However, the Final Session Key, K_{SF} need to be exactly same for both communicating parties, $Party_A$ randomly selects elements from K_{SI_A} and permutes the selected elements randomly to fill the Session Key Index Matrix Idx_{SF}. This Idx_{SF} is sent to $Party_B$ and she, too, come up with the same K_{SF} by selecting the elements from K_{itm_B} that are pointed to by the indexes in Idx_{SF}. The detailed description of decrypting the Session Key Seed and generating the Final Session Key are given in 4.5.3 and 4.5.3 respectively.

Later, both $Party_A$ and $Party_B$ generate their own Session Public Key, K_{SP} by using Short Integer Solution to come up with the hashed values of $Party_A$'s public key, K_{pub_A}. $Party_B$ encrypts the message bits, msg_{bit}s, using LWE based Public Key Encryption, where K_{SP} is considered to be the public key since both $Party_A$ and $Party_B$ holds the common key. Once $Party_A$ receives the msg_{bit}s, she decrypts the messages using her long-term private key K_{pub_A}. For each bit of message, the decrypted value will be close to $\frac{p_{lg}}{2}$ or a value very far from $\frac{p_{lg}}{2}$, where p_{lg} is a large prime number which is common for both $Party_A$ and $Party_B$.

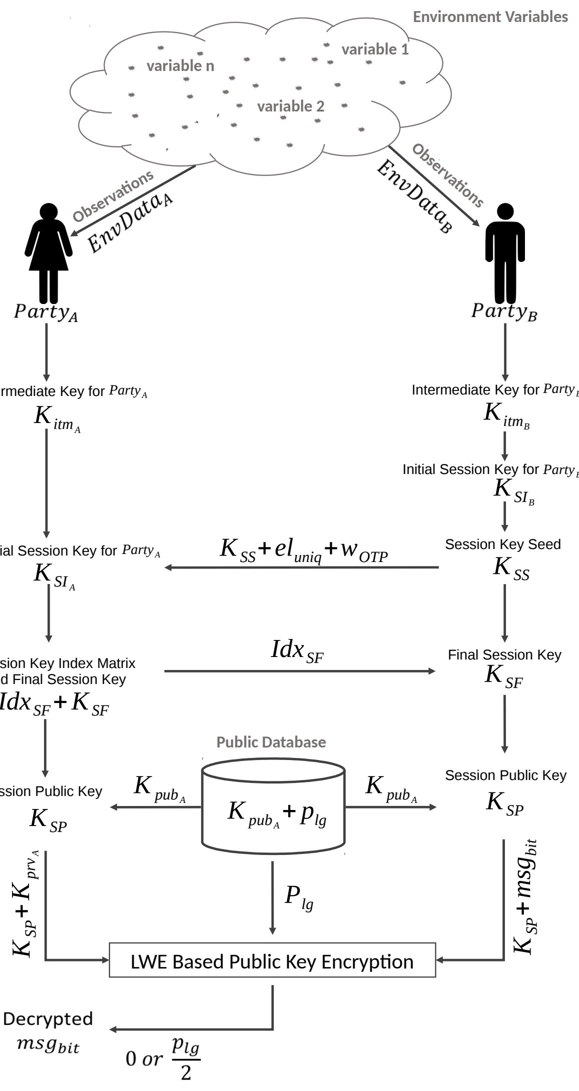


Figure 4.1: Overview of the proposed protocol

4.5 Protocol Details

Herein, we discuss the detailed steps of our proposed procedure - from generating the Intermediate Keys and Session Keys to encrypting and decrypting messages between two parties. In this section, we reflect on how these proposed steps enables us overcome several security threats along with our assumptions behind them as described in 4.3. The steps of our proposed protocol is shown in figure 4.2.

4.5.1 Acquire Environment Variable Data

The first step of our protocol is to acquire environment variable data which requires all the parties in communication to be part of the same environment. Let us first consider

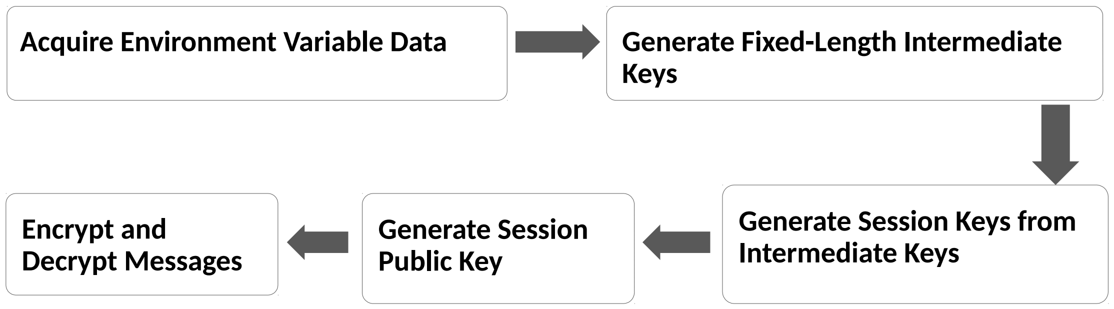

Acquire Environment Variable Data
Generate Fixed-Length Intermediate Keys
Generate Session Keys from Intermediate Keys
Generate Session Public Key
Encrypt and Decrypt Messages

Figure 4.2: Steps in proposed protocol

that both $Party_A$ and $Party_B$ are part of the same environment. As they take their own observations and/or predictions of the environment variables separately, let us assume that there are n_{env} number of variables or features in the environment and both the parties take m_{env} observations. As there may be some noise associated with the observation of the environment variables, we assume that $Party_A$ and $Party_B$ come up with matrices $EnvData_A$ and $EnvData_B$ respectively, each of which has a dimension of $m_{env} \times n_{env}$.

4.5.2 Generate Fixed Length Intermediate Keys

The next step in our protocol is to generate fixed length Intermediate Keys from the observed and/or predicted Environment variables. Both $Party_A$ and $Party_B$ come up with Intermediate Keys, K_{itm_A} and K_{itm_B} respectively. The length of intermediate keys - the number of rows - should be less than or equal to that of the observations of environment variables, though they must be same for both communicating parties. If the length of both Intermediate keys is m_{itm} , then the dimension of both Intermediate Keys will be $m_{itm} \times n_{env}$, where $m_{env} \geq m_{itm}$. In our protocol, we took the moving average of the observations to reduce the length of the intermediate keys. So, each of the parties dose the following to calculate the moving average of their observations and generate Intermediate keys:

1. **Compute the window size for moving average:** To compute the Intermediate keys, we need to, at first, calculate the size of the moving average window. The mean of each window will, later, be the value of an element in the Intermediate keys. Let, ws be the window size for both the communicating parties, where
$$ws = m_{env} - m_{itm} \tag{4.1}$$

2. **Compute mean value of observations for each window:** Let us assume that the Intermediate Key for a communicating party is K_{itm} which has the dimensions of $m_{itm} \times n_{env}$. We calculate each element in K_{itm} from the observations, $EnvData$ with dimensions $m_{env} \times n_{env}$, of the environment made the communicating party from equation 4.2.

$$
\begin{aligned}
&Let, \mathbb{X} = \{1, 2, ..., m_{itm}\}, \mathbb{Y} = \{1, 2, ..., n_{env}\} \\
&\forall(x, y) \in (\mathbb{X} \times \mathbb{Y}), K_{itm}(x, y) = \frac{\sum_{i=x}^{x+ws} EnvData(i, y)}{ws}
\end{aligned}
\tag{4.2}
$$

Computational Complexity: From equation 4.2, we can see that the creation of Intermediate key, K_{itm} matrix takes $|\mathbb{X}| \times |\mathbb{Y}| \times ws$ steps. Since $ws << |\mathbb{X}| \times |\mathbb{Y}|$, the complexity of creating the intermediate keys using moving average is $\mathcal{O}(|\mathbb{X}| \times |\mathbb{Y}|)$.

4.5.3 Generate Session Keys from Intermediate Keys

Once $Party_A$ and $Party_B$ generates their own Intermediate Keys, $Party_B$ generates her Initial Session Key and encrypt it using One-Time pads with her Intermediate Key as the encryption key. This encrypted Initial Session Key - Session Key Seed - is sent to $Party_A$, who decrypt it using her Intermediate Key. Later $Party_A$ generates Final Session Key and send the Session Key Index $Party_B$. The process that leads to generating the Final Session Key is described in the following sections.

Initial Session Key

As $Party_B$ uses One-Time pads to encrypt Initial Session Key, the dimensions of her Initial Session Key should be equal to the key- her Intermediate Key. This conversely implies that $Party_B$ needs to come with a Key of size $m_{itm} \times n_{env}$ to encrypt K_{itm_B}. Let us assume that the key is K_{SI_B}. Now, as $Party_A$'s goal is to decrypt the Session Key Seed using K_{itm_A} and get as close as she can get to K_{SI_B}, we introduced three constraints.

Constraints:

1. **Number of Unique Elements:** If $Party_B$ selects a fixed number of Unique Elements, el_{uniq}, to generate the K_{SI_B} by placing those el_{uniq} values at random, the chance for $Party_A$ to decrypt and approximate from a list of candidate values increase. Thus, K_{SI_B} will contain only these el_{uniq} values arranged randomly.

2. **Window Size of Repeating Element:** The Intermediate Keys generated by both the communicating parties are exactly same in dimensions, though those keys are generated by the observations made by the parties separately. We assume that there are subtle differences among the observer and/or predicted values. To let $Party_A$ use her Intermediate key to decrypt the Session Key Seed, in spite of such nuances, we need to make sure she can make a calculative guess. Thus, we propose to generate K_{SI_B} such a way where consecutive w_{OTP} number of rows are repeated in K_{SI_B}.

   As we can see in figure 4.3, there are three unique elements, namely el_1, el_2 *and* el_3. The number of times, rows are repeated consecutively, w_{OTP} is two. Thus, every two consecutive rows are the same and when $Party_A$ receives the Session Key Seed, can take the mean of these two rows and will get a matrix with minimal error. The process is described in 4.5.3.

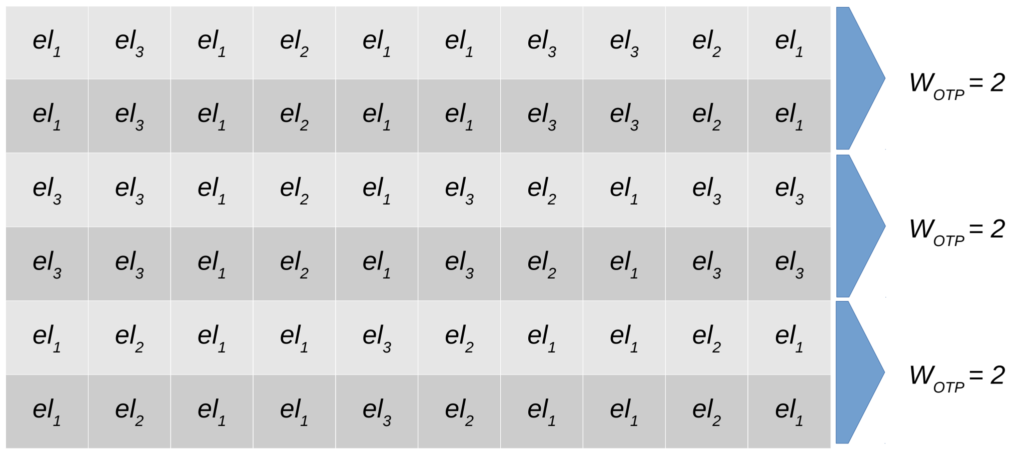

el_1 el_3 el_1 el_2 el_1 el_1 el_3 el_3 el_2 el_1
el_1 el_3 el_1 el_2 el_1 el_1 el_3 el_3 el_2 el_1
$W_{OTP} = 2$
el_3 el_3 el_1 el_2 el_1 el_3 el_2 el_1 el_3 el_3
el_3 el_3 el_1 el_2 el_1 el_3 el_2 el_1 el_3 el_3
$W_{OTP} = 2$
el_1 el_2 el_1 el_1 el_3 el_2 el_1 el_1 el_2 el_1
el_1 el_2 el_1 el_1 el_3 el_2 el_1 el_1 el_2 el_1
$W_{OTP} = 2$

Figure 4.3: Example of a K_{SI_B} to generate Session Key Seed

3. **Range of Elements in K_{SI_B}:** When $Party_A$ tries to decrypt Session Key Seed, the sent key is decrypted using her own Intermediate Key, K_{itm_A} as decryption key. However, the decrypted elements that form K_{SI_A} may deviate from K_{SI_B}. Thus, to allow $Party_A$ to round off K_{SI_A} to the closest unique element, the unique elements should be selected by $Party_B$ sparsely enough that tolerates accurate decryption of K_{SI_A}, even with deviation between K_{itm_A} and K_{itm_B} which is under certain error threshold. Additionally, those unique elements should be selected so closely that, if the deviation crosses a certain error threshold, one unique element would randomly be rounded off as another, on the time of decryption at $Party_A$'s end. Finally, the Unique Elements that are selected should be reduced to the modulus of a large prime number, p_{lg} and the prime should be commonly known by both $Party_A$ and $Party_B$.

   Thus we define two error thresholds for the deviations between K_{itm_A} and K_{itm_B} to let $Party_B$ generate the Unique Elements. **One.** *Average-Case Error Threshold* is the expected average case NRMS error between the Intermediate Keys, represented as th_{avg}. **Two.** *Worst-Case Error Threshold* is the worst tolerable NRMS error between the Intermediate Keys, represented as th_{max}. th_{avg} is used in our work to calculate the sparseness between two consecutive unique elements sorted in ascending order, whereas th_{max} is needed to calculate the maximum value allowed for a unique element. The relation between these two error thresholds and the sparseness of the unique elements is shown in Appendix C.

Encryption of K_{SI_B}: $Party_B$ selects el_{uniq} unique elements and creates a matrix with the dimension of $(m_{itm}/w_{OTP}) \times n_{env}$ using these unique elements randomly, where $el_{uniq} << (m_{itm}/w_{OTP}) * n_{env}$. Each row of this newly created matrix is repeated w_{OTP} times consecutively to create K_{SI_B}. Now, K_{SI_B} is used to element-wise multiply with K_{itm_B} to create K_{SS} following equation 4.3.

$$\begin{aligned}
&Let, \mathbb{X} = \{1, 2, ..., m_{itm}\}, \mathbb{Y} = \{1, 2, ..., n_{env}\} \\
&\forall (x, y) \in (\mathbb{X} \times \mathbb{Y}), \\
&K_{SS}(x, y) = (K_{itm_B}(x, y) * K_{SI_B}(x, y)) \ mod \ p_{lg}
\end{aligned} \tag{4.3}$$

Computational Complexity: From equation 4.3, we can see that the computational complexity is $\mathcal{O}(|\mathbb{X}| \times |\mathbb{Y}|)$, given the fact that generating K_{SS} matrix is a element-wise operation.

Security: This K_{SS} is sent to the other communicating party for decryption after being encrypted using One-Time pads. According to our proposal, each row of K_{SI_B} is repeated w_{OTP} times and no elements along the columns are repeated at all. As we can recall, the rows in K_{itm_B}, which is the key to encrypt K_{SI_B}, are the moving average values of observations made from different environment variables. Our assumption is that the observations made by $Party_A$ and $Party_B$ of the same environment variable may be slightly different and if an attacker wants to get the values of K_{SI_B}, she needs to have access to observations from all environment variables, which is highly unlikely. Thus we can trade off with the security with accuracy by repeating the values of K_{SI_b} along the rows. However, if we repeated the values of K_{SI_b} along the columns, then a compromised environment variable will leak the information of other variables, the ones that are used to encrypt the same K_{SI_B} elements as the compromised one, to an attacker.

Decrypting Session Key Seed

$Party_B$ sends Session Key Seed, K_{SS}, number of Unique Elements, el_{uniq} and number of times each row is repeated in K_{SI_B}, w_{OTP} to $Party_A$. $Party_A$ element-wise divides K_{SS} with its own intermediate key, K_{itm_A} and gets K_{SI_A}. Then the mean of every w_{OTP} rows for each column of K_{SI_A} is calculated and rounded off to the nearest integer value. Later, frequency of each of these integer values are calculated and top el_{uniq} integers with the highest frequencies are considered as the Unique Elements. All other integers are rounded off to the nearest newly calculated Unique Elements and K_{SI_A} is edited accordingly. The detailed procedure is described in algorithm 2.

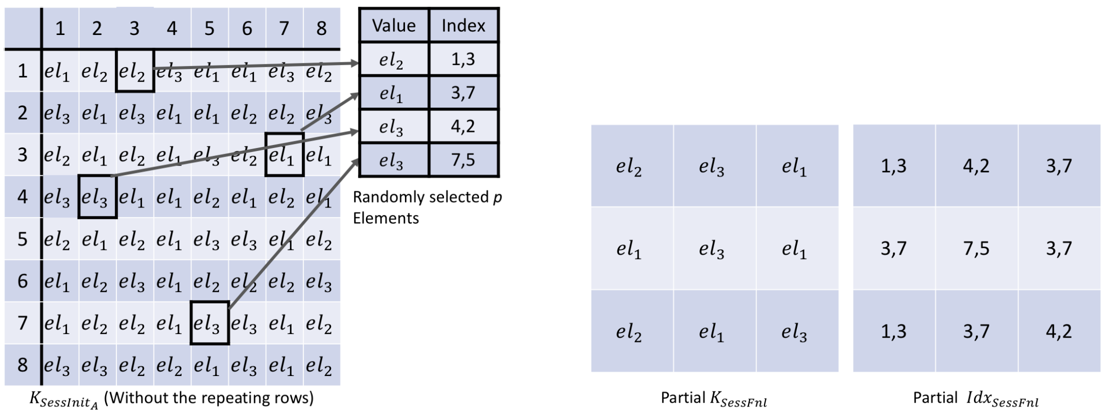
Computational Complexity: From the procedure described in 2, we can see that the computational complexity of the algorithm is $\mathcal{O}(m_{itm} \times n_{env})$

Generating Final Session Key

As it is absolutely imperative that both the communicating parties generate the exact Final Session Key, $Party_A$ selects p elements from K_{SI_A} and repeats these p elements as many times as it takes to fill another matrix with the dimension of $n_{env} \times m_{pk}$, where, $m_{pk} > (n_{env} * \lg(p_{lg}))$. The newly created matrix is permuted randomly to generate K_{SF}. The index values of these elements are tracked while being repeated

Algorithm 2 Decryption of Session Key Seed, K_{SS} to get Initial Session Key, K_{SI_A}

```
Require: $K_{SS}$, $K_{itm_A}$, $el_{uniq}$, $w_{OTP}$, $m_{itm}$, $n_{env}$, $w_{OTP} > 1$
 1: $K_{SI_A} : ARRAY[m_{itm}][n_{env}]$
 2: $sz \leftarrow m_{itm}/w_{OTP}$
 3: $TMP : ARRAY[sz][n_{env}] \leftarrow [0]$ {Initialize with zeros}
 4: $tempc \leftarrow 1$; $c \leftarrow 1$; $\mathbb{X} \leftarrow \emptyset$
 5: for $i = 1$ to $m_{itm}$ do
 6:   for $j = 1$ to $n_{env}$ do
 7:     $TMP[c][j] \leftarrow TMP[c][j] + \lfloor K_{SS}[i][j]/K_{itm_A}[i][j] \rceil$
 8:     if $tempc == w_{OTP} - 1$ then
 9:       $TMP[c][j] \leftarrow \lfloor TMP[c][j]/w_{OTP} \rceil$
10:       $\mathbb{X} \cup \{TMP[c][j]\}$
11:     end if
12:   end for
13:   $tempc \leftarrow tempc + 1$
14:   if $tempc == w_{OTP}$ then
15:     $c \leftarrow c + 1$
16:   end if
17: end for
18: $FREQ : ARRAY[||\mathbb{X}||][3] \leftarrow [x \in \mathbb{X}][0][x \in \mathbb{X}]$ {Initialize 3 columns of $FREQ$}
19: for $i = 1$ to $sz$ do
20:   for $j = 1$ to $n_{env}$ do
21:     $idx \leftarrow INDEX(TMP[i][j]\ in\ FREQ[...][1])$ {Find index of a value}
22:     $FREQ[idx][2] \leftarrow FREQ[idx][2] + 1$
23:   end for
24: end for
25: $SORT(FREQ, 2)$ {Sort $FREQ$'s rows in desc. order by 2nd column values}
26: for $j = 1 + el_{uniq}$ to $||\mathbb{X}||$ do
27:   $minDist \leftarrow \infty$; $minIdx \leftarrow -1$
28:   for $i = 1$ to $el_{uniq}$ do
29:     $dist \leftarrow |FREQ[i][1] - FREQ[j][1]|$
30:     if $dist < minDist$ then
31:       $minDist \leftarrow dist$; $minIdx \leftarrow i$
32:     end if
33:   end for
34:   $FREQ[j][3] \leftarrow FREQ[minIdx][1]$
35: end for
36: for $i = 1$ to $sz$ do
37:   for $j = 1$ to $n_{env}$ do
38:     $idx \leftarrow INDEX(TMP[i][j]\ in\ FREQ[...][1])$
39:     $TMP[i][j] \leftarrow FREQ[idx][3]$
40:   end for
41: end for
42: $K_{SI_A} \leftarrow REPROW(TMP, w_{OTP})$ {Repeat each row of $TMP$ $w_{OTP}$ times}
43: return $K_{SI_A}$
```


Value
Index
Randomly selected p Elements
$K_{SessInit_A}$ (Without the repeating rows)
Partial $K_{SessFnl}$
Partial $Idx_{SessFnl}$

(a) Randomly selecting elements along with their indexes from K_{SI_A}

(b) Generating K_{SF} and Idx_{SF}

Figure 4.4: Generating Final Session Key and Session Key Index Matrix from Initial Session Key of $Party_A$

and permuted creating another $n_{env} \times m_{pk}$ matrix called *Session Key Index Matrix*, Idx_{SF}. It should be noted that every $w_{OTP} - 1$ rows that has the repeating values in K_{SI_A} are ignored while picking elements for K_{SF}. The index matrix, Idx_{SF} is sent to $Party_B$ and she too form the same K_{SF} as $Party_A$ by picking the values pointed by elements in Idx_{SF} from her own *Initial Session Key*, K_{SI_B}.

Figure 4.4a shows how p elements along with their indexes are selected from K_{SI_A}. Later these values are permuted and repeated to create K_{SF} and Idx_{SF} as shown in 4.4b.

Security: An attacker, trying to impersonate as $Party_A$, may monopolize her chance to select Unique Elements randomly, by picking elements from K_{SI_A} that are encrypted by certain environment variables' data from the K_{itm_A}. To prevent such attack, $Party_B$ can provide the least number of different columns from which $Party_A$ must choose her elements to create Idx_{SF} and K_{SF} matrices.

4.5.4 Generate Session Public Key

Session Public Key, K_{SP} is generated by both $Party_A$ *and* $Party_B$ using the Primary Public Key, K_{pub_A} of $Party_A$ and Final Session Key, K_{SF}. K_{pub_A} is publicly available for anyone to access and its hashed values are used as K_{SP}. As we have seen in [34], *Short Integer Solution* can be used to generate *One-way Hash Functions*, in our work we applied the same process to generate the hash values of the Primary Public Key K_{pub_A} of $Party_A$.

We considered K_{pub_A} as a collection of m_{pk} vectors that are short in terms of Euclidean vector distance where each of these short vectors have m_{pk} dimensions. This makes K_{pub_A} a matrix with $m_{pk} \times m_{pk}$ dimensions. According to [34], we took each of these short vectors as input to the One-way Hash Function where we used

K_{SF} as the key. Thus,

$$\begin{aligned} &Let, K_{pub_A} = [\mathbb{x}_1\ \mathbb{x}_2\ ...\ ...\ ...\ \mathbb{x}_{m_{pk}}] \\ &Then, \mathbb{b}_i = (K_{SF} \times \mathbb{x}_i)\ mod\ p_{lg} \\ &where,\ i = \{1, 2, ..., m_{pk}\}\ and\ \mathbb{x}_i \in \{0, 1\}^{m_{pk}} \end{aligned} \tag{4.4}$$

The vectors, $\mathbb{b}_i$'s generated from equation 4.4 has n_{env} dimensions. We take all the $\mathbb{b}_i$'s and augment them to generate the K_{SP} matrix. Consequently from equation 4.5, K_{SP} matrix has $n_{env} \times m_{pk}$ dimensions.

$$\begin{aligned} &K_{SP} = [\mathbb{b}_1\ \mathbb{b}_2\ ...\ ...\ ...\ \mathbb{b}_{m_{pk}}] \\ &where,\ i = \{1, 2, ..., m_{pk}\}\ and\ \mathbb{b}_i \in \mathbb{Z}_{p_{lg}}^{n_{env}} \end{aligned} \tag{4.5}$$

Computational Complexity: From equation 4.4 and 4.5, the step of generating session public key takes $\mathcal{O}(m_{pk}^3)$ steps.

Security. From each vector in K_{pub_A} with m_{pk} dimensions, we get a vector with n_{env} dimensions in equation 4.4. We have seen earlier in section 4.5.3, $m_{pk} > (n_{env} * \lg p_{lg})$, and it means that equation 4.4 maps m_{pk} bits to smaller $n_{env} * \lg p_{lg}$ bits. Thus, equation 4.4 acts as a compression function. As we are compressing elements from a larger field to a smaller field, there are supposed to be collisions which was independently shown in both [42, 31]. However, from [34], it is realized that compression done by *Short Integer Solution* are essentially *One-Way Functions* and reversing the function is computationally hard in average-case since the compressed key, K_{SP} is calculated individually by both $Party_A\ and\ Party_B$, and leaks no information to the outside world.

4.5.5 Encrypt and Decrypt Message

In our work, we used LWE based public key encryption scheme where the Session Public Key by $Party_B$ is used to generate the encryption of message-bits whereas the Session Public Key of $Party_A$ is used to decrypt them. For distinguishing between these two Session Public Keys, which are generated seperately, let us assume that $Party_A$ and $Party_B$ generates K_{SP_A} and K_{SP_B} respectively. Our process includes the following steps:

1. The process starts with $Party_A$ by generating a private key $\mathbb{k}_{prv_A} \in \{0, 1\}^{n_{env}}$. Thus, $\mathbb{k}_{prv_A}$ is a small vector with n_{env} random 0 or 1 values. $\mathbb{k}_{prv_A}$ is the long-term private key of $Party_A$ which can be used throughout the communication or at the start of each session of communication with $Party_B$.

2. $Party_A$ encrypts K_{SP_A} with $\mathbb{k}_{prv_A}$ and perturb it with a column vector of m_{pk} small errors, ϵ_A by following the equation in 4.6.

$$\mathbb{k}_A = K_{SP_A}^T \times \mathbb{k}_{prv_A} + \epsilon_A$$
$$or,$$
$$\begin{bmatrix} k_{A_1} \\ k_{A_2} \\ . \\ . \\ . \\ k_{A_{m_{pk}}} \end{bmatrix} = K_{SP_A}^T \times \begin{bmatrix} k_{p_1} \\ k_{p_2} \\ . \\ . \\ . \\ k_{p_{n_{env}}} \end{bmatrix} + \begin{bmatrix} \epsilon_{A_1} \\ \epsilon_{A_2} \\ . \\ . \\ . \\ \epsilon_{A_{m_{pk}}} \end{bmatrix} \tag{4.6}$$

The small error terms in ϵ_A are obtained by sampling from a fixed Gaussian with mean $\mu = \frac{p_{lg}}{8*m_{pk}}$ and a standard deviation of $\rho = \mu + \sqrt{\frac{2*n_{env}}{\pi}}$. The error terms are also reduced to mod $\frac{p_{lg}}{4}$ [9]. Thus, $\mathbb{k}_A$ is a column vector with m_{pk} dimensions which is sent to $Party_B$.

3. $Party_B$, on the other hand, generates a column vector with m_{pk} small error terms ϵ_B, using the same error distribution mentioned for ϵ_A.

4. As $Party_B$ receives $\mathbb{k}_A$, it calculates a row vector, $\mathbb{u}_1$ of length n_{env} and a scalar u_2 following the equations in 4.7 and 4.8 respectively.

$$\mathbb{u}_1 = \epsilon_B^T \times K_{SP_B}^T \tag{4.7}$$

$$u_2 = \epsilon_B^T \times \mathbb{k}_A + msg_{bit} * \frac{p_{lg}}{2} \tag{4.8}$$

Here, msg_{bit} in equation 4.8 is the one-bit message that is encrypted by $Party_B$. Both $\mathbb{u}_1$ and u_2 are sent to $Party_A$ for decryption of msg_{bit}.

5. $Party_A$ calculates the value of u from equation 4.9 to decrypt the message-bit from $Party_B$.

$$u = (u_2 - \mathbb{u}_1 \times \mathbb{k}_{prv_A})\ mod\ p_{lg} \tag{4.9}$$

The value of u will be close to $\frac{p_{lg}}{2}$ or very far from $\frac{p_{lg}}{2}$ depending on the sent message-bit being 1 or 0 respectively.

Accuracy Proof. The proof of accuracy for equation 4.9 is given in Appendix D, where we have shown that accuracy of decryption depends on the deviation between K_{SP_A} and K_{SP_B}.

Chapter 5

Experimental Implementation

In this section, we have described how we processed data from SensorScope's sensor networks so that such data can be considered as observations and/or predictions of *Environment Variables* for our proposed protocol. At first, we have discussed the properties of the acquired datasets from SensorScope and then the process to convert sensor data is elaborated that was followed to implement and evaluate our proposed protocol.

5.1 Dataset Properties

In 2007, two small SensorScope networks were deployed to collect environmental data - one in Grand St. Bernerd Pass and another on top of rock glacier in Le Genepi. The sensor networks collected data for over one month from 23 different stations in Grand St. Bernerd Pass [51] and 16 different stations in Le Genepi [52]. Each recorded data from all these 39 different stations includes the station ID, the date and time of observation along with nine different sensor outputs - ambient temperature [$^\circ c$], surface temperature [$^\circ c$], solar radiation [W/m^2], relative humidity [%], soil moisture [%], watermark [kPa], rain meter [mm], wind speed [m/s] and wind direction [$^\circ$]. Any unavailable sensor output is represented as [NaN]. In our work, we considered each of these 39 different datasets contains 9 different time-series, each representing output of a specific sensor from SensorScope.

5.2 Implementation Details

Since we have acquired data from sensors that were recorded long time ago, just adding Gaussian noise to the data and using them to demonstrate our proposed protocol, lack real life implications. Thus, we assumed our protocol would get hold of a small portion of a time series containing all the sensor outputs and the rest of the time series data are to be predicted by LSTM-networks. Then we used Cross-Correlation to increase the features of predicted datasets and lastly applied LDA on them to shift axes of the features that generated the observations for environment

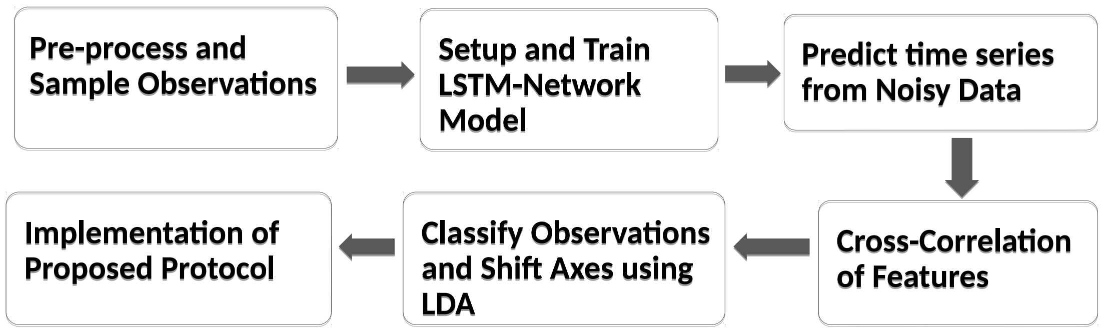

Pre-process and Sample Observations
Setup and Train LSTM-Network Model
Predict time series from Noisy Data
Cross-Correlation of Features
Classify Observations and Shift Axes using LDA
Implementation of Proposed Protocol

Figure 5.1: Experimental implementation steps

variables which are necessary for our proposed protocol to be implemented. The steps that lead to generating such desired observations are given in figure 5.1.

5.2.1 Pre-process and Sample Observations

After we acquired sensor data, we replaced $[NaN]$ values with 0. Though the numbers of observations from St Bernerd and Le Genepi datasets are different and immense at the same time, we wanted to sample a fixed number of observations for both of the datasets. Thus, for each of the 39 different datasets, we selected different intervals to sample observations and came up with 600 samples from each of them. Finally, each of the nine features - output from nine different sensors - are normalized to give them a range between 0.0 and 1.0.

5.2.2 Setup and Train LSTM-Network Model

Herein, our goal has been to create an LSTM-Network model that would, once trained, be able to predict sensor outputs of the next time step, given a series of sensor outputs of several previous steps. We used Keras [12] with Theano [2] back-end to build our network.

Using Keras, we created a deep network architecture with stacked layers of LSTM cells and exploited the fact that LSTM cells can build their states over a period of the whole training sequence. We added three stacked LSTM layers, each of which is stateful and contains 10 LSTM cells, to create a *Sequential* model. Moreover, the first two LSTM layers output the hidden states of their cells to the next layer for each time step. Each LSTM layer has a batch-size of 1, with a look-back period of 15 time-steps and 9 inputs. These 9 inputs are simply the 9 different sensors whose output are to be predicted by the model. Additionally, the model includes an final layer with 9 output fields that predict data from the sensors for the next time step. The model was compiled to minimize *Mean Squared Error* using *Adam (Adaptive Moment Estimation)* [28] optimization method.

We trained the model to predict the next time-step sensor data from every 15 previous time-steps. The training process ran for 120 epochs with single batch-size and no shuffling. The model state was deliberately reset after each training epoch had been completed. It should be noted that any options, not mentioned in this section but available to be customized by Keras, were set as default. For each of the 39 different normalized datasets, we trained separate LSTM-network models and used them to predict time-series from noisy data.

5.2.3 Predict Time-Series from Noisy Data

In this section, we use the trained LSTM-network models from section 5.2.2 to predict the noise-introduced datasets from section 5.2.1. Our main aim here is to predict the time-series data of the sensor outputs and to provide practical applicability of our proposed scheme. Once a model is trained with sensor data, any two parties who have access to the sensors, can observe sensor outputs for 15 time-steps and use the model to predict the rest of the output of the time-series. Otherwise, the parties would have to wait for more than one month to read sensor outputs again to start generating session keys, which is absurd for our scheme.

In our implementation, we demonstrated the predictability of our trained models for each of the 39 dataset that follows the steps described below.

1. After training the LSTM-network models, we added normally distributed random noise to each elements of the normalized and sampled datasets with mean 0 and standard deviations of 0%, 2%, 3% and 5% of the given elements, thus creating four different datasets from one.

2. Since our model takes 15 previous time-steps to predict the next step, for 600 observations of each of these four newly created datasets, we received 585 predicted observations from the model.

3. Finally, we reverse the scaling of every datasets from the range of $0.0 - 1.0$ to their pre-normalization range.

Figure 5.2 shows an example of time-series prediction on noisy data. It depicts the predictions of LSTM-network model trained on the dataset generated by a particular station. Each of the sub-graphs in figure 5.2 represents predictions of an individual sensor outputs over time. Each sub-graph also shows the predicted outputs of the LSTM-network model when fed with data containing normally distributed random noise which has a mean of 0 and standard deviations of 0%, 2%, 3% and 5% of original normalized data.

5.2.4 Cross-Correlation of Features

For each of the datasets of SensorScope, we, by now, have four different predicted outputs of sensors from LSTM-network models. Each of such predicted outputs have nine-dimensional features where each dimension represents the individual outputs of weather sensors. Later, we calculate the cross-correlation of these features with a major goal to increase feature-dimensions. All possible pair of features from the initial nine features are taken into consideration and their cross-correlations are calculated

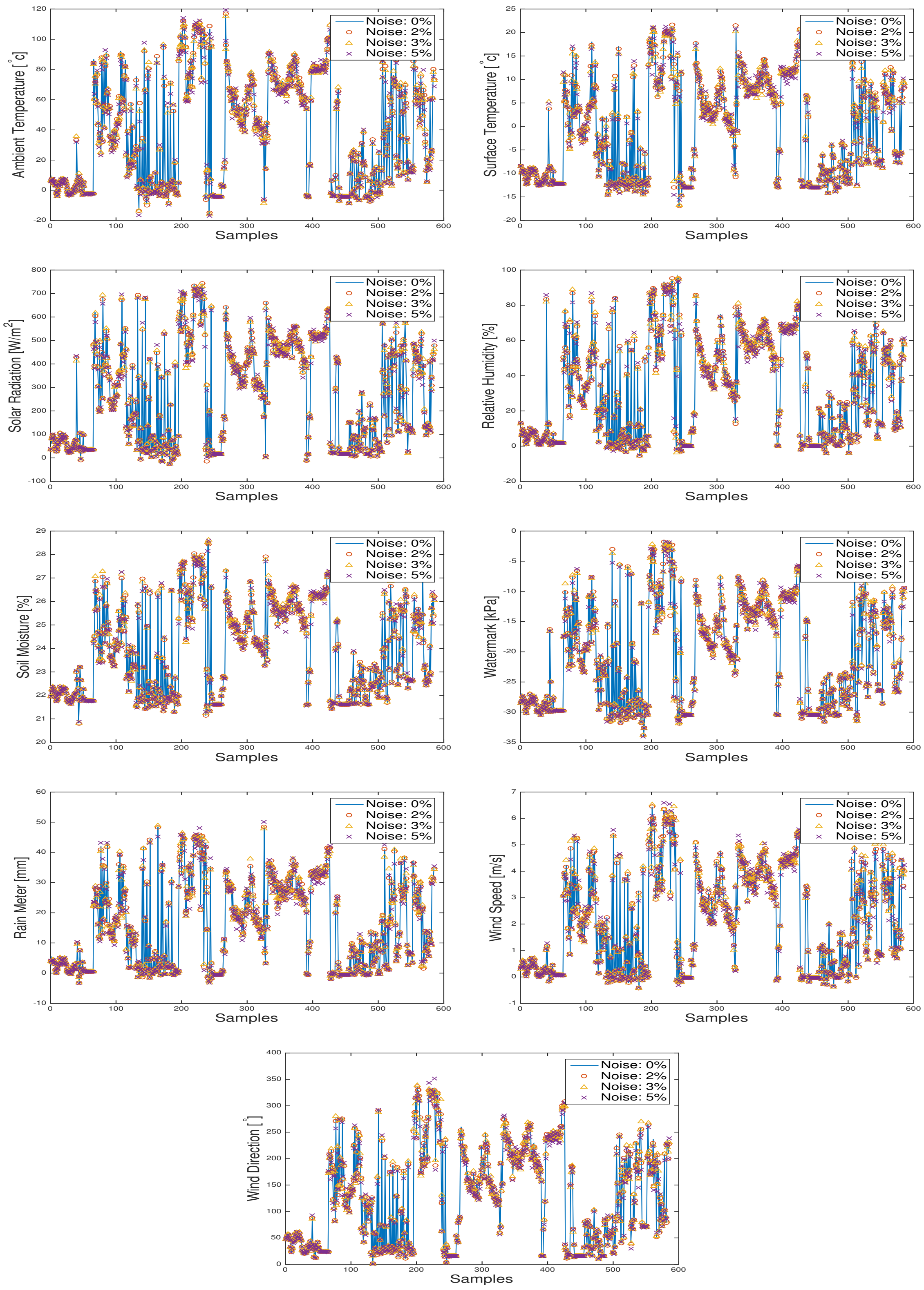


Figure 5.2: Prediction comparison of noisy data originated from a station in Grand St. Bernerd Pass with station ID 2

Algorithm 3 Calculate Cross-Correlation

```
Require: $\mathbb{x}_1$, $\mathbb{x}_2$, $timeSteps$, $||\mathbb{x}_1|| == ||\mathbb{x}_2||$
1: $len \leftarrow ||\mathbb{x}_1||$
2: $\mathbb{y} : ARRAY[len]$
3: for $i = 1$ to $len$ do
4:     $xx \leftarrow 0$
5:     $yy \leftarrow 0$
6:     $xy \leftarrow 0$
7:     $j \leftarrow i$
8:     loop
9:         if $j > 0$ and $j > (i - timeSteps)$ then
10:            $xx \leftarrow xx + (\mathbb{x}_1[j] \times \mathbb{x}_1[j])$
11:            $yy \leftarrow yy + (\mathbb{x}_2[j] \times \mathbb{x}_2[j])$
12:            $xy \leftarrow xy + (\mathbb{x}_1[j] \times \mathbb{x}_2[j])$
13:            $j \leftarrow j - 1$
14:        else
15:            break
16:        end if
17:    end loop
18:    $\mathbb{y}[i] \leftarrow \sqrt{\frac{xy \times xy}{xx \times yy}}$
19: end for
20: return $\mathbb{y}$
```

which are incorporated as new features. Thus, from initial nine features, we now have $9 + \binom{9}{2} = 45$ features for each dataset. Moreover, since we have taken 15 time-steps to predict the next step, when calculating the cross-correlation, the values are generated going at most 15 time-steps back. The procedure to calculate cross-correlation from two different features and a predetermined time-step is given in algorithm 3.

5.2.5 Classify Observations and Shift Axes using LDA

After increasing the number of features using Cross-Correlation, our datasets, predicted by the LSTM-network model, now contain 45 features and 585 observations. The observations from where we predicted these datasets spanned throughout a period of more than one month. As the evidence suggests from the time series of figure 5.2, these predicted observations do not show any particular seasonality or trend. Consequently, we classified the predicted datasets based on the date they were recorded. Thus, all 585 observations of a particular predicted dataset fell under c classes, where c is the number of unique dates when the actual observations were made. Our assumption behind classifying the observations in such manner is that the data varied slightly within one day which is also evident from figure 5.2 where we can see that sudden fluctuations in values in the time-series tend to persist for a while before showing any drastic change again. After classifying the predicted observations, we shifted the feature-axes to further minimize the inter-class distance among their values by applying LDA on the predicted dataset of every stations which were introduced with no noise. The Eigen Vectors acquired from this process were then used to shift the axes of the predicted datasets with noise having standard deviations of 2%, 3% and 5% and were originated from the same station. This whole process of classifying and shifting feature axes by implementing LDA essentially results in decreasing the magnitude of the predicted values which in turn means that any deviation or noise in predicted values will impact less when generating Session Keys using One-Time pads - the process we have already described in section 4.5.3.

5.2.6 Implementation of Proposed Protocol

Throughout chapter 4, we referred to the behaviors of the pervasive devices as observations from the environment variables. Similarly, during our implementation of the proposed protocol, we realized the predicted sensor outputs as observations of environment variables. The predicted datasets with 0% error we generated here for each station, are considered to be the observations of the environment variables made by $Party_A$. On the other hand, we considered noisy datasets for each station, are observations for $Party_B$. We also tested the decryption accuracy of our protocol where $Party_A$ and $Party_B$ are from two separate environment. As we evaluated our implementations, we have found that increasing noise in the observations of $Party_B$ decreases the decryption accuracy. In all these cases, our tests include results where we varied *Intermediate Key* length, number of *Unique Elements* and number of random elements selected from *Initial Session Key* of $Party_A$ to create the *Final Session Key.* In chapter 6, we have described, in details, the tests we have performed on our protocol and the results we have acquired from these tests.

Since we have processed sensor data to implement the proposed protocol which generates session key for our encryption scheme, we have managed to answer our first **Research Question (RQ1)**.

Chapter 6

Accuracy Evaluation

The main goals of implementing our work are twofold. Primarily, we have aspired to demonstrate that as we introduce varying noise in the observations of environment variables, the decryption accuracy of messages varies accordingly. Secondly, it is also shown that when the communicating parties use observations of variables from two different environments, it results in absolutely random decryption of message-bits. These are the results we have discussed in this section while comparing them against the prediction accuracy of our LSTM-network model.

6.1 Prediction Accuracy of LSTM-Network Model

The prediction accuracy of our LSTM-Network model is calculated against the network predicted dataset which had no added noise prior to the prediction. We considered *Normalized Root-Mean-Squared Error (NRMSE)* as the measure of our prediction accuracy. For predicted datasets that were introduced with prior noise having 2%, 3% and 5% *Standard Deviations*, we found their average NRMS errors as 7.03%, 10.2% and 16.2% respectively, while comparing them with datasets having no noise introduced. Moreover, in order to demonstrate such deviations in observations between different environments, we calculated the NRMS errors between two different weather stations that were located in separate locations, namely- Grand St. Bernerd and Genepi. For such calculation of accuracy, we found that our network model, trained in dataset of one environment, fails to predict the datasets from another one and had an NRMS error that is more than 100%. It should be noted that in all those cases, lower values of NRMSE mean the model predicted datasets are predicted more accurately than the other way round.

6.2 Comparison of Decryption Accuracy

During the implementation of our proposed protocol, we tested decryption accuracy by taking three different length of *Intermediate Keys* - 300, 400 and 450 - and two different w_{OTP}s -10 and 20 - to compare the results. Moreover, we tested our protocol where we used $2, 4, 6$ and 8 unique elements along with varying the number of random

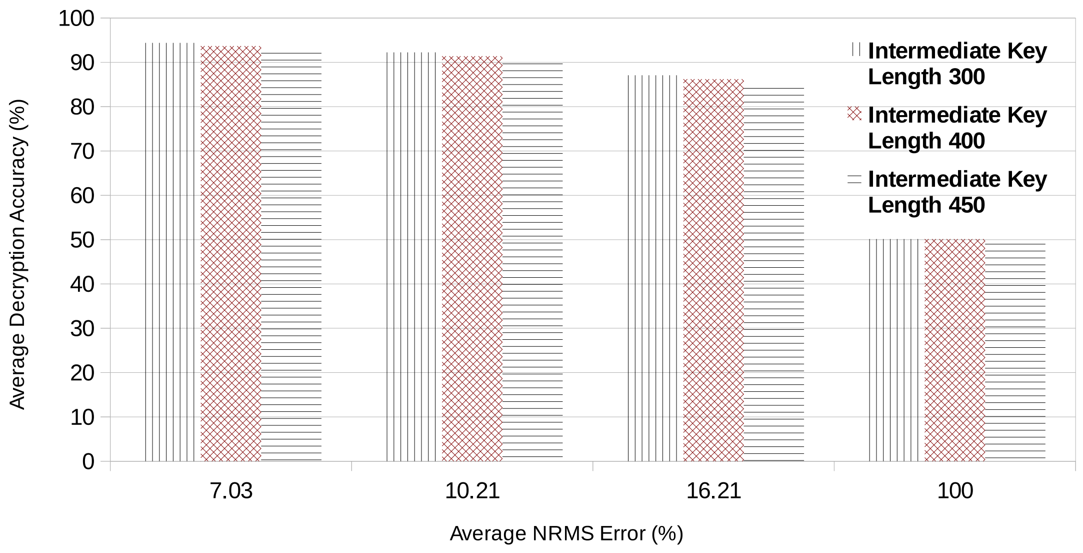


Figure 6.1: Average Decryption Accuracy over Noisy Data

elements picked by $Party_A$ to create K_{SF} from 45 to 200. Each of these tests were repeated 25 times for every weather stations over varying noisy datasets and weather stations from different environments. The calculation of decryption accuracy is conducted by $Party_B$ encrypting 1000 message bits and measuring the percentage of bits that were decrypted accurately at $Party_A$'s end, while varying the parameters when generating the *session keys*. This should be noted that since the message contains only 1 or 0, a true random decryption of messages would get 50% of the message bit correct.

Figure 6.1 shows the summary of average decryption accuracy when varying the noise in predicted datasets along with different length for *Intermediate Keys*. As we can see for datasets having an NRMS error of 100% aamong them, we get an accuracy around 50%, which represents random decryption of message-bits. Moreover, with the introduction of more noise in predicted datasets, decryption accuracy falls accordingly. Additionally, it can be seen from the results in figure 6.1 that reducing the length of *Intermediate Keys* give more accurate decrypted messages.

As we tested our protocol by varying the unique elements count and the number of random elements picked from K_{SI_A}, we can see from figure 6.2 that decreasing both of these parameters result in high accuracy in decryption. With 2 unique elements and 45 randomly picked elements from K_{SI_A}, we can achieve decryption accuracy as high as 96% for datasets with 7.03% NRMS error.

We added variable noise to sensor data and our proposed protocol generates decryption accuracy according to the induced noise, we have managed to answer the second **Research Question (RQ2)**.

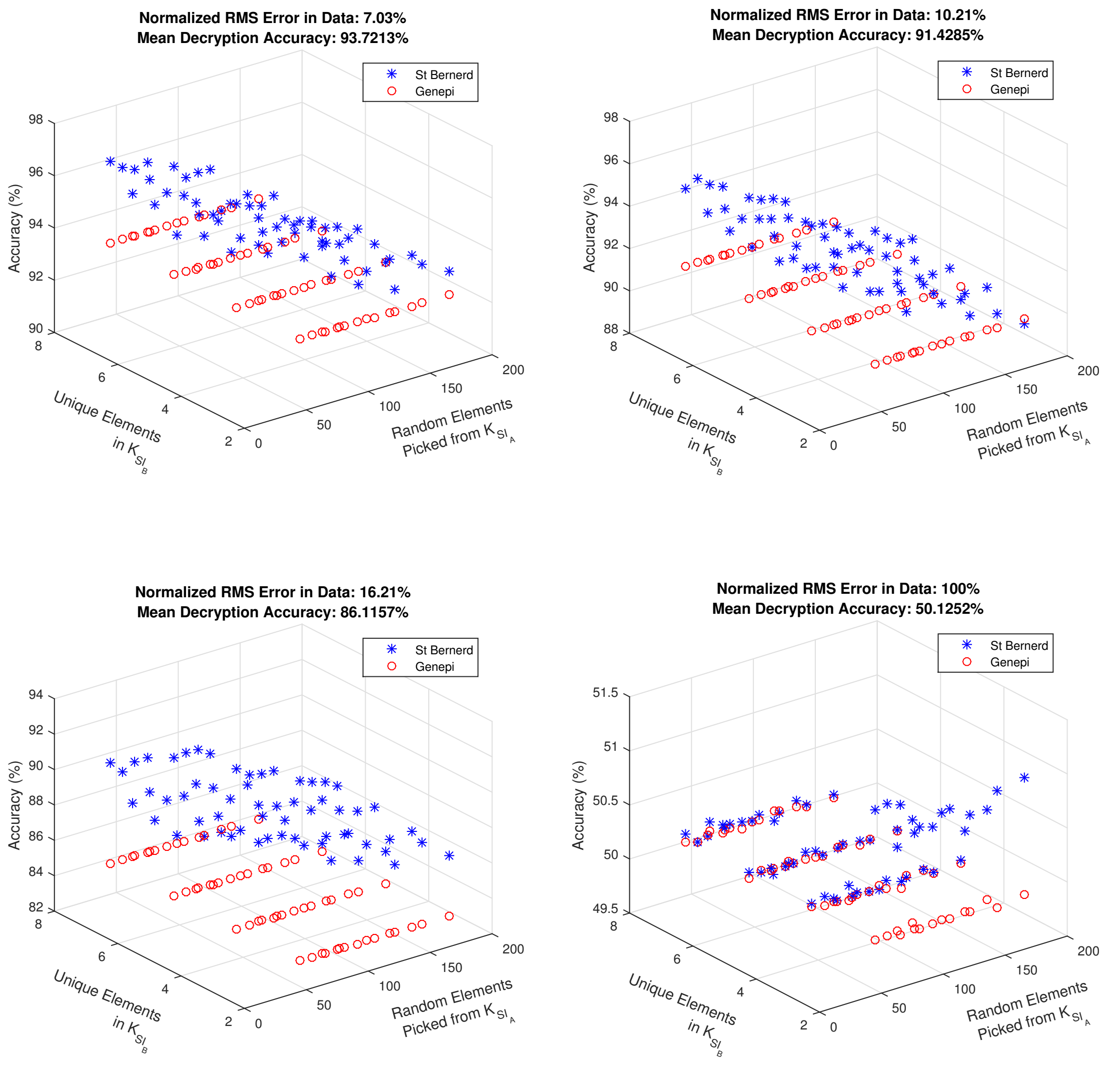


Figure 6.2: Decryption accuracy against varying *Unique Elements* count and Random Elements from *Initial Session Key* of $Party_A$

Chapter 7

Conclusion

Numerous works related to Security in IoT pointed out one of the most vulnerable fact about these devices - their heterogeneity. It is the property that hinders any standardization of the security protocols. However, in this work we intended to use this very heterogeneity property of IoT devices to our advantage in establishing a secure communication protocol between two parties. Going back to the attack discussed in 1.1, if the victim servers were to consider the client behavioral patterns to encrypt and decrypt payloads, as we proposed in this work, then all the encrypted payloads from compromised IoT devices would have been impossible to decrypt. Upon detecting such flawed communication, the servers could have identified such attacking devices and blocked any further communication.

One of the major hurdle in going forward with the proposed protocol is to minimize the data payload between the communicating parties while generating session keys. Though we did not concern ourselves with the efficiency of our protocol in this work, this aspect should be looked into in any future direction of this proposal. Moreover, to handle mismatched session-keys so that such incident could be determined by both parties before passing information, further investigation should be done and standard protocols should be established. Finally, our implementation of the proposed protocol deals with only time-series data from weather sensors. More work is required to implement this protocol in other data generating fields including image processing, HCI etc.

Finally, it might not be self evident how an encryption scheme can be useful if some of the decrypted message-bit are flawed. We envision a new research area to work with this type of flawed decryption, probably similar to network communication where established protocols are there to manage dropped packets during transfer of messages. But such schemes should be attractive given the difficulty one may face to predict the behavior of an elaborate heterogeneous IoT environment.

Appendix A

Modular Multiplicative Inverse

if p and q are coprimes, then *Modular Multiplicative Inverse* of $p \pmod{q}$ exists and it essentially means to find the inverse of p in p modulo q.

For a worked example, let, $p = 17$ and $q = 780$. Since, $gcd(17, 780) = 1$, the inverse of 17 modulo 780 exists. Let, the inverse is $\bar{p}$. Thus,

$$\begin{aligned} 17 \text{ moduo } 780 &\implies \bar{p} \times 17 \equiv 1 \pmod{780} \\ &\implies \bar{p} \times 17 = k \times 780 + 1 \\ &\implies 1 = \bar{p} \times 17 - k \times 780 \\ &\text{where, } k \in \mathbb{Z} \end{aligned} \tag{A.1}$$

Applying *Extended Euclidean Algorithm* in equation A.1, we get,

$$\begin{aligned} 780 &= 45 \times 17 + 15 \\ 17 &= 1 \times 15 + 2 \\ 15 &= 7 \times 2 + 1 \end{aligned} \tag{A.2}$$

Now from equation A.2, we get,

$$\begin{aligned} 1 &= 15 - 7 \times 2 \\ &= 15 - 7 \times (17 - 1 \times 15) \\ &= (780 - 45 \times 17) - 7 \times (17 - 1 \times (780 - 45 \times 17)) \\ &= -367 \times 17 + 8 \times 780 \end{aligned} \tag{A.3}$$

From equation A.1 and A.3, we get,

$$\bar{p} = -367 \text{ and } k = 8$$

Since, $\bar{p} \times 17 \equiv 1 \pmod{780}$, equation A.1 holds for all values of $\bar{p}$ where,

$$\bar{p} \in \{..., -367 + (-1) \times 780, -367 + 0 \times 780, -367 + 1 \times 780, ...\}$$

Appendix B

Derivation of Equation 2.8

The equation of the elliptic curve, E is given by equation 2.7.

B.1 Calculating Point Addition

The equation of a line L through points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ can be given as:

$$\begin{aligned}
&\frac{x - x_1}{x_1 - x_2} = \frac{y - y_1}{y_1 - y_2}\\
\Longrightarrow &\frac{x - x_1}{y - y_1} = \frac{x_1 - x_2}{y_1 - y_2}\\
\Longrightarrow &\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}\\
\Longrightarrow &\frac{y - y_1}{x - x_1} = m;\ \text{[The slope of the line]}\\
\Longrightarrow &y = m(x - x_1) + y_1
\end{aligned} \tag{B.1}$$

Now, putting the value of y from equation B.1 in equation 2.7. we get,

$$\begin{aligned}
&y^2 = x^3 + ax + b\\
\Longrightarrow &\{m(x - x_1) + y_1\}^2 = x^3 + ax + b\\
\Longrightarrow &m^2(x^2 - 2x_1x + x_1^2) + y_1^2 + 2m(x - x_1) = x^3 + ax + b\\
\Longrightarrow &x^3 + (-m^2)x^2 + (a + 2x_1m^2 - 2m)x + (b - m^2x_1^2 - y_1^2 + 2mx_1) = 0
\end{aligned} \tag{B.2}$$

Since we know the points P_1 and P_2, which should not be the same for point addition, it is understood that x_1 and y_1 in equation B.2 are constants along with a, m and b.

Again, the equation in B.2 is a *Cubic Polynomial*. Let the line L intersects the elliptic curve, E, at point $S(x_s, y_s)$. So the three roots of the equation B.2 will be x_1, x_2 and x_s. From Viete theorem in [25], we know that for any cubic polynomial, where, $P(x) = ax^3 + bx^2 + cx + d$ with roots r_1, r_2 and r_3, $P(x) = 0$ satisfy,

$$r_1 + r_2 + r_3 = -\frac{b}{a} \tag{B.3}$$

Thus from equation B.2, we get,

$$\begin{aligned} x_1 + x_2 + x_s &= -\frac{-m^2}{1} \\ \implies x_s &= m^2 - x_1 - x_2 \end{aligned} \tag{B.4}$$

Putting the value of $S(x_s, y_s)$ from equation B.4 in B.1, we get,

$$y_s = m(x_s - x_1) + y_1 \tag{B.5}$$

When we reflect S over the x-axis, the sign of the y-coordinate changes. Thus the equations from B.4 and B.5 becomes B.6, where the point $P_3(x_3, y_3)$ is located where $y_3 = -y_s$ and $x_3 = x_s$ since the values of x-coordinates remain unchanged.

$$\begin{aligned} x_3 &= m^2 - x_1 - x_2 \\ y_3 &= m(x_3 - x_1) - y_1 \\ where&, \\ m &= \frac{y_2 - y_1}{x_2 - x_1} \end{aligned} \tag{B.6}$$

B.2 Calculating Point Doubling

In Point Doubling for Elliptic Curves, the given points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ are the same. Thus, $x_1 = x_2$ and $y_1 = y_2$. Let, L be a tangent line touching the elliptic curve, E from equation 2.7 at points $P_1(x_1, y_1)$ and intersecting E on point $P_s(x_s, y_s)$. The coordinates of P_1 is given and we need to find out the point P_3 and its reflection along x-axis, $P_3(x_3, y_3)$.

If the slope of the line, L, is m and it crosses the y-axis at point $(0, c)$, the equation of the line becomes:

$$y = mx + c \tag{B.7}$$

Moreover, the slope of a curve can be calculated by taking the partial derivative of the curve. For the elliptic curve, E, the slope $m = \frac{dy}{dx}(x_1, y_1)$

$$\begin{aligned} \frac{dy}{dx}(y^2) &= \frac{dy}{dx}(x^3 + ax + b) \\ \implies 2y(\frac{dy}{dx}) &= 3x^2 + a \\ \implies \frac{dy}{dx} &= \frac{3x^2 + a}{2y} \end{aligned} \tag{B.8}$$

Thus we can get the value of m from equation B.8:

$$m = \frac{3x_1^2 + a}{2y_1}$$

Now putting the value of y from equation B.7 in 2.7, we get,

$$\begin{aligned} &(mx + c)^2 = x^3 + ax + b \\ \implies &x^3 - m^2x^2 - 2mxc - c^2 + ax + b = 0 \\ \implies &x^3 + (-m^2)x^2 + (a - 2mc)x + (b - c^2) = 0 \end{aligned} \tag{B.9}$$

Since the roots of the cubic polynomial are x_1 and x_s where the line L touches the curve at $P_1(x_1, y_1)$, from equation B.3, we get,

$$\begin{aligned} &2x_1 + x_s = -\frac{(-m^2)}{1} \\ \implies &x_1 + x_2 + x_s = m^2 \text{ ;[since } x_1 = x_2] \\ \implies &x_s = m^2 - x_1 - x_2 \end{aligned} \tag{B.10}$$

As the line L goes through points $P_1(x_1, y_1)$ and $P_s(x_s, y_s)$, the equation of the line can be given by,

$$\begin{aligned} &\frac{x - x_1}{x_1 - x_s} = \frac{y - y_1}{y_1 - y_s} \\ \implies &y - y_1 = \frac{(x - x_1)(y_1 - y_s)}{x_1 - x_s} \\ \implies &y = m(x - x_1) + y_1 \text{ ;[Since m is the slope of the line]} \end{aligned} \tag{B.11}$$

Now, reflecting equation B.10 and B.11 over x-axis we get the point $P_3(x_3, y_3)$. Thus the values of the coordinates can be calculated by:

$$\begin{aligned} x_3 &= m^2 - x_1 - x_2 \\ y_3 &= m(x_3 - x_1) - y_1 \\ where&, \\ m &= \frac{3x_1^2 + a}{2y_1} \end{aligned} \tag{B.12}$$

Appendix C

Range of Unique Elements

When $Party_B$ selects the unique elements, she needs to consider two thresholds: *Average-Case Error Threshold* (th_{avg}) and *Worst-Case Error Threshold* (th_{max}). Let, $Party_B$ selects el_{uniq} unique elements and they are sorted in ascending order: $\{el_1 < el_2 < ... < el_{el_{uniq}}\}$. The following lemma relates the sparseness of the unique elements to the Error Thresholds.

Lemma 1. *For a set of Unique Elements* $\mathbb{UE}$ *with* el_{uniq} *elements, where* $\mathbb{UE} = \{el_1, el_2, ..., el_{el_{uniq}}\}$ *and* $el_1 < el_2 < ... < el_{el_{uniq}}$*, for each* $el_i \in \mathbb{UE}$*,*

$$\frac{100 * el_{i-1}}{100 - th_{avg}} < el_i < \frac{100 * el_1}{100 - th_{max}}$$

$$or,$$

$$\frac{100 * el_{i+1}}{100 + th_{avg}} > el_i > \frac{100 * el_1}{100 + th_{max}}$$

Here we assumed, Average-Case Error Threshold is $th_{avg}\%$ *and Worst-Case Error Threshold is* $th_{max}\%$ *.*

Proof. Let, $val_A = K_{itm_A}(x, y)$ and $val_B = K_{itm_B}(x, y)$. For any two Unique Elements el_i and el_j, where $el_i, el_j \in \mathbb{UE}$, let us consider the Session Key Seed, $K_{SS}(x, y) = val_B * el_i$. Now, let $val = K_{SI_A}(x, y)$. Thus,

$$\begin{aligned} val &= \frac{val_B * el_i}{val_A} \\ &= \left\{ \begin{array}{ll} \frac{100*el_i}{100-th_{avg}}; & if, val_A = (1 - \frac{th_{avg}}{100}) * val_B \\ \frac{100*el_i}{100+th_{avg}}; & if, val_A = (1 + \frac{th_{avg}}{100}) * val_B \end{array} \right\} \end{aligned}$$

Hence, to select a value for el_j which will ensure that no unique element can be rounded off as the next one,

$$el_j > \frac{100 * el_i}{100 - th_{avg}} \; or, \; el_j < \frac{100 * el_i}{100 + th_{avg}}$$

Conversely, to ensure that el_j value will result in rounding off to some other unique element randomly, given NRMS error more than Worst-Case Error Threshold, we need to consider el_1 as the smallest element in $\mathbb{UE}$. Thus,

$$el_j < \frac{100 * el_1}{100 - th_{max}} \ or, \ el_j > \frac{100 * el_1}{100 + th_{max}}$$

So, for any two Unique Elements $el_{i-1}\ and\ el_i$, where, $el_i > el_{i-1}$,

$$\frac{100 * el_{i-1}}{100 - th_{avg}} < el_i < \frac{100 * el_1}{100 - th_{max}}$$

Moreover, for any two Unique Elements $el_{i+1}\ and\ el_i$, where, $el_i < el_{i+1}$,

$$\frac{100 * el_{i+1}}{100 + th_{avg}} > el_i > \frac{100 * el_1}{100 + th_{max}}$$

Appendix D

Proof of Accuracy for Decryption

If we replace $\mathbb{u}_1$ and u_2 of Equation 4.9 with their values from equation 4.7 and equation 4.8 respectively, we get equation D.1.

$$\begin{aligned} u = ((\epsilon_B^T \times \mathbb{k}_A + msg_{bit} * \frac{p_{lg}}{2}) \\ - (\epsilon_B^T \times K_{SP_B}) \times \mathbb{k}_{prv_A}) \ mod \ p_{lg} \end{aligned} \tag{D.1}$$

Moreover, if we replace $\mathbb{k}_A$ with equation 4.6, we get equation D.2.

$$\begin{aligned} u = ((\epsilon_B^T \times (K_{SP_A}^T \times \mathbb{k}_{prv_A} + \epsilon_A) \\ + msg_{bit} * \frac{p_{lg}}{2}) - (\epsilon_B^T \times K_{SP_B}^T) \times \mathbb{k}_{prv_A}) \ mod \ p_{lg} \end{aligned} \tag{D.2}$$

Now, applying distribution over matrix addition and association over matrix multiplication in equation D.2 we get equation D.3.

$$\begin{aligned} u = ((\epsilon_B^T \times (K_{SP_A}^T - K_{SP_B}^T) \times \mathbb{k}_{prv_A}) \\ + (\epsilon_B^T \times \epsilon_A) + msg_{bit} * \frac{p_{lg}}{2}) \ mod \ p_{lg} \end{aligned} \tag{D.3}$$

From equation D.3, we can see that the term $(K_{SP_A}^T - K_{SP_B}^T)$ determines the accuracy of decryption for $msg_{bit} * \frac{p_{lg}}{2}$, since $(\epsilon_B^T \times \epsilon_A)$ is very small compared to p_{lg}. Thus, to improve the accuracy of decryption, the deviation between K_{SP_A} and K_{SP_B} should be minimized.

The normalized noise ϵ_B^T in equation D.3 presents a probabilistic deviation by working on $(K_{SP_A}^T - K_{SP_B}^T)$. Moreover we can rewrite $(K_{SP_A}^T - K_{SP_B}^T) \times \mathbb{k}_{prv_A}$ as two different SIS functions where the key is $\mathbb{k}_{prv_A}$. In other words, $(K_{SP_A}^T \times \mathbb{k}_{prv_A}) - (K_{SP_B}^T \times \mathbb{k}_{prv_A})$. Thus, the whole probabilistic deviation is compressed into a linear combination of vectors where the wight of the vectors are 1 or 0 (since, $\mathbb{k}_{prv_A} \in \{0,1\}^{n_{env}}$. As we are compressing two larger set to a smaller set with the same key, both the compressed vectors should fall close to each other, should the deviation between the two larger sets are small. And finally, the position of the compressed vectors are perturbed by the noise ϵ_B^T which determines the decryption accuracy in a probabilistic way.

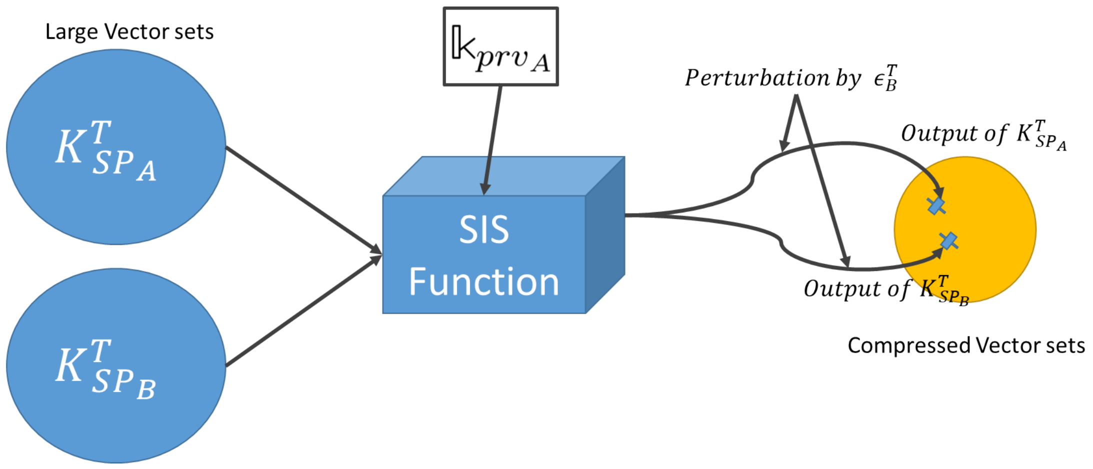


Figure D.1: Compression using SIS function in $\epsilon^T_B \times (K^T_{SP_A} - K^T_{SP_B}) \times \Bbbk_{prv_A}$

The process is depicted more clearly in Figure D.1. Here we can see that two large vector fields are going through SIS function and finally the output fiel is constrained. However, the error induced by ϵ^T_B has the propensity to make the output pair closer or take them further away. And this is the reason when the message is decrypted, some of them tend to present flawed value and the decryption depends upon the closeness between K_{SP_A} and K_{SP_B}.

Bibliography

[1] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 99–108, 1996.

[2] Rami Al-Rfou, Guillaume Alain, et al. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[3] Abdullah Al-Tariq, Abu Raihan Mostofa Kamal, et al. A scalable framework for protecting user identity and access pattern in untrusted web server using forward secrecy, public key encryption and bloom filter. *Concurrency and Computation: Practice and Experience*, 29(23), 2017.

[4] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium*, Vancouver, BC, Canada, August 2017.

[5] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J Wu. Private database queries using somewhat homomorphic encryption. In *International Conference on Applied Cryptography and Network Security*, pages 102–118. Springer, 2013.

[6] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[7] Marcus Brandenburger, Christian Cachin, and Nikola Knežević. Dont trust the cloud, verify: Integrity and consistency for cloud object stores. *ACM Transactions on Privacy and Security (TOPS)*, 20(3):8, 2017.

[8] A. Brandt and J. Buron. *Home automation routing requirements in low-power and lossy networks.* [Online]. Available: `https://tools.ietf.org/html/rfc5826`.

[9] Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. On error distributions in ring-based lwe. *LMS Journal of Computation and Mathematics*, 19(A):130–145, 2016.

[10] Qian Chen, Sherif Abdelwahed, and Abdelkarim Erradi. A model-based validated autonomic approach to self-protect computing systems. *IEEE Internet of Things Journal*, 1(5):446–460, 2014.

[11] S. R. Chhetri, A. Canedo, and M. Al Faruque. Confidentiality breach through acoustic side-channel in cyber-physical additive manufacturing systems. *ACM Transactions on Cyber-Physical Systems (TCPS)*, 2016.

[12] François Chollet et al. Keras. `https://github.com/keras-team/keras`, 2015.

[13] CISCO. *The Internet of Things reference model.*, 2014. Available at `http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf`.

[14] Shane S Clark, Benjamin Ransford, Amir Rahmati, Shane Guineau, Jacob Sorber, Wenyuan Xu, Kevin Fu, A Rahmati, M Salajegheh, D Holcomb, et al. Wattsupdoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices. In *HealthTech*, 2013.

[15] Michele De Donno, Nicola Dragoni, Alberto Giaretta, and Manuel Mazzara. Antibiotic: Protecting iot devices against ddos attacks. In *International Conference in Software Engineering for Defence Applications*, pages 59–72. Springer, 2016.

[16] Michele De Donno, Nicola Dragoni, Alberto Giaretta, and Angelo Spognardi. Analysis of ddos-capable iot malwares. In *Proceedings of 1st International Conference on Security, Privacy, and Trust (INSERT)*, 2017.

[17] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[18] Junqi Duan, Deyun Gao, Dong Yang, Chuan Heng Foh, and Hsiao-Hwa Chen. An energy-aware trust derivation scheme with game theoretic approach in wireless sensor networks for iot applications. *IEEE Internet of Things Journal*, 1(1):58–69, 2014.

[19] Pierre Dusart and Sinaly Traoré. Lightweight authentication protocol for low-cost rfid tags. In *IFIP International Workshop on Information Security Theory and Practices*, pages 129–144. Springer, 2013.

[20] PUB FIPS. 186-4. *Digital Signature Standard (DSS)*, 2013.

[21] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[22] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471, October 2000.

[23] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[24] Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[25] Michiel Hazewinkel. Viète theorem. *Encyclopedia of Mathematics*, 2001.

[26] Nikolaos Karapanos, Alexandros Filios, Raluca Ada Popa, and Srdjan Capkun. Verena: End-to-end integrity protection for web applications. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 895–913. IEEE, 2016.

[27] S McCURLEY Kevin. The discrete logarithm problem. *Cryptology and computational number theory*, 42:49, 1990.

[28] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen Lenstra, Emmanuel Thomé, Joppe Bos, Pierrick Gaudry, Alexander Kruppa, Peter Montgomery, Dag Arne Osvik, et al. Factorization of a 768-bit rsa modulus. In *CRYPTO 2010*, volume 6223, pages 333–350. Springer, 2010.

[30] Arjen K Lenstra. Integer factoring. *Designs, codes and cryptography*, 19(2):101–128, 2000.

[31] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *Proceedings of Proceedings of ICALP*, pages 144–155, Venice, Italy, July 2006.

[32] Diego M Mendez, Ioannis Papapanagiotou, and Baijian Yang. Internet of things: Survey on security and privacy. *arXiv preprint arXiv:1707.01879*, 2017.

[33] Daniele Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Transactions on Information Theory*, 47(3):1212–1215, March 2001.

[34] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, Vancouver, BC, Canada, February 2002.

[35] Evgeny Milanov. The rsa algorithm. *RSA Laboratories*, 2009.

[36] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 417–426. Springer, 1985.

[37] Arsalan Mosenia and Niraj K. Jha. A comprehensive study of security of internet-of-things. *IEEE Transactions on Emerging Topics in Computing*, 5:586–602, Oct.-Dec. 2017.

[38] Arsalan Mohsen Nia, Susmita Sur-Kolay, Anand Raghunathan, and Niraj K Jha. Physiological information leakage: A new frontier in health information security. *IEEE Transactions on Emerging Topics in Computing*, 4(3):321–334, 2016.

[39] Job Noorman, Jo Van Bulck, Jan Tobias Mühlberg, Frank Piessens, Pieter Maene, Bart Preneel, Ingrid Verbauwhede, Johannes Götzfried, Tilo Müller, and Felix Freiling. Sancus 2.0: A low-cost security architecture for iot devices. *ACM Transactions on Privacy and Security (TOPS)*, 20(3):7, 2017.

[40] Guevara Noubir and Guolong Lin. Low-power dos attacks in data wireless lans and countermeasures. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(3):29–30, 2003.

[41] Abdullah Nazma Nowroz, Kangqiao Hu, Farinaz Koushanfar, and Sherief Reda. Novel techniques for high-sensitivity hardware trojan detection using thermal and power maps. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(12):1792–1805, 2014.

[42] C. Peikert and A. Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *Theory of Cryptography: TCC*, pages 145–166, 2006.

[43] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *Proceedings of the 41st ACM Symposium on Theory of Computing*, 2009.

[44] Chris Peikert. Lattice cryptography for the internet. In *International Workshop on Post-Quantum Cryptography*, pages 197–219. Springer, 2014.

[45] Raluca A Popa and Nickolai Zeldovich. Multi-key searchable encryption. *IACR Cryptology ePrint Archive*, 2013:508, 2013.

[46] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[47] Raluca Ada Popa, Emily Stark, Jonas Helfer, Steven Valdez, Nickolai Zeldovich, M Frans Kaashoek, and Hari Balakrishnan. Building web applications on top of encrypted data using mylar. In *NSDI*, pages 157–172, 2014.

[48] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):34, 2009.

[49] Karim Rostamzadeh, Hasen Nicanfar, Narjes Torabi, Sathish Gopalakrishnan, and Victor CM Leung. A context-aware trust-based information dissemination framework for vehicular networks. *IEEE Internet of Things Journal*, 2(2):121–132, 2015.

[50] Sepp Hochreiter; Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, Nov. 1997.

[51] Sensorscope. *Grand-St-Bernard Deployment*, 2007. Available at `http://lcav.epfl.ch/files/content/sites/lcav/files/research/Sensorscope/stbernard-meteo.zip`.

[52] Sensorscope. *Le Genepi Deployment*, 2007. Available at `http://lcav.epfl.ch/files/content/sites/lcav/files/research/Sensorscope/genepi-meteo.zip`.

[53] D. M. Shila and V. Venugopal. Design, implementation and security analysis of hardware trojan threats in fpga. In *Proceedings of IEEE International Conference on Communication*, pages 719–724, 2014.

[54] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal of Computing*, 26(5):1484–1509, 1997.

[55] F. Stajano. The resurrecting duckling. In *Proceedings of Security Protocols*, pages 183–194, 2000.

[56] M. Tehranipoor and F. Koushanfar. A survey of hardware trojan taxonomy and detection. *IEEE Design and Test of Computers*, 27(1):10–25, 2010.

Publication List

[1] Abdullah Al-Tariq and Abu Raihan Mostofa Kamal. Forward-secrecy and group membership verification using behavioral patterns of heterogeneous iot devices. *ACM Transaction on Privacy and Security*, Submitted.

[2] Abdullah Al-Tariq, Abu Raihan Mostofa Kamal, et al. A scalable framework for protecting user identity and access pattern in untrusted web server using forward secrecy, public key encryption and bloom filter. *Concurrency and Computation: Practice and Experience*, 29(23), 2017.