A Portable and Improved Implementation of the Diffie-Hellman Protocol for Wireless Sensor Networks

by

Naveed Shoaib

Submitted in Partial Fulfillment of the Requirements

for the degree of

Master of Science

in the

Mathematics

Program

YOUNGSTOWN STATE UNIVERSITY

August, 2009

A Portable and Improved Implementation of the Diffie-Hellman Protocol for Wireless Sensor Networks

Naveed Shoaib

I hereby release this thesis to the public. I understand that this thesis will be made available from the OhioLINK ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Signature:

Naveed Shoaib, Student Date

Approvals:

Dr. Graciela Perera, Thesis Advisor Date

Dr. John Sullins, Committee Member Date

Dr. Jamal Tartir, Committee Member Date

Peter J. Kasvinsky, Dean of School of Graduate Studies & Research Date

ABSTRACT

Wireless sensor nodes generally face serious limitations in terms of computational power, energy supply, and network bandwidth. One of the biggest challenges faced by researches today is to provide effective and secure techniques for establishing cryptographic keys between wireless sensor networks. Public-key algorithms (such as the Diffie-Hellman key-exchange protocol) generally have high energy requirements because they require computational expensive operations. So far, due to the limited computation power of the wireless sensor devices, the Diffie-Hellman protocol is considered to be beyond the capabilities of today's sensor networks. We analyzed existing methods of implementing Diffie-Hellman and proposed a new improved method of implementing the Diffie-Hellman key-exchange protocol for establishing secure keys between wireless sensor nodes. We also provide an easy-to-use implementation of the Elliptic Curve Diffie-Hellman key-exchange protocol for use in wireless sensor networks.

ACKNOWLEDGEMENTS

First and foremost I offer my sincerest gratitude to my supervisor, Dr. Graciela Perera, for her continuous encouragement, guidance and support during this work. I thank my other committee members, Dr. John Sullins and Dr. Jamal Tartir, for their helpful suggestions and comments.

I thank Dr. Neil Flowers for helping me understanding some of the underlying mathematics behind elliptic curve cryptography. I also thank Tony Noe for providing me with the *safe* primes list. Finally I thank my family and friends for supporting me throughout the course of my studies.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's world, the number of users in the Internet grows fast. It is made of billions of interconnected computers. However, the very nature of the network is changing. New types of devices are connecting to the net every day. Wireless sensor nodes can be imagined as small computers, extremely basic in terms of their interfaces and their components. They are typically used for monitoring, tracking, and controlling in information systems. Wireless sensor networks are, in general, more vulnerable to attacks and unauthorized access than traditional (wired) networks. They are often deployed in hostile environments where they can be easily captured, compromised, or manipulated by an adversary. So effective protection mechanisms are required and security must be provided at every aspect of the design of a wireless sensor network. In [1], Boyle and Newe discuss popular and progressive security architectures available and used to-date in wireless sensor networks. However, as demonstrated in [1] the situation is still uncertain. There is still a lot of research going on to find out the best method of providing security and authentication in wireless sensor networks.

1.1 Motivation

The problem of establishing secure keys across a traditional (wired) network has been well-studied by researchers over several decades and they have proposed a variety of key-exchange protocols. Why can't the same key-establishment protocols be used in wireless sensor networks? The inherent properties of sensor networks render previous

protocols impractical. Therefore, the implementation of effective and secure techniques for establishing cryptographic keys between wireless sensor nodes is a challenging task.

Most of the security protocols used in wireless sensor networks up-to-date rely on symmetric key cryptography because public-key cryptosystems are considered inefficient due to their high energy requirements. Active research on how to make public-key cryptosystems efficient on low-end devices is still ongoing.

The Diffie-Hellman key-exchange protocol allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. The main advantage of using the Diffie-Hellman key-exchange protocol is that a node can set up a secure key with any other node in the network. It allows two entities to directly establish a key by exchanging messages over an insecure communication channel.

1.2 Contributions

The contributions of this thesis are the development of an improved and portable Diffie-Hellman key-exchange protocol for wireless sensor networks. The specific contributions are:

- The design of a new method for establishing secure keys in wireless sensor networks called the Portable Diffie-Hellman (PDH).
- The development and implementation of a class library showing the mathematical concepts behind the PDH protocol and the secret key generation process. The library is called Simplified PDH (S-PDH). It is developed in Java so that it can be used on all platforms.

- The design of a new method for implementing the Elliptic-Curve Diffie-Hellman key-exchange protocol. It is a variant of the PDH protocol called the Portable Diffie-Hellman using Elliptic Curves (PDH-EC).
- The development and implementation of a simple class library showing the basic functionality of an elliptic curve cryptosystem along with the process of secret key generation behind the PDH-EC protocol. The library is called Simplified PDH-EC (S-PDH-EC) and is also developed in Java.
- All libraries were developed for educational purposes so that students have a practical tool to experiment and learn the mathematics behind the Diffie-Hellman and the Elliptic curve Diffie-Hellman key-exchange protocols. The libraries support the use of small primes so that anyone can get the basic understanding of the protocols. The S-PDH-EC library can be particularly helpful for someone who is new to elliptic curve cryptography.

1.3 Organization of this Thesis

The remainder of this thesis is organized as follows:

- Chapter 2 provides the background information about wireless sensor networks and important security issues involving wireless sensor networks, public-key cryptography, and the Diffie-Hellman key-exchange protocol, elliptic curve cryptography, and the Elliptic Curve Diffie-Hellman key-exchange protocol.
- Chapter 3 provides an analysis and overview of the existing methods of implementing the Diffie-Hellman key-exchange protocol. Afterwards, the design, implementation and evaluation of the Portable Diffie-Hellman (PDH) protocol are described. The implementation is done using the Java programming language.

- Chapter 4 presents the design, implementation and evaluation of the Portable Diffie-Hellman using Elliptic Curves (PDH-EC) protocol.
- Chapter 5 describes the summary of our work, conclusions and future work.

Chapter 2

Background

This chapter provides a general description of wireless sensor networks and the security aspects regarding distributed key-exchange in wireless sensor networks. In particular, we discuss the Sun SPOTS; wireless computing devices used to study the portability of the Diffie-Hellman key-exchange protocol. We also discuss the basic concepts behind public-key cryptography, the Diffie-Hellman key-exchange, elliptic curve cryptography and the elliptic curve Diffie-Hellman key-exchange protocol.

2.1 Wireless Sensor Networks

A wireless sensor network (WSN) consists of a number of independent nodes that communicate with each other wirelessly over limited frequency and bandwidth. The nodes are equipped with limited capabilities of sensing, computation and communication. Wireless sensor nodes are densely deployed and coordinate with each other to produce high-quality information about the sensing environment. The exact location of a particular sensor is unknown. It means that sensor network protocols and algorithms must provide self-organizing capabilities [2].

Figure 2.1 shows a typical wireless sensor network in which the sensor nodes are scattered in a sensor field. Each node has the capability of collecting data and routing it back to the base station. The base station is like a gateway to the wireless sensor network. It connects the complete wireless network with the task manager node via the Internet or satellite. The sensor nodes are also capable of sending information to other sensor nodes in the sensor field.

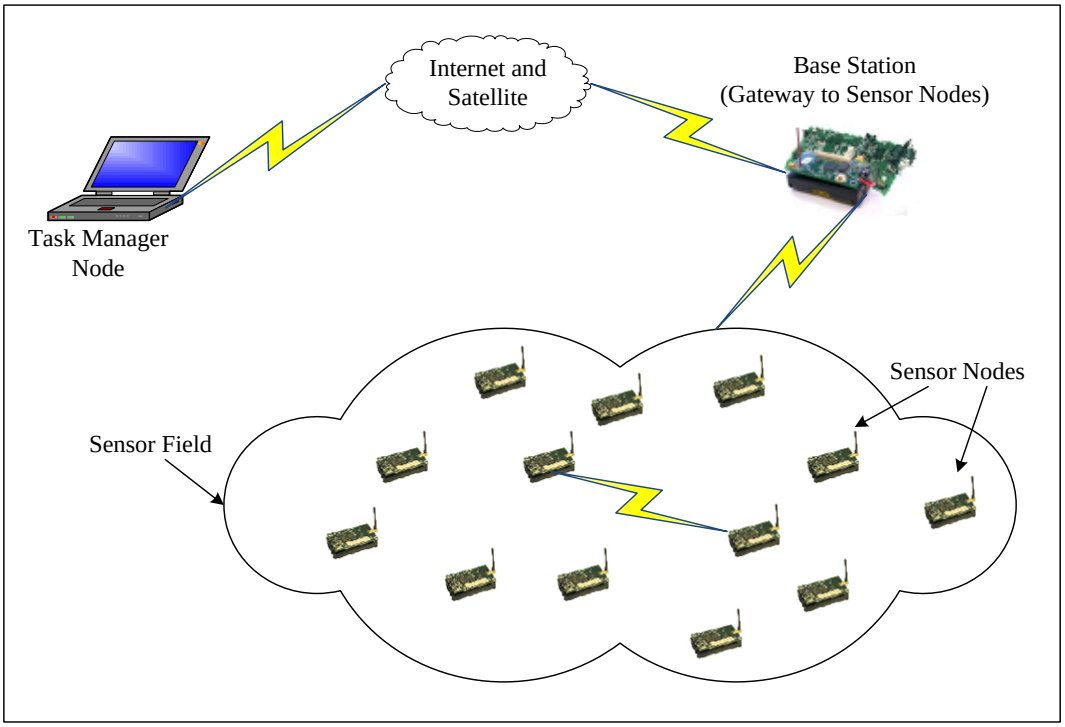

Figure 2.1: Wireless Sensor Network

There are many different applications of wireless sensor networks but typically involve some kind of monitoring, tracking, and controlling. The two most important security-oriented applications of wireless sensor networks include their use in military and medical solutions. For example, in military situations, sensor networks can be used in surveillance missions and can be used to detect moving enemy targets. Most applications of wireless sensor networks require protection against eavesdropping, tampering of data and devices, denial-of-service attacks, secure routing and node capture [1].

2.1.1 Important Security Issues

Key management and distribution are two essential security services that must be provided for a secure wireless sensor network. The establishment of a shared secret key between a pair of wireless sensor nodes is the basis for other security services such as

symmetric or private-key encryption [3]. There are many key establishment techniques designed by researchers to address the tradeoff between limited memory and security [4-5]. However, which scheme is the most effective is still debatable. The most common approach for key establishment is a network-wide key shared by all nodes or a set of keys that use random-key pre-distribution protocols [3]. These protocols are easy to implement and entail little overhead since no complex key agreement has to be performed. However, most protocols based on pre-deployed keys have disadvantages regarding scalability and vulnerability to node capture [6].

Another approach towards a secure wireless network is bootstrapping keys using a trusted base station. In this approach each node shares only a single key with the base station and set up keys with other nodes through the base station. However, if the base station fails, then the wireless sensor nodes are not able to establish any keys [3].

Public-key cryptography can solve the problem of key distribution since the communicating nodes do not need to secretly share a key in advance of their communication. It allows two nodes to communicate secretly even if all communication between them is monitored. However, pubic-key cryptography is so far considered to be beyond the capabilities of today's wireless sensor networks. The high energy requirements of public key-exchange protocols make them infeasible for use in wireless sensor networks. Active research on how to make public-key cryptography more efficient for wireless sensor networks is still ongoing [3].

In [5], Xiao et al. determined that no key distribution technique is ideal to all the scenarios where sensor networks are used. A wireless sensor network used in a battlefield

requires more security than others. Therefore, the key distribution and management techniques employed must depend upon the requirements of target applications and resources of each individual sensor network.

Authentication against eavesdropping, injection and modification of packets is another important aspect of security in wireless sensor networks. The trend has moved from pre-deployed keying mechanisms to elliptic curve cryptography (ECC) algorithms for performing authentication in wireless sensor networks [1].

Another security issue in wireless sensor networks is how to protect the node from physical capturing and cloning. Physical security is often taken for granted in traditional computing. However, wireless sensor nodes may be placed in hostile environments where they can be easily accessible to attackers. Becher et al. [7] and Xing et al. [8] discuss effective detection mechanisms against node capturing and cloning.

Now, because wireless sensor nodes differ in hardware and software, we will describe in the next section a wireless computing device called Sun SPOT which was used in our study of the portability of the Diffie-Hellman key-exchange protocol.

2.1.2 Sun SPOT

Sun SPOT (Sun Small Programmable Object Technology) [9] developed by Sun Microsystems is a battery powered, wireless computing device for use in sensor networks. It is an experimental technology used to study and develop prototype systems based on wireless communication. A detailed description of the hardware and software specifications is explained below. According to [9] the Sun SPOT has the following hardware properties:

Processing Capabilities

- 180 MHz 32 bit ARM920T core - 512K RAM - 4M Flash.
- 2.4 GHz IEEE 802.15.4 radio with integrated antenna.
- AT91 timer chip.
- USB interface.

Demo Sensor Board

- 8 tri-color LEDs.
- 2G/6G 3-axis accelerometer.
- Light and temperature sensors.
- 6 analog inputs.
- 2 push buttons.
- 5 general purpose I/O pins and 4 high current output pins.

Battery

- 3.7V rechargeable 750 mAh lithium-ion battery.
- 30 uA deep sleep mode.
- Automatic battery management provided by the software.

Radio Communication

- The devices communicate using the IEEE 802.15.4 standard including the base-station approach to sensor networking.

The Sun SPOT uses a small J2ME-level VM called Squawk [9] which runs directly on the processor without an OS. The development tools are:

- Standard Java IDEs (e.g. NetBeans) can be used to create SunSPOT applications.

- "SPOTWorld" is an application that can be used for management, programming, debugging and deployment of applications on the Sun SPOT.

The next section provides a discussion on public-key cryptography.

2.2 Public-Key Cryptography

Symmetric-key cryptography requires the sender and receiver to share a common secret key. One of main drawbacks of symmetric-key cryptography is how the sender and receiver exchange the secret key in the first place. The secret key cannot simply be sent over an insecure communication channel where the key can be compromised. The authors of [10] and [11] discuss several different approaches to secret-key distribution. Key distribution centers and protocols such as Kerberos are very popular and can be used to solve the issue of secret-key distribution. However, when we talk about the Internet and wireless networks, the protocols used in these networks lack a secure channel to exchange secret keys. Therefore, the problem of key distribution is a current area of research. Nevertheless, until 1976, it was believed that encryption simply cannot be done without first sharing a secret key. In 1976, Diffie and Hellman [12] proposed a new type of cryptography known as asymmetric or public-key cryptography. Public-key cryptography uses two keys instead of one; one of these keys is used by the sender of the message for encryption, called the *public* key, and the other one is used by the receiver of the message for decryption, called the *private* key.

Public-key cryptography uses mathematical functions for encryption and decryption. The main idea is to use a *trapdoor one-way function* that is easy to compute in one direction, yet believed to be difficult to compute in the opposite direction (finding its inverse)

without special information, called the "trapdoor". In mathematical terms, if f is a trapdoor function, there exists some secret information y, such that given $f(x)$ and y it is easy to compute x [10]. An example of a trapdoor one-way function is the factorization of a product of two large primes. The multiplication of two large prime numbers is easy; however, factoring the resulting product is very difficult. Therefore, many of the public-key cryptosystem base their security on the difficulty of solving mathematical problems such as the integer factorization problem, finite field discrete logarithm problem and the elliptic curve discrete logarithm problem (explained in later sections). All of these problems are believed to be both secure and practical after years of intensive studying.

One of the most important aspects of public-key algorithms such as the Diffie-Hellman key-exchange protocol is that it solves the problem of key management and distribution. A *key-exchange* protocol allows key distribution to be done over an insecure channel since the communicating parties do not need to secretly share a key in advance of their communication. In the next section we describe the Diffie-Hellman key-exchange protocol and some of the underlying mathematics behind it.

2.3 Diffie-Hellman Key Exchange

The Diffie-Hellman key-exchange protocol uses the multiplicative group of integers modulo p, $<\mathbb{Z}_{p^*}, \times>$, where p is a large prime number on the order of 300 decimal digits (1024 bits). We represent this group by $G = <\mathbb{Z}_{p^*}, \times>$. Before discussing the protocol, we first describe the important properties of the group. This group G consists of all integers from 1 to $p - 1$ where p is a prime. Following are some important properties of G:

- The *order* of the group is given by, $|G| = p - 1$.
- Every member of the group has an *additive* and a *multiplicative* inverse.
- The group is *abelian* i.e., for all $a, b \in G$, $ab \bmod p = ba \bmod p$.
- The group is *cyclic* i.e., there exists an element $g \in G$ such that:

  $$< g >= \{g^0, g^1, g^2, ..., g^{|G|-1}\} = G$$

The group and the generator do not need to be confidential. They can be sent over an insecure communication channel such as the Internet. So p and g are public. The following interchange of messages between Alice and Bob demonstrates the Diffie-Hellman key-exchange protocol:

1. Alice and Bob publicly agree on a cyclic group G, its generator g and a prime p.
2. Alice and Bob each secretly choose large random numbers a and b, such that

   $$0 \le a, b \le p - 1.$$

3. Alice calculates $R_A = g^a \bmod p$, while Bob calculates $R_B = g^b \bmod p$.
4. Alice sends R_A to Bob and Bob sends R_B to Alice.
5. Alice calculates $K = (R_B)^a \bmod p$ while Bob calculates $K = (R_A)^b \bmod p$.
6. Both get the same value for the key i.e.

   $$K = (g^a \bmod p)^b \bmod p = (g^b \bmod p)^a \bmod p = g^{ab} \bmod p$$

$K = g^{ab} \bmod p$ is called the Diffie-Hellman *secret* key.

2.3.1 Security of Diffie-Hellman

The security of the Diffie-Hellman key-exchange protocol relies on the presumed hardness of the discrete logarithm problem (DLP) in a group of large order, i.e., computing the Diffie-Hellman secret key is considered computationally impossible given

the public parameters. At the end of the protocol, the values g^a and g^b have become public while the value of g^{ab} remains private. Thus, the *Diffie-Hellman Problem* (DHP) is to compute g^{ab} from the given values of g^a and g^b. This is widely believed to be difficult as long as the discrete logarithm problem has not been solved in *G* [10].

One of shortcomings of the Diffie-Hellman key-exchange protocol is the *Man-in-the-Middle* attack. In this kind of attack an eavesdropper intercepts all messages between Alice and Bob and makes independent connections with them. The eavesdropper can then replay messages between Alice and Bob, making them believe that they are talking directly to each other over a private connection when in fact the entire conversation is controlled by the eavesdropper. To thwart this kind of an attack, the *Station-to-Station* key agreement protocol can be used. The protocol is based on Diffie-Hellman and provides authentication. It uses digital signatures with public key certificates to establish a secure session key between Alice and Bob [10]. To provide authentication, a variant of the Diffie-Hellman protocol called HMQV [13] can also be used.

A more recent method used in public-key cryptography to generate keys is to use elliptic curves and is described in the next section.

2.4 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. In 1985, Neal Koblitz [14] and Victor S. Miller [15] independently proposed the use of elliptic curves in cryptography. ECC is emerging as an attractive public-key cryptosystem for wireless sensor networks. It provides an alternative to established public-key systems such as the

DSA (Digital Signature Algorithm) and RSA (Rivest-Shamir-Adleman) algorithms. In recent years, ECC has gained a lot of attention. The main reason for the attractiveness of ECC is the fact that there is no sub-exponential algorithm known to solve the discrete logarithm problem on a properly chosen elliptic curve [16]. This means that significantly smaller parameters can be used in ECC (with the same level of security) than in other competitive systems [17]. This makes ECC ideal for wireless sensor networks which are typically limited in terms of their CPU power, memory and network connectivity.

2.4.1 Mathematical Foundations of ECC

Much of the following discussion in this section is based on material presented in Lawrence C. Washington's book "Elliptic Curves, Number Theory and Cryptography" [18].

The mathematical operations of ECC are defined over a special class of elliptic curve of the form:

$$y^2 = x^3 + Ax + B \bmod p \qquad \longrightarrow (2.1)$$

Where $A, B \in \mathbb{Z}_p$ are constants satisfying the condition: $4A^3 + 27B^2 \neq 0 \bmod p$. This condition ensures that the equation, $x^3 + Ax + B = 0 \bmod p$ has no repeated roots (nonsingular). The modulo p is a prime with $p > 3$. The theory can be adopted to deal with the case of $p = 2$ or 3. However, the discussion presented here deals only with the field of characteristics not equal to 2 or 3.

Let $E_p(A, B)$ denote the set of points $P = (x, y)$ that satisfy equation (2.1), i.e.

$E_p(A, B) = \{(x, y) \mid (x, y) \in \mathbb{Z}_p$ and $y^2 = x^3 + Ax + B \bmod p\}$

We define the set $E(\mathbb{Z}_p)$ as follows:

$E(\mathbb{Z}_p) = E_p(A, B) \cup \{\boldsymbol{O}\}$

The elements of the set $E(\mathbb{Z}_p)$ are called the *points on the elliptic curve E* defined by equation (2.1) together with an extra point $\boldsymbol{O}$ which is called the *point at infinity*. The specific properties of a nonsingular elliptic curve allows us to define a binary operation, called "addition" (denoted by '+'), on the points of $E(\mathbb{Z}_p)$. The operation is the addition of two points on the curve to get another point on the curve.

$R = P + Q$ where $P = (x_1, y_1)$, $Q = (x_2, y_2)$ and $R = (x_3, y_3)$.

The point $\boldsymbol{O}$ is defined as an (additive) identity i.e., for all $P \in E(\mathbb{Z}_p)$, $P + \boldsymbol{O} = \boldsymbol{O} + P = P$.

It can be shown that every line intersecting the curve E intersects the curve in exactly three points, where:

1. a point P is counted twice if the line is tangent to the curve at P, and
2. the point at infinity is also counted (when the line is vertical).

2.4.2 ECC Arithmetic

The rules for negation, addition and point doubling are described below. However, to conceptualize the basic arithmetic behind $E(\mathbb{Z}_p)$ we will first give a graphical explanation of elliptic curves over the reals i.e. the equation $y^2 = x^3 + Ax + B \mod p$ without reduction modulo p. This is because; in modular arithmetic the points on the curve do not make nice graphs. Nevertheless, the concept remains the same.

Negation

A negative of a point is the reflection of that point with respect to x-axis. Given a point P, its negation $-P$ is the point for which $P + (-P) = \boldsymbol{O}$. The line connecting the two points

intersects the curve at ***O***. We can think of the "point at infinity" ***O*** as sitting at the top of the *y*-axis and lying on every vertical line. As shown in Figure 2.2 (b), $-P$ is simply the reflection of P in the *x*-axis, that is, if $P = (x_1, y_1)$ then $-P = (x_1, -y_1)$.

Point Addition

For an elliptic curve *E*, take two arbitrary points $P, Q \neq \boldsymbol{O}$. Point addition is the process of adding these points to obtain another point *R* on the same elliptic curve. If $Q \neq -P$, as in Figure 2.2 (a), then the line drawn through the points *P* and *Q* will intersect the elliptic curve at exactly one more point, $-R$. If $P = Q$, then draw the line tangent to *E* at *P* (see point doubling). Graphically, $P + Q$ can be found by reflecting the point $-R$ with respect to the *x*-axis.

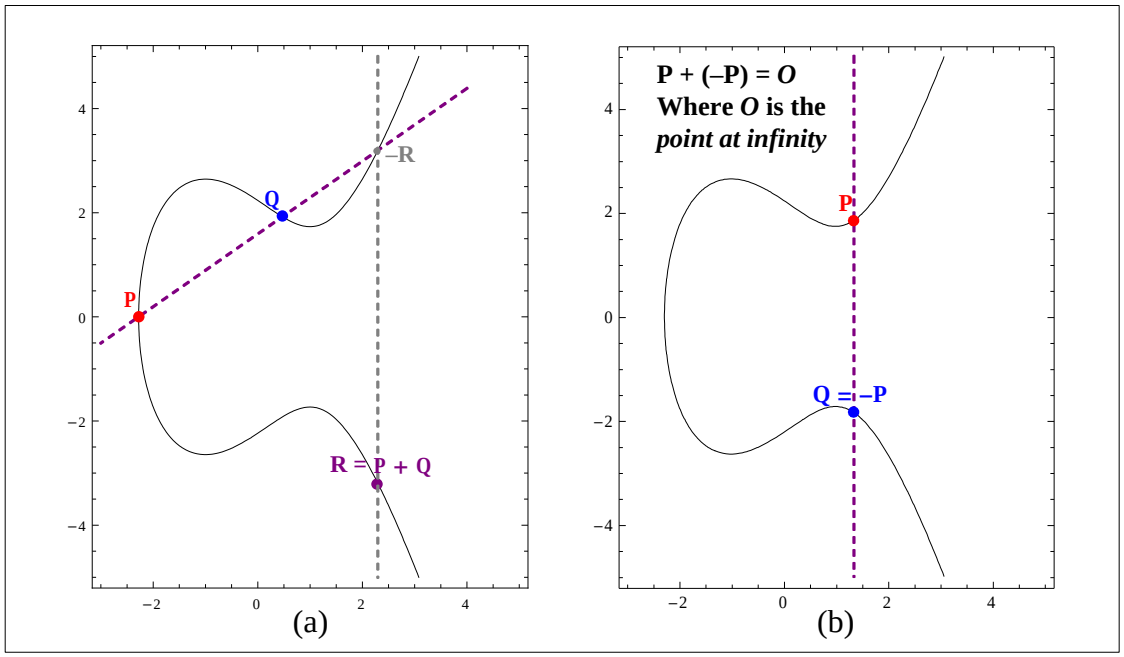

Figure 2.2: Point Addition

In $E(\mathbb{Z}_p)$, we use the same addition operation but the calculations are done modulo *p*. So let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points in $E(\mathbb{Z}_p)$ with $P, Q \neq \boldsymbol{O}$**.** We first consider the case where *P* and *Q* have different *x* and *y* coordinates ($x_1 \neq x_2$ and $y_1 \neq y_2$). Then

the coordinates of the point R, x_3 and y_3 can be found by first finding the slope of the line m, through P and Q, and then calculating the values of x_3 and y_3 as shown below:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \bmod p$$

$$x_3 = m^2 - x_1 - x_2 \bmod p$$

$$y_3 = m(x_1 - x_3) - y_1 \bmod p$$

Our assumption, that $x_1 \neq x_2$ means that $x_1 - x_2 \neq 0 \bmod p$. This implies that the inverse of $x_2 - x_1$ modulo p exists. However, if $Q = -P$ ($x_1 = x_2$ and $y_1 = -y_2$), as shown in Figure 2.2 (b), then the two points are additive inverses of each other. As already mentioned, then $P + (-P) = \boldsymbol{O}$.

Point Doubling

For an elliptic curve E, take an arbitrary point $P \neq \boldsymbol{O}$. Point doubling is the process of adding the point P to itself to obtain another point R on the same elliptic curve. If the y-coordinate of point P is not zero as in Figure 2.3 (a), then the tangent line drawn at P will intersect the elliptic curve at exactly one more point, $-R$. Graphically, $2P$ can be found by reflecting the point $-R$ with respect to the x-axis.

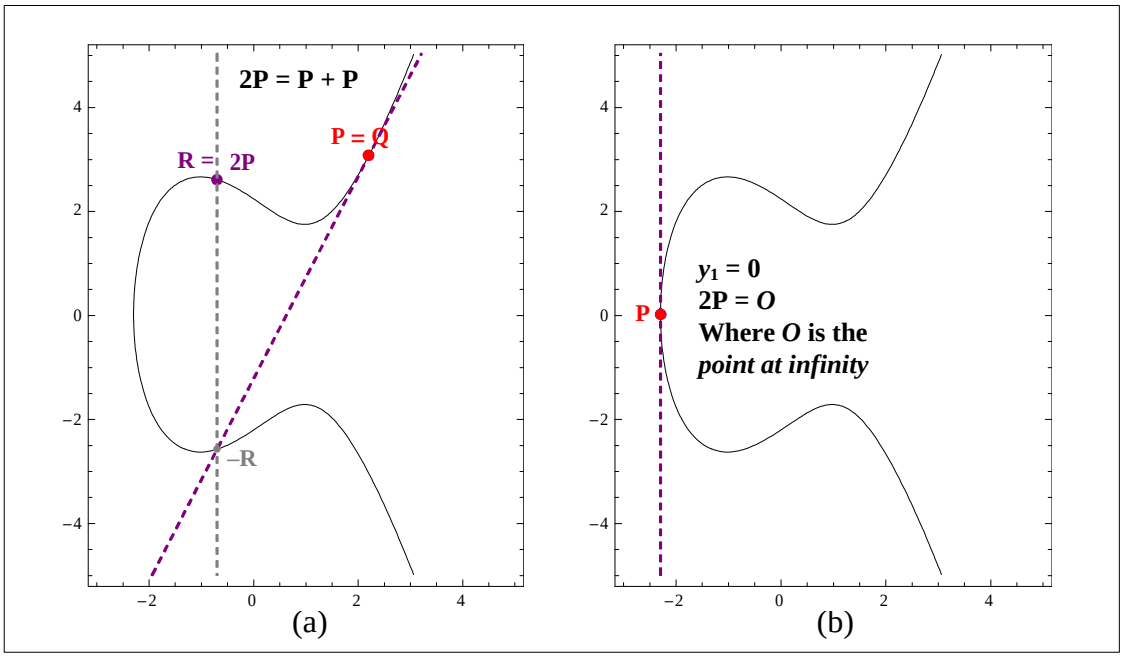

Figure 2.3: Point Doubling

Consider a point $P = (x_1, y_1)$ be in $E(\mathbb{Z}_p)$ with $P \neq \boldsymbol{O}$. We first consider the case where the y-coordinate of P is not zero ($y_1 \neq 0$). In this case, the slope of the line and the coordinates of the point R, x_3 and y_3 can be found as shown below:

$$m = \frac{3x_1^2 - A}{2y_1} \bmod p$$

$$x_3 = m^2 - 2x_1 \bmod p$$

$$y_3 = m(x_1 - x_3) - y_1 \bmod p$$

If the y-coordinate of the point P is zero as shown in Figure 2.3 (b), then the tangent at this point intersects the curve at $\boldsymbol{O}$, So, $2P = \boldsymbol{O}$.

2.4.3 Properties of the Elliptic Curve

It is shown in [18] that the sets of points $E(\mathbb{Z}_p)$ along with the addition rule defined above form an abelian group:

- *Closure*: Adding two points on the curve creates another point on the curve.
- *Associatively*: $(P + Q) + R = P + (Q + R)$.
- *Commutatively*: $P + Q = Q + P$.
- *Identity*: The "point at infinity" $\boldsymbol{O}$ is the additive identity. In other words $P = P + O = O + P$.
- *Inverse*: Every point on the curve has an inverse. The inverse of a point is its reflection with respect to *x*-axis. In other words, the point $P = (x_1, y_1)$ and $Q = (x_1, -y_1)$ are inverses of each other, which means that $P + Q = O$. Note that the identity element is the inverse of itself.

2.4.4 Finding Points on the Elliptic Curve

Recall from above that $E_p(A, B)$ denote the set of points $P = (x, y)$ that satisfy equation (2.1), i.e. $E_p(A, B) = \{(x, y) \mid (x, y) \in \mathbb{Z}_p$ and $y^2 = x^3 + Ax + B \mod p\}$.

It is shown in [18] that there can only be finitely many points that can satisfy the above equation. Therefore, the points on the curve $E(\mathbb{Z}_p)$ form a finite abelian group. To generate a point on the elliptic curve we choose an $x \in \mathbb{Z}_p$ and check to see if there is a corresponding *y* satisfying the elliptic curve equation. Let $f(x) = x^3 + Ax + B$ and consider the curve $y^2 = f(x) \mod p$. This means that we have to evaluate whether $f(x)$ is a quadratic residue modulo *p*.

An element $z \in \mathbb{Z}_{p^*}$ is a quadratic residue modulo *p* if it is congruent to a perfect square (mod *p*), i.e., if there exists a $y \in \mathbb{Z}_{p^*}$ such that $y^2 \equiv z \mod p$. Modulo an odd prime number *p* there are $(p + 1)/2$ quadratic residues (including 0) and $(p - 1)/2$ quadratic non-

residues. In this case, it is customary to consider 0 as a special case and work within the multiplicative group of nonzero elements $\mathbb{Z}_{p^*}$.

Now depending on whether $f(x)$ is a quadratic residue or not modulo *p*, we can have the following cases:

- $f(x)$ is a quadratic residue, then there are two points on the curve $(x,\pm y)$.
- $f(x)$ is a non-quadratic residue, then there is no point on the curve.
- $f(x) = 0 \bmod p$, then there is a single point on the curve $(x, 0)$.

This allows us to determine the points on the curve.

2.4.5 Order of an Elliptic Curve

The order of an elliptic curve is defined as the number of points on the curve and is denoted by $|E(\mathbb{Z}_p)|$. The simplest way of finding out the number of points on the curve is to use the following equation, the proof of which can be found in [18].

$$|E(\mathbb{Z}_p)| = p + 1 + \sum_{x \in Z_p} \left(\frac{x^2 + Ax + B}{p} \right)$$

Where $\left(\frac{x^2 + Ax + B}{p} \right)$ can be evaluated by using "Legendre" symbol. Given an odd prime *p* and an integer *a*, then the **Legendre symbol** is defined as follows:

$$\left(\frac{a}{p} \right) = \begin{cases} +1 & \text{if } a \text{ is a quadratic residue modulo } p. \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p. \\ 0 & \text{if } p \text{ divdes } a. \end{cases}$$

The above method is considered insufficient when *p* is extremely large. So a different point counting algorithm, such as the Schoof's algorithm or Schoof-Elkies-Atkin algorithm [19-20] must be used.

2.4.6 Scalar Point Multiplication

Scalar point multiplication is the core of elliptic curve arithmetic. It is the most essential part of a secure elliptic curve cryptosystem. For an elliptic curve E, take an arbitrary point $P \neq \boldsymbol{O}$. Scalar multiplication is the process of adding the point P to itself k times to obtain another point Q on the same elliptic curve.

$$Q = kP = \underbrace{P + P + \ldots + P}_{k}$$

Where $k < |E(\mathbb{Z}_p)|$ is a scalar. Scalar multiplication of a point on E can be performed through a combination of point additions and point doublings, e.g. $11P = 2((2(2P)) + P) + P$.

The above method is called the *"double and add"* algorithm for scalar point multiplication. However, there are a number of other efficient algorithms also available for scalar point multiplications [21].

2.4.7 Elliptic Curve Discrete Logarithm Problem

Much of today's ECC is based on the elliptic curve discrete logarithm problem (ECDLP). Recall from the last section that scalar multiplication is the core of elliptic curve arithmetic. When the elliptic curve group is described using additive notation, the elliptic curve discrete logarithm problem is defined as follows:

"Given the points kP and Q in the group, find the value of k such that $kP = Q$".

The problem is considered computationally difficult unless the curve is "weak". Several classes of curves are weak and must be avoided e.g. example, if $|E(\mathbb{Z}_p)| = p$, then the

curve is vulnerable to attack [22-23]. It is because of these issues that point-counting on elliptic curves is such a hot topic in ECC.

2.4.8 ECC Domain Parameters

To use ECC all parties involved in the communication must agree on all the elements defining the elliptic curve *E* i.e., the domain parameters of the scheme. The domain parameters for the elliptic curve $E(\mathbb{Z}_p)$ are *p*, *A*, *B*, *G*, *n* and *h*. Following is a brief description of each:

- *p* is a prime such that $p > 3$.
- *A* and *B* are the parameters defining the elliptic curve equation.
- *G* is the called the *base* point or the *generator* point. Each curve has a specially designated point, *G*, chosen such that a large fraction of the elliptic curve points are multiples of it. We call <*G*> the cyclic subgroup generated by *G*. Scalar point multiplication can be used for finding the multiples of *G*. The scalar for point multiplication is chosen such that it is a number between 0 and $n - 1$.
- *n* is the order of *G*, i.e. the smallest non-negative integer *n* such that $n\mathrm{G} = \boldsymbol{O}$.
- *h* is called the *cofactor* where $h = \frac{|E(\mathrm{Z}_p)|}{n}$. Since *n* is the size of the cyclic subgroup generated by *G*, it follows from Lagrange's theorem that the order of the subgroup must divide the order of the group. So *h* is an integer. For cryptography applications *h* must be small $(h \leq 4)$.

2.5 Elliptic Curve Diffie-Hellman (ECDH)

The Elliptic Curve Diffie-Hellman key-agreement protocol is a variant of the Diffie-Hellman protocol using elliptic curve cryptography. It allows two parties, each having an elliptic curve public-private key pair, to establish a shared secret over an insecure communication channel. Suppose Alice wants to establish a secret key with Bob. The following interchange between Alice and Bob demonstrates the Elliptic Curve Diffie-Hellman key-exchange protocol:

1. Alice and Bob publicly agree on an elliptic curve $E(\mathbb{Z}_p)$ and all the domain parameters, i.e., (p, A, B, G, n, h).
2. Alice and Bob each secretly choose large random integers, a and b, such that $0 < a, b < n$.
3. Using elliptic curve scalar point multiplication, Alice calculates $G_A = aG$ while Bob calculates $G_B = bG$ on E.
4. Alice sends G_A to Bob and Bob sends G_B to Alice.
5. Alice calculates $aG_B = abG$ while Bob calculates $bG_A = baG$.
6. Both get the same value for the key.
7. Alice and Bob use some publicly agreed on method to extract a secret key from abG. For example, they could use the x-coordinate of this point as the secret key.

The ECDH protocol is secure because no one can derive the private key of the other unless one can solve the Elliptic Curve Discrete Logarithm Problem. However, the protocol does not provide authentication. If authentication is required then protocols such as ECMQV can be used.

In the next chapter we provide our analysis of the Diffie-Hellman key-exchange protocol and then describe the new improved and portable Diffie-Hellman key-exchange protocol for use in wireless sensor networks.

Chapter 3

Portable Diffie-Hellman

In this chapter we first provide our analysis of the Diffie-Hellman key-exchange protocol and develop an improved key-exchange protocol for wireless sensor networks called Portable Diffie-Hellman (PDH). Afterwards we give a detailed description of the PDH protocol. Then we describe and discuss the implementation of a Simplified PDH (S-PDH) library designed to show the mathematical concepts and the key generation process behind the PDH protocol. The last section presents the evaluation of the S-PDH library.

3.1 Analysis of the Diffie-Hellman Key-Exchange Protocol

We begin our analysis by looking at the public parameters of the Diffie-Hellman key-exchange protocol (i.e., the prime number p and the generator g). To make Diffie-Hellman secure from the discrete logarithm attack, the following are recommended:

- p must be more than 300 decimal digits (1024 bits).
- p must be chosen such that $p - 1$ has at least one large prime factor of more than 60 decimal digits.
- g must be chosen from the group $<\mathbb{Z}_{p^*}, \times>$.
- a and b must be more than 100 decimal digits.

All of the above recommendations mean that the Diffie-Hellman key-exchange protocol has an expensive computational cost specifically in computing the public parameters p and g. During our investigation of the practical implementations of the Diffie-Hellman key-exchange protocol, we encountered that the parties involved in the communication publicly agree on the public parameters. Several standard bodies such as the National

Institute of Standards and Technology (NIST) calculate these parameters in advance and later publish them for use in cryptographic applications. So most of the time the same parameters are used over and over again.

However, using prime numbers on the order of 300 decimal digits is not convenient for all practical implementations, due to the fact that embedded systems (such as the wireless sensor networks) have low computing power. Finding a generator *g* for such groups is also computationally very expensive. The high energy requirements from computational expensive operations have raised serious concerns about the protocol's feasibility in wireless sensor networks.

In particular, Diffie-Hellman key-exchange protocol is characterized by high energy consumption for calculating cryptographic primitives, but relatively low communication energy cost. Therefore, to make the protocol feasible for wireless sensor networks, especially Sun SPOTS, we have to reduce the computational costs of the protocol, without jeopardizing the security of the protocol.

The main difficultly in implementing the Diffie-Hellman key-exchange protocol in wireless sensor networks is to tie the underlying mathematics behind it with its practical use. The implementer must understand the exact mathematical concepts on which the protocol is based. That may not often be the case if the implementer is just replying on pre-generated public parameters without having a full understanding of the mathematical concepts.

So we reviewed the relationship between the practical implementation and mathematical foundations on which the Diffie-Hellman key-exchange protocol was based upon. We

also provided a simple practical implementation of the Diffie-Hellman key-exchange protocol that shows the basic principles of the protocol.

Next we look at the mathematical foundations of the public parameters of the Diffie-Hellman key-exchange protocol.

3.1.1 Using Prime Order Groups

The theory discussed here is inspired from the ideas presented in Jonathan Katz's and Yehuda Lindell's book "Introduction to Modern Cryptography" [11]. The theory leads us to the development of an improved and Portable Diffie-Hellman key-exchange protocol for use in wireless sensor networks. We start our discussion with an important concept from group theory:

"If *G* is a group of prime order, then *G* is cyclic. Furthermore all elements of *G* except the identity are generators of *G*"

Consider the group $(\mathbb{Z}_p, +)$ which contains all integers from 0 to $p - 1$. The group has prime order. All elements except 0 (identity) are generators of the group.

For cryptographic applications, there is a general preference for using certain cyclic groups of prime order. The main reasons behind this are:

- The discrete logarithm and the Diffie-Hellman problems are believed to be the hardest in certain cyclic groups of prime order e.g. $(\mathbb{Z}_{p^*}, \times)$. Note that the additive group of integers modulo p $(\mathbb{Z}_p, +)$ is a cyclic group of prime order, however, the group is not used for the Diffie-Hellman key-exchange protocol because the Diffie-Hellman problem is believed to be easy in this group. That's why we only

use the multiplicative group of integers modulo p ($\mathbb{Z}_{p^*}$, $\times$) for the Diffie-Hellman key-exchange protocol.

- Finding a generator for a cyclic group of prime order is trivial.
- In a cyclic group of prime order, testing whether a given element is a generator is also trivial. So for a given group G of prime order, we can use as many generators as we want.

Now the group ($\mathbb{Z}_{p^*}$, $\times$) is cyclic. However for $p > 3$ prime, the group does not have prime order. So finding generators for such groups may take longer than groups of prime order and as discussed there is a preference of using groups of prime order. Also, the Diffie-Hellman problem is simply not hard in groups that do not have prime order. The solution to this problem is to work in subgroups of $\mathbb{Z}_{p^*}$.

3.1.2 Using Subgroups of $\mathbb{Z}_{p^*}$

As discussed in chapter 2, an element $z \in \mathbb{Z}_{p^*}$ is a quadratic residue modulo p if it is congruent to a perfect square modulo p (i.e., if there exists a $y \in \mathbb{Z}_{p^*}$ such that $y^2 \equiv z \mod p$). It can be proved that the set of quadratic residues modulo p forms a subgroup of $\mathbb{Z}_{p^*}$. Moreover, modulo an odd prime number p, half of the elements of $\mathbb{Z}_{p^*}$ are quadratic residues. Excluding 0 (which is considered as a special case), the order of the subgroup of quadratic resides modulo p is $(p - 1)/2 = q$. We want q to be a prime number. For that we first introduce the concept of *safe* primes.

A *safe* prime p is a prime number of the form $p = 2q + 1$, where q is also a prime. The first ten safe primes are 5, 7, 11, 23, 47, 59, 83, 107, 167, 179 & 227. These primes are called *safe* because of their relationship to *strong* primes. A prime number p is a *strong*

prime if $p + 1$ and $p - 1$ both have large prime factors. For the Diffie-Hellman key-exchange protocol it is desirable to use *strong* primes.

Now if p is a *strong* prime of the form $p = 2q + 1$ where q is also prime, then the subgroup of quadratic residues has exactly $(p - 1)/2 = q$ (prime) elements. Now since the order of the subgroup q is a prime, the subgroup is cyclic and furthermore all elements (except the identity) are generators.

From the above discussion a generator g of the cyclic subgroup of quadratic residues can easily be found by picking an arbitrary $x \in \mathbb{Z}_{p^*}$ such that $x \neq \pm 1 \mod p$ and setting $g = x^2 \mod p$.

In the next section, we describe the Portable Diffie-Hellman (PDH) key-exchange protocol for wireless sensor networks.

3.2 Portable Diffie-Hellman (PDH) Key-Exchange Protocol

Following is a brief description of the Portable Diffie-Hellman (PDH) protocol:

- The general idea behind our protocol is to use a file containing a large number of *safe* primes. This file will be downloaded on the wireless sensor nodes using a secure link (USB cable) at the time of node deployment or from a secure desktop. The file is *secret* and will not be publicized. This means that the prime number used by Alice to generate her public-key will not be sent over the insecure communication channel.

- Instead, *counters* will be used at both ends. Initially, at the time of node deployment, the counters will be set to a predetermined number. All nodes will have the same value for the counter.
- In order for the communication to happen, Alice will pick a prime using the value of the counter from the large *safe* prime file. Then, she will randomly generate a value for the generator *g* of the group. She will then calculate her public key and send it to Bob along with the value of *g*.
- On receiving Alice's public key and the value of *g*, Bob will generate his public key by picking the same prime (using the value of the counter) and the generator *g*. Bob will then send his public key to Alice. Both parties can now calculate the Diffie-Hellman secret key.
- At the end of the communication, the value of the counter will be incremented so that every time a new prime can be used. Alice and Bob will destroy their private keys after they have calculated the session key.

Next we look at the detailed description of the PDH protocol and prove its security in the presence of an eavesdropping adversary.

3.2.1 Detailed PDH

The design of PDH is based on the following definition of security. A key-exchange protocol is secure if the secret key generated by Alice and Bob is completely unknown to an eavesdropping adversary. Moreover, an adversary should not be able to generate all possible secret keys given the public parameters of the protocol. The secret key will only

be used within a private-key encryption scheme. Furthermore, the assumptions of our work are:

- Two wireless sensor nodes execute the protocol in order to establish a secure key. We refer to these nodes as Alice and Bob.
- The deployment of the protocol is over Sun SPOTS which have limited resources. As a consequence the base station will not provide authentication of the wireless sensor nodes. This assumption is made because protocols which provides authentication using digital signature schemes cannot be implemented due to the limited resources of the wireless sensor networks.
- At the time of deployment, each node will have a large file containing *safe* primes. The protocol assumes that the wireless sensor nodes will only be able to establish the *same* secret key using these primes.
- The protocol assumes that the wireless sensor network is protected against *node capture* and *node cloning* attacks. Effective detection mechanisms as discussed in [7-8] are available. Node capturing requires absence of a node from the network for a substantial amount of time (minimum five minutes). The protocol assumes that mechanisms for revocation of a node (which was absent for too long from the network) by its neighbors is present in the wireless sensor network [7]. Furthermore, in order to detect clone attackers in real-time, fingerprint verification schemes are present locally (via neighboring nodes) or globally (via the base station) [8].

We now present the details of the Portable Diffie-Hellman (PDH) key-exchange protocol for wireless sensor networks.

3.2.2 The PDH Protocol with Safe Primes

The main goal of a key-exchange protocol is to establish a secure key between the two communicating parties. The secret key is also called a *session* key as it is only used for a particular session. The session key is used to encrypt data using a private-key encryption scheme. As already mentioned, we assume that there is no central authority available that can provide authentication of public keys. So in PDH, a session key will only be created when a wireless sensor node wants to send data to another node. This means that if Alice wants to send secret data to Bob, she will start the key-exchange protocol and vice versa. Also if Alice and Bob want to send secret data to each other simultaneously, they will generate two different session keys. To send secret data, Alice will use the session key created when she initiated the key-exchange protocol while Bob will use the session key created when he initiated the key-exchange protocol.

The main purpose for having a secret large file containing *safe* primes is to randomize the process of key exchange as much as possible. This will also help in reducing the computational costs of the Diffie-Hellman key-exchange protocol.

The prime used by Alice to generate her public-key will not be sent over the insecure communication channel. The question is then, how does Bob know which prime number Alice has used? Each node is the wireless sensor network will have two counters set to a predestined value. These counters will allow nodes to determine which *safe* prime to use. The integrity of the protocol depends on keeping these counters secure.

Initially all the nodes will have the same value of the counters. The first counter is called the *initiate counter* (IC). This counter will be used by the wireless sensor node who wants

to initiate the key-exchange protocol. The second counter is called the *receive counter* (RC). This counter will be used by the wireless sensor node who will receive the first message in the key-exchange protocol. In other words, this means that if Alice wants to send secret data to Bob she will use the value of IC while Bob will use the value of RC to figure out which *safe* prime to use.

Figure 3.1 shows the working of counters. Each node in the wireless sensor network maintains a table called Node Registration and Counters (NRC). There table has three columns i.e., Node ID, IC and RC. Initially IC and RC counters are set to a predetermined value, for example, 1000. If Alice initiates a key exchange with Bob, she will first have to register Bob. After the registration, both counters are automatically set to 1000. This means that Alice will use the first prime in the *safe* prime file. Similarly, when Bob receives the first key exchange message, he will first have to register Alice. The counters are set to 1000 which allows Bob to figure out which *safe* prime Alice has used. After the key exchange has occurred, Alice will increment her IC while Bob will increment his RC. The main reason for using two counters is to avoid problems of asynchronous communications. Figure 3.1 shows that Alice has initiated the key-exchange protocol four times while Bob has initiated the key-exchange protocol two times.

Now let us suppose that a third wireless sensor node, Carol wants to communicate with Alice. As Alice is a new node with which Carol wants to communicate, she will first register Alice. When Alice receives the first message, she will add a new entry for Carol in the node registration table. After the key exchange, Carol will increment her IC while

Alice will increment her RC. Note that the counters will only be incremented when a key exchange has completely occurred and acknowledgements have been received.

Alive		
Node ID	**IC**	**RC**
Bob	1004	1002
Carol	1000	1001

Bob		
Node ID	**IC**	**RC**
Alice	1002	1004

Carol		
Node ID	**IC**	**RC**
Alice	1001	1000

Figure 3.1: Node Registration and Counters (NRC) Table

The description of the PDH assumes that Alice will initiate the protocol. So Alice generates the parameters (p, G, q, g). Following is a brief description about the parameters:

- p is a *safe* prime of the form $p = 2q + 1$, where q is also prime.
- G is the cyclic group $\mathbb{Z}_{p^*}$.
- q is order of the subgroup of quadratic residues i.e. $q = (p - 1)/2$.
- g is the generator of the subgroup of quadratic residues. g is found by picking an arbitrary $x \in \mathbb{Z}_{p^*}$ such that $x \neq \pm 1 \mod p$ and setting $g = x^2 \mod p$.

Figure 3.2 shows the process of key exchange between Alice and Bob. The steps illustrated in Figure 3.2 are described below:

1. Alice picks a *safe* prime p from the "*Safe* Prime File" using the current value of IC. She then generates the parameters (G, q, g).
2. Alice chooses a large random number $a \in \mathbb{Z}_q$ and calculates $R_A = g^a \bmod p$.
3. Alice sends (g, R_A) to Bob.
4. On receiving (g, R_A), Bob picks the same *safe* prime p (as Alice) from the "*Safe* Prime File" using the current value of RC. He then generates (G, q).
5. Bob chooses a large random number $b \in \mathbb{Z}_q$ and calculates $R_B = g^b \bmod p$.
6. Bob sends R_B to Alice and calculates $K = (R_A)^b \bmod p$.
7. Alice receives R_B and calculates $K = (R_B)^a \bmod p$.
8. Alice increments IC by one while Bob increments RC by one when acknowledgement from Alice is received.
9. Both parties have the same value for the key that is:

   $K = (g^a \bmod p)^b \bmod p = (g^b \bmod p)^a \bmod p = g^{ab} \bmod p$

$K = g^{ab} \bmod p$ is called the Potable Diffie-Hellman *secret* key.

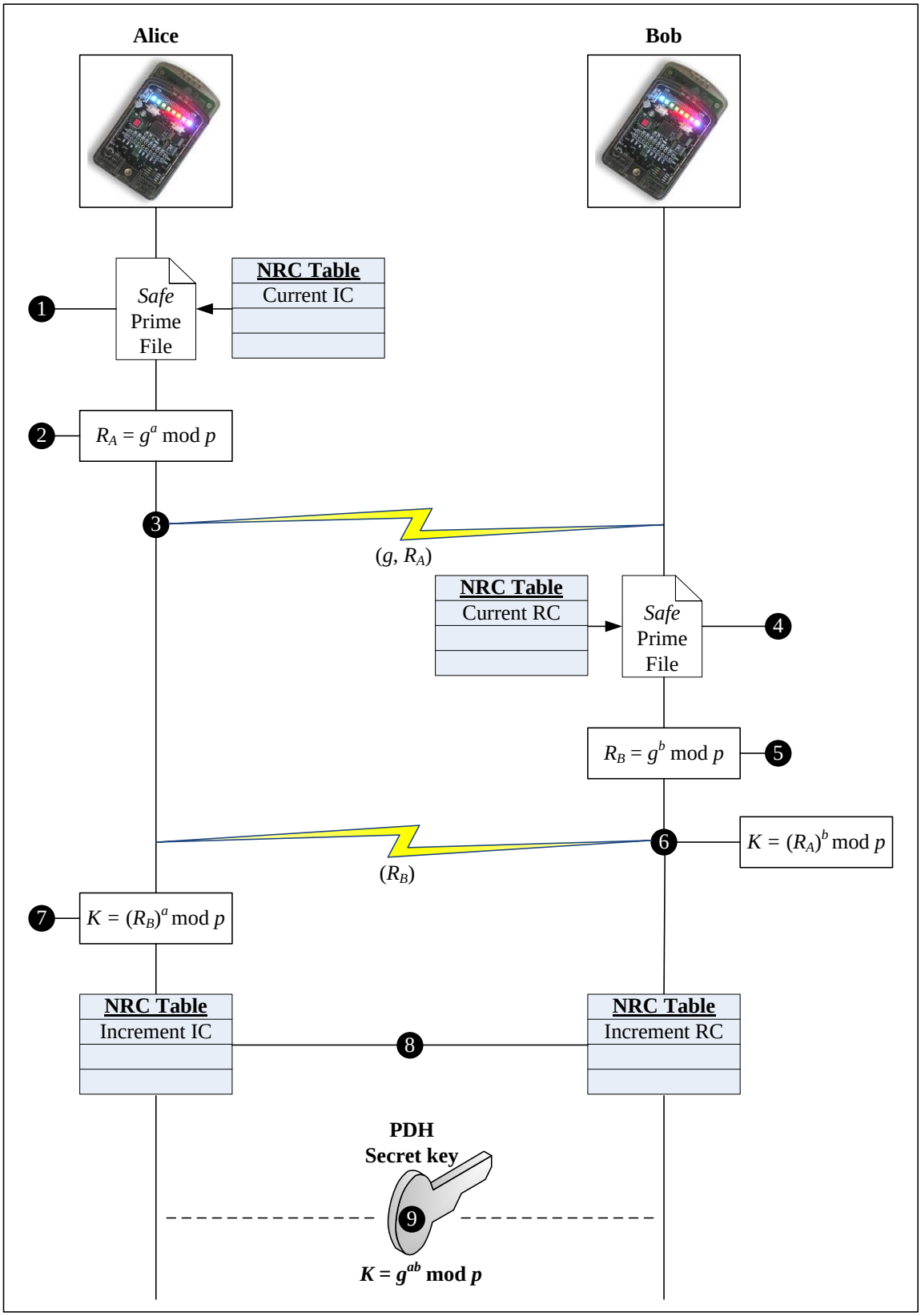

Figure 3.2: PDH Protocol Steps

3.2.3 Security Analysis of the PDH Protocol

The following discussion provides important security issues of the PDH protocol:

- A first look at PDH clearly indicates that it is not vulnerable to the discrete logarithm and the man-in-the-middle attacks because the value of *p* is not public.
- An eavesdropper cannot inject false messages into the network unless he knows the exact prime Alice and Bob are using. So the whole security of PDH relies on keeping the *safe* prime file and the counters secret. If extra security is required, then the *safe* prime file can be encrypted. The file can also be changed (if possible) using a secure link over a period of time.
- The PDH does not provide complete authentication of wireless sensor nodes. The only authentication provided is that parties involved in the communication have assurance that the secret key can only be computed using the *safe* prime file. So if an attacker tries to establish his own key with Alice, they both will end up with different session keys. Also, if an attacker tries to establish his own key with Alice, she will not be sending any secure data to the attacker because the PDH requires the initiator of the key exchange to send the encrypted data. The receiver in the key exchange will only use the session key for decryption of the secret data. If the decrypted data doesn't make any sense, then Alice can identify the node as an intruder and reject any other messages coming from him.
- There is still a possibility of the Denial of Service (DoS) attack from which the wireless sensor network should be protected against. In [24], Wood and Stankovic provide effective techniques that can be adopted.

- One of the main aspects of PDH is to add as much randomness as possible into the Diffie-Hellman key-exchange protocol. That is why the generator *g* is always selected at random. This adds to the security of the PDH because every time a different *p* and *g* is used. An eavesdropper will not be able to figure out which group Alice and Bob are working in.
- Another, important aspect in the security of PDH is how PDH can compute faster? As *p* is private, an eavesdropper cannot determine the value of *p*, thus making the discrete logarithm and the Diffie-Hellman problems even more had. Now an attacker must find the value of *p* before launching the discrete logarithm attack. Therefore, we can significantly reduce the size of *p* from 1024 bits. This will allow us to do faster computations in wireless sensor nodes. We believe that this is the only method by which public-key algorithms such as the Diffie-Hellman key-exchange protocol can actually be implemented on devices having low computing power. In order to make public key-exchange protocols faster, we have to hide certain public parameters so that they are not known to an adversary in advance.

3.3 Implementation of the PDH Protocol

We provide an easy-to-use implementation of the concepts behind the PDH protocol. The implementation is done in *Java* so that it can be used on all platforms. We call the library **Simplified PDH** (S-PDH). S-PDH is an educational tool designed to help students learn the mathematical concepts and secret key generation of the PDH protocol with smaller *safe* primes (9 decimal digits). The implementation should only be used for educational purposes as the source code provided does not include the communication

over the wireless sensor network. So no *counters* are used because our primary focus was to show the mathematical concepts and generation of secret keys. It is much easier to test the source code if implemented stand alone. Furthermore, the results will not vary our implementation if a layer of the communication library is added.

The implementation is done keeping in mind the current Java class library's available for Sun SPOTS. The "BigInteger" class used to generate large numbers is not supported by Sun SPOTS, so that is why the implementation only generates keys of 9 decimal digits. In the next section we take a look at the implementation of S-PDH.

3.3.1 Simplified PDH (S-PDH) Java Library

Simplified-PDH (S-PDH) is as easy-to-use library developed in Java for use with Sun SPOTS. Figure 3.3 shows the classes of the S-PDH library:

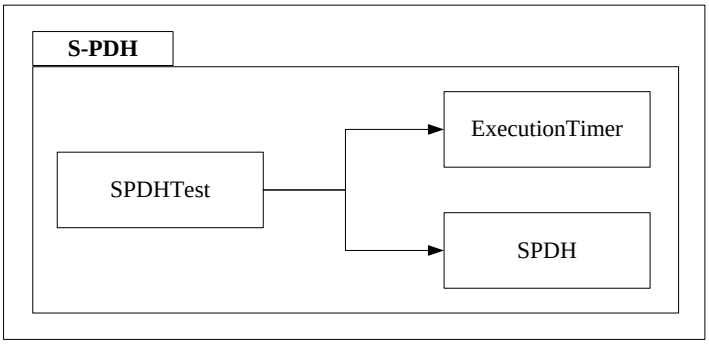

Figure 3.3: S-PDH Class Library

The S-PDH library has three main classes.

1. SPDH Class

The SPDH class provides the functionality of the PDH key-exchange protocol. A brief explanation of some of the important class methods is given below:

- *SPDH* initializes a newly created object to the default values. It also creates an array called *primesArray* containing all *safe* primes by calling the *getPrimesFromFile* method.
- *getPrimesFromFile* method reads all primes from the text file "safe_primes.txt" and puts them in the *primesArray*. The file contains 100,000 *safe* primes.
- *getRandomPrime* method returns a random *safe* prime from the *primesArray*.
- *getOrderOfSubgroup* method returns the order of the cyclic subgroup of quadratic residues modulo *p*.
- *getRandomGenerator* method returns a random generator for the cyclic subgroup of quadratic residues modulo *p*. It picks and arbitrary $x \in \mathbb{Z}_{p^*}$ such that $x \neq \pm 1 \mod p$ and returns $g = x^2 \mod p$.
- *getPrivateKey* method returns a random integer less than the order of the cyclic subgroup of quadratic residues modulo *p*.
- *getPublicKey* method returns the public key i.e., the value $g^{privateKey} \mod p$.
- *getPDHSecretKey* method returns the secret PDH key. The function works by getting a random prime, calculating the order of the subgroup, generating a random generator, computing private and public keys of Alice and Bob and then finally calculating the PDH secret key.
- *power* method returns the value of $b^e \mod p$ by reducing modulo *p* repeatedly throughout the process of computing the result. However, the function is really slow and should not be used for big integer values.
- *fast_exponentiation* method returns the value of $b^e \mod p$ by using the **square and multiply** algorithm. The algorithm drastically reduces both the number of

operations and the memory required to perform modular exponentiation. In [25], Vasyltsov et al. investigated the performance of modern exponentiation algorithms. Figure 3.4 shows the square and multiply algorithm.

```
Input: integer base b, integer exponent e, prime p
Output: b^e mod p
//convert the exponent e to binary and store the bits in an array x
x[1, n] ← convert_to_binary(e)                    //n is the number of bits in x
result ← 1;
for (i ← n down to 1) {                           //the bits are read from right to left
        if (i^th bit of x = 1) result ← (result × b) mod p
        else garbageVariable ← (result × b) mod p          //to avoid timing attacks
        b ← b^2 mod p
}
return result
```

Figure 3.4: Square and Multiply Algorithm

2. ExecutionTimer Class

The ExecutionTimer is a utility class used to calculate the time taken to generate different secret keys.

3. SPDHTest Class

The SPDHTest class has a main function that can be used to test the S-PDH. It also calculates the time taken to generate secret keys by using the ExecutionTimer class.

The steps to test the SPDH class are:

1. Create an object of SPDH class.
2. Use the method *getPDHSecretKey* to compute the secret key between Alice and Bob. The method returns the secret PDH key.
3. To display the protocol parameters use the methods *getPrime*, *getGenerator*, *getOrderOfSubgroup*, *getPrivateKeyAlice*, *getPrivateKeyBob*, *getPublicKeyAlice*, *getPublicKeyBob*. The methods return the corresponding field value.
4. To calculate the execution time of the protocol, create an object of the ExecutionTimer class and use the *start* and *end* methods. This is an optional step that can be added to calculate generation of multiple keys. A loop can be used if multiple key generation is required.

3.4 Evaluation of S-PDH

In this section we present the results and analysis of the results obtained from the evaluation of the S-PDH tool.

3.4.1 Empirical Results

1. The Simplified PDH (S-PDH) tool uses *safe* primes up to 9 decimal digits. Therefore, the S-PDH can only produce keys up to 9 decimal digits.
2. Table 3.1 shows the average execution time for different number of keys generated using S-PDH. The tests were conducted using a Compaq laptop having a Intel(R) Core(TM)2 CPU T5600 @ 183 GHz processor and 1.50 GB of RAM.

Table 3.1: Execution Time for different number of Keys (Sample Size 50)

No. of Keys	Time (nanoseconds)	Time (seconds)
10,000	216,313,805	0.21
100,000	2,066,806,536	2.06
1,000,000	20,412,772,573	20.41

3. The histogram in Figure 3.5 shows the distribution of keys generated when the parameters (*safe* prime *p* and generator *g*) are fixed and the private keys of Alice and Bob (*a* and *b* respectively) are selected randomly. The histogram is symmetric indicating that all keys are evenly distributed. The test was run for 10,000, 100,000 and 1,000,000 keys. However, the distribution of keys remains the same. So we only present the histogram for 1,000,000 keys.

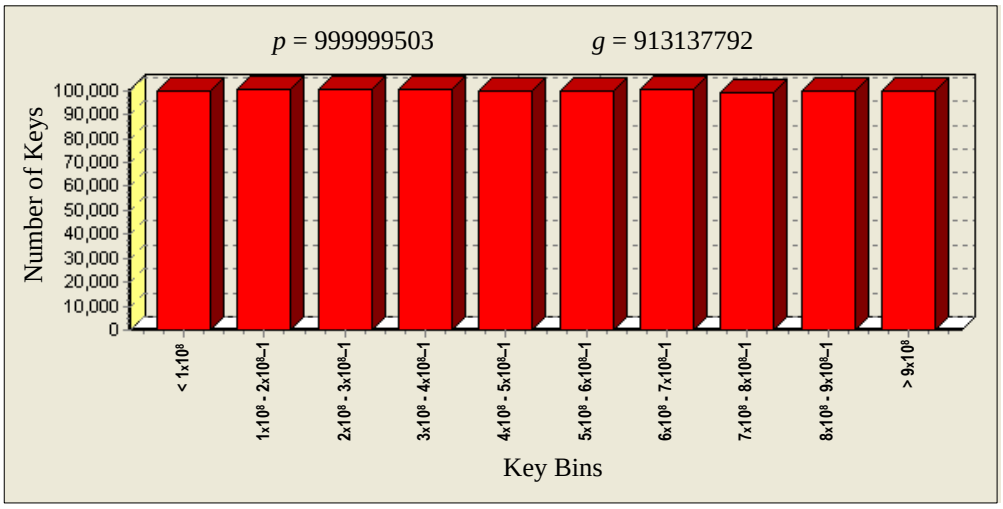

Figure 3.5: Histogram of 1,000,000 Keys with Fixed (*p*, *g*) & Random (*a*, *b*)

4. The histogram is Figure 3.6 shows the distribution of keys generated when the parameters (*p*, *g*, *a*, *b*) are selected randomly. The histogram is slightly skewed to the right. Again as in Figure 3.5, the distribution of keys remains the same for a

run for 10,000, 100,000 and 1,000,000 keys. So we only present the histogram for 1,000,000 keys.

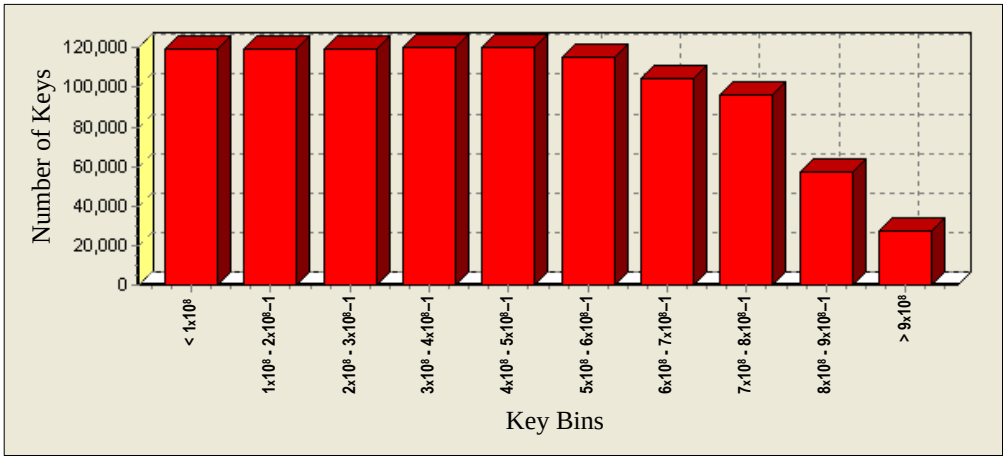

Figure 3.6: Histogram of 1,000,000 Keys with Random (*p*, *g*, *a*, *b*)

5. Table 3.2 shows the difference between the average number of keys duplicated when the parameters (*p* and *g*) were fixed and when the parameters (*p* and *g*) were selected randomly. The private keys (*a* and *b*) are random in both cases. The sample size for the experiment was 50.

Table 3.2: Average Number of Keys Duplicated (Sample Size 50)

No. of Keys	**Fixed *p*, *g***		**Random *p*, *g***	
	2 times	**3 times**	**2 times**	**3 times**
10,000	0.04 (1/25)	0	0	0
100,000	10	0	5	0
1,000,000	1000	1	550	0.2 (1/5)

3.4.2 Analysis of Empirical Results

S-PDH can be used as an educational tool to teach students about the mathematical concepts of the Diffie-Hellman and the Portable Diffie-Hellman key-exchange protocols. The above results show interesting properties of how the different parameters effect the generation of keys.

1. Table 3.1 shows that the S-PDH can generate keys very quickly. Thus, the implementation can be extended easily to bigger prime numbers.
2. The histogram shown in Figure 3.5 is symmetric. This histogram shows that when the value of the *safe* prime *p* and the generator *g* were fixed to 999999503 and 913137792 respectively, the keys were evenly distributed. This means the keys cannot be predicated because all keys are probably.
3. In contrast the histogram shown in Figure 3.6 is slightly skewed to the right. The main reason for this is that *p* is selected randomly from a given range of *safe* primes (526671599 - 999999503). The keys are still fairly evenly distributed indicating that an adversary will not be able to predict the value of the key.
4. Table 3.2 shows that the number of duplicate keys when *p* and *g* were fixed is almost twice compared to when they were selected randomly. This shows that randomizing *p* and *g* decreases the number of duplicate keys. Duplicate keys can be a problem if we are using smaller primes. So it is better to select a different prime every time. However, the duplications will get fewer as the size of the primes increase.

In general because of the results obtained from S-PDH, we can state that the PDH protocol can generate secure and fast keys using the Diffie-Hellman key-exchange

protocol. Thus, the PDH protocol gives us a new direction in which public key-exchange protocols can be implemented in wireless sensor networks. The S-PDH results are promising; however, its use may be limited by the size of the *safe* primes used. In the next chapter, we present an initial study and implementation of the PDH protocol using elliptic curves.

Chapter 4

Portable Diffie-Hellman using Elliptic Curve Cryptography

In this chapter we first explain the Portable Diffie-Hellman (PDH) protocol using elliptic curves (PDH-EC). Afterwards, we discuss the implementation of a Simplified PDH-EC (S-PDH-EC) library designed to show the mathematical concepts and the key generation process behind the PDH-EC protocol. The library is in its initial stage of development. However, it can be used for educational purposes especially for teaching students the mathematics of elliptic curve cryptography.

4.1 The PDH-EC Protocol using *Safe* Primes

The PDH-EC protocol is a variant of the Portable Diffie-Hellman (PDH) protocol presented in Chapter 3. Again we consider a setting in which two wireless sensor nodes execute the protocol in order to generate a shared secret key. For the PDH protocol we considered a file containing a large number of *safe* primes. In contrast, the PDH-EC protocol uses a secret file containing a large number of elliptic curve domain parameters. These elliptic curves are randomly generated using *safe* primes. We refer to this file as "EC Secret Parameters File". The file contains the following parameters of $E(\mathbb{Z}_p)$:

- The *safe* prime $p \geq 7$ of the form $p = 2q + 1$, where q is also prime.
- The parameters A and B defining the elliptic curve equation

  $$y^2 = x^3 + Ax + B \bmod p.$$

- The order of the elliptic curve $|E(\mathbb{Z}_p)|$ i.e., the number of points on the elliptic curve including the point at infinity. For the protocol we consider that the order of all elliptic curves must be a prime number.

The rest of the setting remains the same as the PDH protocol. Following is a brief description about the main aspects of the protocol:

- The *initiate counter* (IC) and *receive counter* (RC) are used by both nodes involved in the communication.
- In order for the communication to happen, Alice will pick an elliptic curve from the EC Secret Parameters File using the current value of IC. Then, she will randomly generate the base point *G* on the elliptic curve. Since, the order of elliptic curve is prime, every point except the point at infinity is a generator. She will then calculate her public key and send it to Bob along with the base point *G*.
- On receiving Alice's public key and the base point *G*, Bob will generate his public key by picking the same elliptic curve (as Alice) using the current value of the RC and the generator *G*. Bob will then send his public key to Alice. Both parties can now calculate the Diffie-Hellman secret key. They can use either the *x* or *y* coordinate of the key for the communication. In our setting, we assume that both parties will use the *y*-coordinate as their secret key.
- At the end of the communication, the value of the IC and RC will be incremented so that the every time a new elliptic curve can be used. Alice and Bob will destroy their private keys after they have calculated the session key. Note that the counters will only be incremented when a key exchange has completely occurred and acknowledgements have been received. The complete working of counters IC and RC is explained in Chapter 3.

The description of the PDH-EC, assumes that Alice will initiate the protocol. So Alice generates the elliptic curve domain parameters (*p*, *A*, *B*, *G*, *n* and *h*). *p*, *A* and *B* are picked

from the EC Secret Parameters File. The order n of the generator G is equal to the order of the elliptic curve i.e. $n = |E(\mathbb{Z}_p)|$. The *cofactor* $h = |E(\mathbb{Z}_p)| / n$, since $n = |E(\mathbb{Z}_p)|$, h is always equal to 1.

Figure 4.1 shows the process of key exchange between Alice and Bob. The steps illustrated in Figure 4.1 are described below:

1. Alice generates an elliptic curve $E(\mathbb{Z}_p)$ from the "EC Secret Parameters File" using the current value of IC. She then generates the base point G.
2. Alice chooses a large random number a such that $0 < a < n$. Using elliptic curve scalar point multiplication, Alice calculates $G_A = aG$ on E.
3. Alice sends (G, G_A) to Bob.
4. On receiving (G, G_A), Bob generates the same elliptic curve $E(\mathbb{Z}_p)$ (as Alice) from the "EC Secret Parameters File" using the current value of RC.
5. Bob chooses a large random number b such that $0 < b < n$. Using elliptic curve scalar point multiplication, Bob calculates $G_B = bG$ on E.
6. Bob sends G_B to Alice and calculates $K = bG_A$.
7. Alice receives G_B and calculates $K = aG_B$.
8. Alice increments IC while Bob increments RC by one when acknowledgement from Alice is received.
9. Both get the same point K for the key i.e.

   $K = aG_B = bG_A = abG$.
10. Alice and Bob use the y-coordinate of the point K as their secret key.

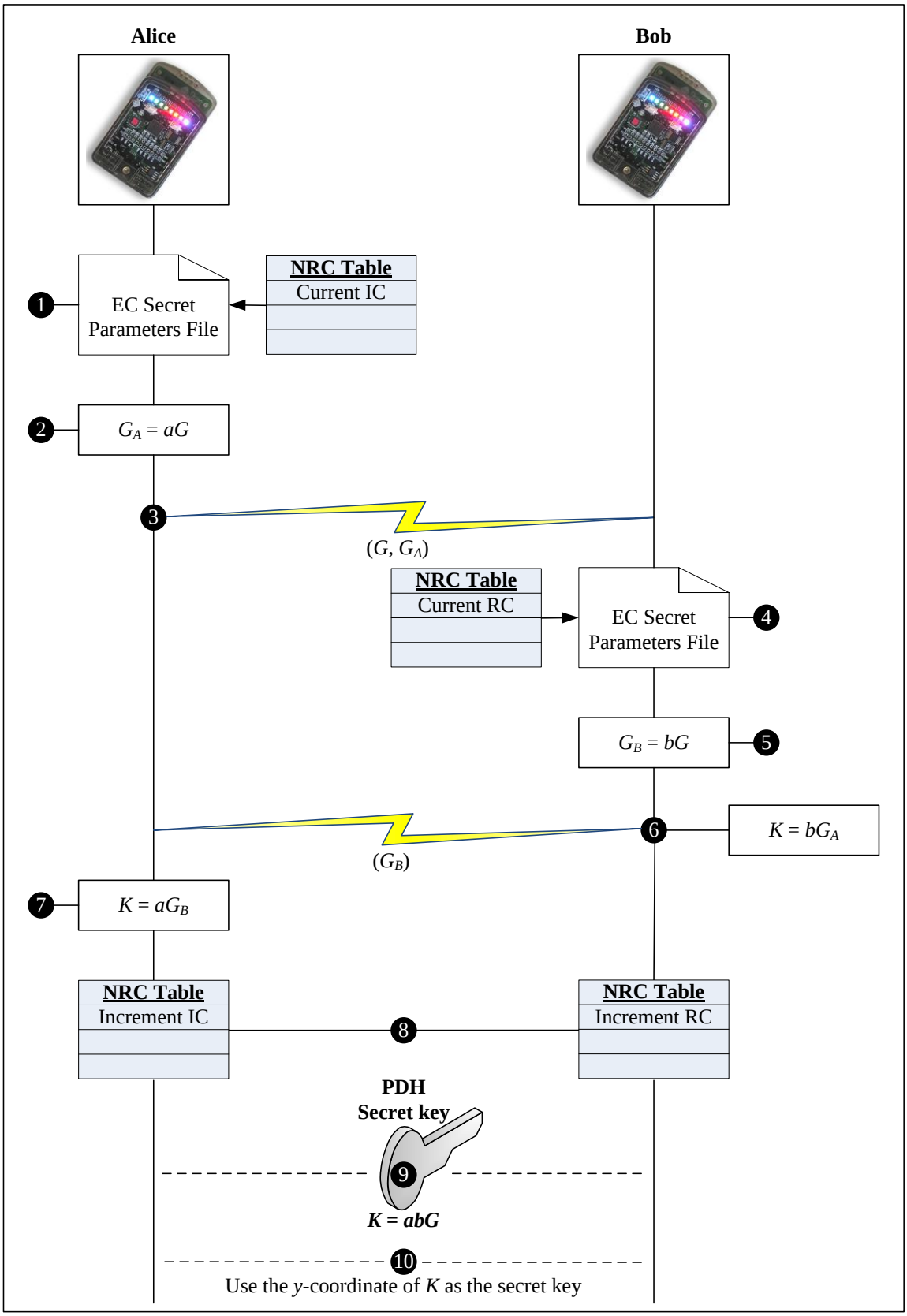

Figure 4.1: PDH-EC Protocol Steps

4.2 Implementation of the PDH-EC Protocol

We provide an easy-to-use implementation of the concepts behind the PDH-EC protocol. The implementation as in S-PDH is in *Java* so that it can be used on all platforms. We call the library **Simplified PDH-EC** (S-PDH-EC). S-PDH-EC is an educational tool designed to help students learn the mathematical concepts of elliptic curve cryptography (ECC) and the key generation process of the PDH-EC protocol. The library uses smaller *safe* primes (up to 7 decimal digits). The library is still under development. So it only uses basic algorithms for scalar multiplication and computing the order of an elliptic curve. Figure 4.2 shows the classes of the S-PDH-EC library. The S-PDH-EC library has seven main classes which are described in the following sections.

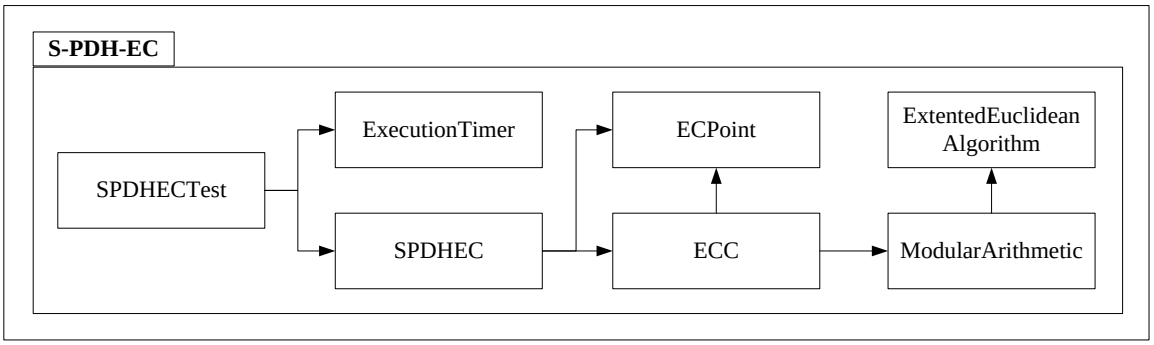

Figure 4.2: S-PDH-EC Class Library

4.2.1 ExtendedEculideanAlgorithm Class

The class implements the extended Euclidean algorithm. The algorithm calculates the greatest common divisor (GCD) of two integers *a* and *b*. It also finds integers *S* and *T* such that $aS + bT = \gcd(a, b)$.

The extended Euclidean algorithm is particularly useful for calculating modular multiplicative inverses in $\mathbb{Z}_p$, since each member of $\mathbb{Z}_p$ (except 0) has a multiplicative

inverse. This means that given an integer $a \in \mathbb{Z}_p$ such that $a \neq 0$, the gcd(a, p) = 1 and the multiplicative inverse of a is the value of S after being mapped to $\mathbb{Z}_p$.

4.2.2 ModularArithmetic Class

The class provides the basic functionality for modular arithmetic. Some of the important class methods are given below:

- *Mod* method returns the value of a mod p.
- *fast_exponentiation* method returns the value of b^e mod p by using the square and multiply algorithm (Figure 3.4).
- *MultiplicativeInverse* method returns the modular multiplicative inverse of an integer in $\mathbb{Z}_p$. Figure 4.3 shows the algorithm for computing modular multiplicative inverses.

Input: Modulas n; element $a \in \mathbb{Z}_n$

Output: [a^{-1} mod n] (if it exists)

(d, S, T) $\leftarrow$ *ExtendedEuclideanAlgorithm.getGCD*(a, n) //note that $aS + nT = GCD(a, n)$

if ($d \neq 1$) **return** "a is not invertible modulo n"

else return [S mod n]

Figure 4.3: Algorithm for computing Modular Multiplicative Inverse

- *isQuadraticResidue* method returns true if an integer x is a quadratic residue modulo p in $\mathbb{Z}_{p^*}$. In the equation $y^2 \equiv z \mod p$, z is called a quadratic residue (QR) if the equation has two solutions; z is called a quadratic non-residue (QNR) if the equation has no solutions. Figure 4.4 shows the algorithm for deciding quadratic residuosity modulo a prime.

Input: A prime p; element $x \in \mathbb{Z}_{p^*}$

Output: Whether x is a quadratic residue or quadratic non-residue

$b \leftarrow$ *fast_exponentiation*(x, $(p-1)/2$, p)

if ($b = 1$) **return** "Quadratic Residue"

else return "Quadratic Non-Residue"

Figure 4.4: Algorithm for deciding Quadratic Residuosity modulo a prime

- *ModularSquareRoot* method returns the square root modulo a *safe* prime of the form $p \equiv 3 \bmod 4$. Figure 4.6 shows the algorithm for computing squaring roots modulo a *safe* prime greater than 5.

Input: A *safe* prime p of the form $p \equiv 3 \bmod p$; quadratic residue $a \in \mathbb{Z}_{p^*}$

Output: A square root of a i.e. $x \equiv a^{\frac{p+1}{4}} \bmod p$

$x \leftarrow$ *fast_exponentiation*(a, $(p+1)/4$, p)

return x

Figure 4.6: Algorithm for computing Square Roots modulo a *safe* prime

4.2.3 ECPoint Class

The class is used to represent a point (x, y) on the elliptic curve.

4.2.4 ECC Class

ECC is the main class that provides the functionality required for elliptic curve cryptography. Some of the important class methods are given below:

- *ECC* initiates a newly created object. The ECC class has three constructors i.e.
    1. If the constructor is called without any arguments, then it initiates a newly created object to the default field values.

2. If the constructor is called by passing the elliptic curve parameters *A*, *B* and *safePrime*, then it constructs a new ECC object so that it represents the specified elliptic curve parameters. However, if the elliptic curve is singular or the order of elliptic curve is not prime, then the constructor prints an error message and returns without initiating the field values.
3. If the constructor is called by passing the elliptic curve parameters *A*, *B*, *safePrime* and *primeOrder*, then it constructs a new ECC object so that it represents an elliptic curve of prime order.

- *generateRandomEC* method generates a random elliptic curve of prime order. The function takes a lot of time to generate a curve. So it should only be used to generate new elliptic curves.
- *isECNonSingular* method returns true if the elliptic curve is non-singular. Figure 4.7 shows the algorithm for deciding non-singularity of an elliptic curve.

Input: A *safe* prime $p \geq 7$; an elliptic curve $y^2 = x^3 + Ax + B \bmod p$

Output: Whether an elliptic curve is singular or non-singular

result $\leftarrow [4A^3 + 27B^2 \bmod p]$

if (result = 0) **return** "Singular"

else return "Non-Singular"

Figure 4.7: Algorithm for deciding Non-Singularity of an Elliptic Curve

- *IsEqual* method returns true if two points on the elliptic curve have the same *x* and *y* coordinates i.e. $x_1 = x_2$ and $y_1 = y_2$.
- *IsAdditiveInverse* method returns true if two points on the elliptic curve are additive inverses of each other i.e. $x_1 = x_2$ and $y_1 = -y_2$.

- *ECPointAddition* method returns the result of adding two points on the elliptic curve. Figure 4.8 shows the algorithm for point addition.

Input: A *safe* prime $p \geq 7$; two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on the elliptic curve

$y^2 = x^3 + Ax + B \bmod p$ with $P_1, P_2 \neq \boldsymbol{O}$ (point at infinity)

Output: $P_1 + P_2 = P_3 = (x_3, y_3)$

if ($x_1 = x_2$ & $y_1 \neq y_2$) **return** [$P_1 + P_2 = \boldsymbol{O}$] //additive inverse i.e. $P_1 = -P_2$

if ($P_1 = P_2$ & $y_2 = 0$) **return** [$P_1 + P_2 = 2P_1 = \boldsymbol{O}$] //point doubling

if ($P_1 = P_2$ & $y_2 \neq 0$) { //point doubling

$m \leftarrow [(3x_1^2 + A) \bmod p] \times [\mathit{MultiplicativeInverse}(2\, y_1, p) \bmod p] \bmod p$

$x_3 \leftarrow [m^2 - 2x_1 \bmod p]$

$y_3 \leftarrow [m(x_1 - x_3) - y_1 \bmod p]$

r**eturn** [$P_3 = (x_3, y_3)$]

}

if ($x_1 \neq x_2$) { //point addition

$m \leftarrow [(y_2 - y_1) \bmod p] \times [\mathit{MultiplicativeInverse}\,(x_2 - x_1\,, p) \bmod p] \bmod p$

$x_3 \leftarrow [m^2 - x_1 - x_2 \bmod p]$

$y_3 \leftarrow [m(x_1 - x_3) - y_1 \bmod p]$

r**eturn** [$P_3 = (x_3, y_3)$]

}

Figure 4.8: Point Addition Algorithm

- *ECPointDoubling* method returns the result of doubling a point on an elliptic curve. Figure 4.9 shows the algorithm for point doubling.

Input: A *safe* prime $p \geq 7$; a point $P = (x_1, y_1)$ on the elliptic curve $y^2 = x^3 + Ax + B \bmod p$ with $P \neq \boldsymbol{O}$ (point at infinity)

Output: $P + P = P_3 = (x_3, y_3)$

if ($y_2 = 0$) **return** [$P + P = 2P = \boldsymbol{O}$]

else {

$m \leftarrow [(3x_1^2 + A) \bmod p] \times [MultiplicativeInverse(2\, y_1, p) \bmod p] \bmod p$

$x_3 \leftarrow [m^2 - 2x_1 \bmod p]$

$y_3 \leftarrow [m(x_1 - x_3) - y_1 \bmod p]$

r**eturn** [$P_3 = (x_3, y_3)$]

}

Figure 4.9: Point Doubling Algorithm

- *ECDoubleAndAdd* method returns the result of multiplying a point on the elliptic curve with a scalar. Figure 4.10 shows the double and add algorithm for scalar multiplication.

Input: A *safe* prime $p \geq 7$; a point $P = (x_1, y_1)$ on the elliptic curve $y^2 = x^3 + Ax + B \bmod p$ with $P \neq \boldsymbol{O}$ (point at infinity); a scalar a

Output: $Q = aP$

//convert the scalar a to binary and store the bits in an array x

$x[1, n] \leftarrow convert\_to\_binary(a)$ //n is the number of bits in x

$Q \leftarrow P$

for ($i \leftarrow 2$ to n) {

$Q \leftarrow PointDoubling(Q)$

if (i^{th} bit of $x = 1$) $Q \leftarrow PointAddition(Q, P)$

}

return Q

Figure 4.10: Double and Add Algorithm

- *GenerateAllECPoints* method generates all points on the elliptic curve and prints them on the screen. Figure 4.11 shows the algorithm for generating all points on the elliptic curve.

Input: A *safe* prime $p \geq 7$; an elliptic curve $y^2 = x^3 + Ax + B \bmod p$

Output: Print all points on the elliptic curve

$P = (x_1, y_1)$, $Q = (x_2, y_2)$

$x \leftarrow 0$

while ($x < p$) {

    $w \leftarrow [x^3 + Ax + B \bmod p]$

    if ($w = 0$) {

        print "$P = (x, 0)$"

        $x \leftarrow x + 1$

        continue

    }

    if (*isQuadraticResidue*(w, p)) {

        $sqrt \leftarrow$ *ModularSquareRoot*(w, p)

        $x_1 \leftarrow x$

        $y_1 \leftarrow sqrt$

        $x_2 \leftarrow x$

        $y_2 \leftarrow [-1 \times sqrt + p]$

        print "$P = (x_1, y_1)$ & $Q = (x_2, y_2)$"

        $x \leftarrow x + 1$

    }

}

Figure 4.11: Algorithm for finding all points on the Elliptic Curve

- *GenerateRandomECPoint* method returns a random point on the elliptic curve. The function can be used to get a generator of the elliptic curve. Figure 4.12 shows the algorithm for finding a random point on an elliptic curve.

Input: A *safe* prime $p \geq 7$; an elliptic curve $y^2 = x^3 + Ax + B \bmod p$

Output: a random point $P = (x_1, y_1)$ on the elliptic curve

```
while (true) {
        x ← Random(0, p)                    //randomly pick an integer x such that 0 ≤ x < p
        w ← [x^3 + Ax + B mod p]
        if (w = 0) {
                x1 ← x
                y1 ← w
                break
        }
        if (isQuadraticResidue(w, p)) {
                sqrt ← ModularSquareRoot(w, p)
                x1 ← x
                r ← Random(0, 2)            //randomly pick an integer r such that 0 ≤ r < 2
                if (r = 0) y1 ← sqrt
                else y1 ← [–1 × sqrt + p]
                break
        }
}
return [P = (x1, y1)]
```

Figure 4.12: Algorithm for finding a Random Point on the Elliptic Curve

- *isPrime* method returns true if the given number is prime.

- *getECOrder* method returns the total number of points on an elliptic curve including the point at infinity. Figure 4.13 shows the algorithm for computing the order of an elliptic curve. The algorithm takes a lot of time and should only be used with smaller *safe* primes. Otherwise, it would be impossible to find the order of the elliptic curve.

Input: A *safe* prime $p \geq 7$; an elliptic curve $y^2 = x^3 + Ax + B \bmod p$

Output: number of points on the elliptic curve including the point at infinity

ecOrder $\leftarrow p + 1$

$x \leftarrow 0$

while ($x < p$) {

    $w \leftarrow [x^3 + Ax + B \bmod p]$

    if ($w = 0$) {

        ecOrder $\leftarrow$ *ecOrder* + 0

        $x \leftarrow x + 1$

        continue

    }

    if (*isQuadraticResidue*(w, p)) *ecOrder* $\leftarrow$ *ecOrder* + 1

    else *ecOrder* $\leftarrow$ *ecOrder* – 1

    $x \leftarrow x + 1$

}

return *ecOrder*

Figure 4.13: Algorithm for computing Order of an Elliptic Curve

- *getECOrderOfElement* method returns the order of an element. For elliptic curves having prime order, the order of an element is always equal to the order of the elliptic curve.

4.2.5 SPDHEC Class

The SPDHEC class provides the functionality of the PDH-EC protocol. A brief explanation of some of the important class methods is given below:

- *SPDHEC* initializes a newly created object to the default values. It also creates an array called *ecParameterArray* containing parameters for all elliptic curves by calling the *getECFromFile* method.
- *getECFromFile* method reads parameters for all elliptic curves from the text file "ec_parameters.txt" and puts them in the *ecParameterArray*. The file contains parameters for 10 elliptic curves.
- *getRandomEC* method randomly selects an elliptic curve from the *ecParameterArray*.
- *getRandomGenerator* method returns a random generator *G* for the elliptic curve by calling the *generateRandomECPoint* method of the ECC class.
- *getPrivateKey* method returns a random integer less than the order of the elliptic curve.
- *getPublicKey* method returns the public key for Alice and Bob.
- *getPDHECSecretKey* method returns the secret PDH-EC key. The function works by picking a random elliptic curve, generating a random generator for the elliptic curve, computing the private and public keys of Alice and Bob and then finally calculating the PDH-EC secret key.

4.2.6 ExecutionTimer Class

The ExecutionTimer is a utility class used to calculate the time taken to generate different secret keys.

4.2.7 SPDHECTest Class

The SPDHECTest class has a main method that can be used to test the S-PDH-EC. It also calculates the time taken to generate secret keys by using the ExecutionTimer class.

The steps to test the SPDHEC class are:

1. Create an object of SPDHEC class.
2. Use the method *getPDHECSecretKey* to compute the secret key between Alice and Bob. The method returns the secret PDH-EC key.
3. To display the protocol parameters use the methods *getA, getB, getPrime, getGenerator, getPrivateKeyAlice, getPrivateKeyBob, getPublicKeyAlice, getPublicKeyBob*. The methods return the corresponding field value.
4. To calculate the execution time of the protocol, create an object of the ExecutionTimer class and use the *start* and *end* methods. This is an optional step that can be added to calculate generation of multiple keys. A loop can be used if multiple key generation is required.

4.3 Evaluation & Analysis of Empirical Results of S-PDH-EC

In this section we present the results obtained from the evaluation of the S-PDH-EC tool.

1. The Simplified PDH-EC (S-PDH-EC) tool uses *safe* primes up to 7 decimal digits. Therefore, the S-PDH can only produce keys up to 7 decimal digits.
2. Table 4.1 shows the elliptic curves on which the S-PDH-EC protocol was tested. These are experimental curves of prime order generated to test the performance of the S-PDH-EC. These curves can also be used for educational purposes.

Table 4.1: Experimental Elliptic Curves of Prime Order

No.	Elliptic Curves $E(\mathbb{Z}_p)$	Prime Order
1	$y^2 = x^3 + 4x + 1 \bmod 7$	5
2	$y^2 = x^3 + 8x + 1 \bmod 11$	17
3	$y^2 = x^3 + x + 4 \bmod 23$	29
4	$y^2 = x^3 + 13x + 48 \bmod 47$	53
5	$y^2 = x^3 + 66x + 28 \bmod 83$	97
6	$y^2 = x^3 + 940279\ x + 909289 \bmod 38603$	38449
7	$y^2 = x^3 + 12x + 39 \bmod 811379$	812101
8	$y^2 = x^3 + 390310\ x + 241001 \bmod 280703$	281423
9	$y^2 = x^3 + 783256\ x + 261836 \bmod 1173539$	1171463
10	$y^2 = x^3 + 830100\ x + 221139 \bmod 3318167$	3320477
11	$y^2 = x^3 + 681732\ x + 768442 \bmod 5394539$	5397697
12	$y^2 = x^3 + 258x + 306 \bmod 5847239$	5845403
13	$y^2 = x^3 + 433778\ x + 263620 \bmod 6348383$	6348977
14	$y^2 = x^3 + 1330053\ x + 8830619 \bmod 8793839$	8795089
15	$y^2 = x^3 + 4959549\ x + 2491395 \bmod 9999047$	10000721

3. Table 4.2 shows the average execution time for different number of keys generated using curve number 15 in Table 4.1. The tests were conducted using a Compaq laptop having a Intel(R) Core(TM)2 CPU T5600 @ 183 GHz processor and 1.50 GB of RAM.

Table 4.2: Execution Time for different number of Keys (Sample Size 10)

No. of Keys	Time (nanoseconds)	Time (seconds)
10,000	2,473,430,841	2.47
100,000	24,628,840,816	24.62
1,000,000	245,552,982,013	245.55

The execution time for S-PDH-EC is 10 times slower than the S-PDH protocol. The main reason for this is that S-PDH-EC is in its initial development stage. So we have implemented the basic algorithms especially for point counting. Further enhancements in the algorithms will significantly increase the computation cost of the S-PDH-EC tool.

In the next chapter we present the summary of our research and conclusions. We also discuss the future work planned for the S-PDH and S-PDH-EC protocols.

Chapter 5

Conclusions and Future Work

5.1 Summary

This thesis investigated the mathematical foundations of the Diffie-Hellman key-exchange protocol and the elliptic curve cryptography for the purpose of understanding the practical problems of implementing the theoretical concepts in wireless sensor networks. The main results are as follows:

1. Designed a new improved protocol for establishing secure keys in wireless sensor networks called the Portable Diffie-Hellman (PDH). The protocol is not vulnerable to the discrete logarithm and the man-in-the-middle attacks.
2. Developed a portable implementation of the PDH protocol called Simplified PDH (S-PDH) in Java. The evaluation of the protocol showed that S-PDH has an even distribution of keys so that an adversary cannot predict the value of keys. Furthermore, there were no duplicates for 10,000 keys.
3. Designed a variant of the PDH protocol called the Portable Diffie-Hellman using Elliptic Curves (PDH-EC).
4. Developed a portable implementation of the PDH-EC protocol called Simplified PDH-EC (S-PDH-EC) in Java. The performance evaluation of the key generation computation of PDH-EC compared to S-PDH is 10 times slower. So further enhancements in the algorithms are required.
5. S-PDH and S-PDH-EC are also educational tools designed to help students understand and experiment how the mathematical properties of the public and private parameters affect the key generation process in the Diffie-Hellman key-

exchange protocol and elliptic curve cryptography. Furthermore, the tools can be used to understand how to implement the theoretical aspects of the Diffie-Hellman key-exchange protocol. The tools can also be used to understand the secret key generation process of the PDH and the PDH-EC protocols.

In general the PDH protocol introduces a new concept in which public key-exchange protocols can be used in wireless sensor networks. The idea of implementing public-key algorithms by hiding particular public parameters (such as the prime number) is probably the only way in which public-key encryption can be used in today's wireless sensor networks. The PDH protocol shows that:

- The randomization of the public parameters gives a new method of establishing secure keys in wireless sensor networks. S-PDH was used to test the randomization and the results show that the ideas presented in the PDH protocol can actually be implemented for a wireless sensor network that uses Sun SPOTS.
- The selection of the *safe* prime *p* (from a secret file containing a large number of *safe* primes) and the randomization of *g* can significantly reduce the size of *p* from the current standard of 1024 bits (or 300 decimal digits) in the Diffie-Hellman key-exchange protocol. Thus, reducing the computation cost of the protocol and making it feasible for use in wireless sensor networks.
- It can solve the problem of key distribution and management in wireless sensor networks as long as the network has a reasonable protection against physical capturing of nodes. Even if a single node in the wireless sensor network is compromised, the rest of the nodes can still establish secure keys for a certain amount of time depending on the time it will take to decrypt the primes file.

5.2 Future Work

Our future work contains:

- Deployment of the PDH and the PDH-EC protocols on the Sun SPOTS.
- Design and implement a set of attacks against PDH and the PDH-EC protocols.
- Calculating the computation cost of the PDH and PDH-EC protocols after deployment on the Sun SPOTS.
- Testing the key generation process between multiple Sun SPOT nodes and test its use in a multicast implementation.
- Implementation of faster algorithms (for point counting and calculating order of the elliptic curves) for the S-PDH-EC protocol.
- Develop animation tools that model the key generation process in PDH and the PDH-EC protocols.

REFERENCES

[1] D. Boyle and T. Newe, "Security Protocols for Use with Wireless Sensor Networks: A Survey of Security Architectures," in *Proceedings of the Third International Conference on Wireless and Mobile Communications (ICWMC '07)*, Guadeloupe, French Caribbean, 2007, p. 54.

[2] L. F. Akyildiz, *et al.* (2002, Aug.) A Survey on Sensor Networks. *IEEE Communications Magazine*. 102-114. Available: http://dl.comsoc.org/comsocdl/

[3] A. Perrig, *et al.* (2004, June) Security in wireless sensor networks. *Communications of the ACM* [Online]. 53-57. Available: http://cacm.acm.org/magazines/2004/6

[4] J. Großschädl, *et al.*, "The energy cost of cryptographic key establishment in wireless sensor networks," in *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, Singapore, 2007, pp. 380-382.

[5] Y. Xiao, *et al.*, "A Survey of Key Management Schemes in Wireless Sensor Networks," *Computer Communications: Special Issue on Security on Wireless Ad Hoc and Sensor Networks*, vol. 30, pp. 2314-2341, Sept. 2007.

[6] W. Du, *et al.*, "A pairwise key predistribution scheme for wireless sensor networks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 8, pp. 228-258, May 2005.

[7] A. Becher, *et al.*, "Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks," in *Proceeding of the 3rd International Conference on Security in Pervasive Computing (SPC)*, York, UK, 2006, pp. 104-118.

[8] K. Xing, *et al.*, "Real-time Detection of Clone Attacks in Wireless Sensor Networks," in *Proceedings of the 28th IEEE International Conference on Distributed Computing Systems (ICDCS '08)*, Beijing, China, 2008, pp. 3-10.

[9] R. B. Smith, "SPOTWorld and the Sun SPOT," in *Proceedings of the 6th international conference on Information processing in sensor networks (IPSN '07)*, Cambridge, Massachusetts, 2007, pp. 565-566.

[10] B. A. Forouzan, *Cryptography and Network Security*, International ed.: McGraw Hill, 2008.

[11] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*: Chapman and Hall/CRC, 2008.

[12] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, pp. 644- 654, Nov. 1976.

[13] H. Krawczyk, "HMQV: A High-Performance Secure Diffie-Hellman Protocol," in *Advances in Cryptology—Proceedings of CRYPT0 2005*, Santa Barbara, California, 2005, pp. 546-566.

[14] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203-209, 1987.

[15] V. S. Miller, "Use of Elliptic Curve in Cryptography," in *Advances in Cryptology—Proceedings of CRYPT0 '85*, Santa Barbara, California, 1985, pp. 417-426.

[16] J. López and R. Dahab, "An Overview of Elliptic Curve Cryptography," *Relatório Técnico IC-00-10*, 2000.

[17] K. Lauter. (2004, Feb) The Advantages of Elliptic Curve Cryptography for Wireless Security. *IEEE Wireless Communications Mag.* 62-67.

[18] L. C. Washington, *Elliptic Curves, Number Theory and Cryptography*: Chapman and Hall/CRC, 2003.

[19] R. Schoof, "Elliptic curves over finite fields and the computation of square roots mod *p*," *Mathematics of Computation,* vol. 44, pp. 483-494, 1985.

[20] I. Blake, *et al.*, "Schoof's Algorithm and Extensions," in *Elliptic Curves in Cryptography*. vol. 265, ed Cambridge: University Press, 2000, pp. 109-147.

[21] N. Zhang, *et al.*, "Efficient elliptic curve scalar multiplication algorithms resistant to power analysis," *Information Sciences: an International Journal,* vol. 177, pp. 2119-2129, 2007.

[22] T. Satoh and K. Araki, "Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves," *Comm. Math. Univ. Sancti Pauli,* vol. 47, pp. 81-92, 1998.

[23] N. P. Smart, "The discrete logarithm problem on elliptic curves of trace one," *Journal of Cryptology,* vol. 12, 1999.

[24] A. D. Wood and J. A. Stankovic. (2002, Oct.) Denial of Service in Sensor Networks. *Computer Magazine*. 54-62. Available: http://www2.computer.org/portal/web/csdl/doi/10.1109/MC.2002.1039518

[25] I. Vasyltsov, *et al.*, "Investigation of modern exponentiation algorithms," in *Modern Problems of Radio Engineering, Telecommunications and Computer science—Proceedings of the International Conference*, 2004, pp. 291-293.