Software Implementation of Gong-Harn Public-key Cryptosystem and Analysis

by

Susana Sin

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2004

I hereby declare that I am the sole author of this thesis.

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

Susana Sin

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

Susana Sin

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

With the emergence of the 3G (third-generation) networks for mobile communications, data security becomes ever more important. Designing cryptosystems that meet both power constraints and computing constraints of mobile units is very challenging. The Gong-Harn Public-Key Cryptosystem (GH-PKC) reduces the size of the modulus and speeds up the computations with the same degree of security as existing cryptosystems. This thesis focuses on the software implementation of the GH-PKC and provides an analysis of its performance over the existing cryptosystems. The GH Diffie-Hellman (GH-DH) key agreement protocol and the GH digital signature algorithm (GH-DSA) is implemented in both C++ and Java. The Diffie-Hellman key agreement protocol and Digital Signature Scheme have also been implemented in Java for comparison. To analyze the performance on constrained devices, RIM's Blackberry handheld is chosen as a platform.

Acknowledgement

I would like to thank Dr. G. Gong, my supervisor, for her suggestions, encourgement and constant support during this research. I would also like to thank my committee members, Dr. A. Hasan and Dr. G. Agnew, for their valuable comments on my thesis.

My colleagues also gave me valuable comments on my research and seminar. I would like to thank them all, and especially Kenneth Giuliani, who helped me in generating parameters and polynomials.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the emergence of the third generation network for mobile communication, while speech transmission is still dominating the airway, the demands for fax, short messages and data transmissions are growing rapidly. Combined with the rapid development of Internet applications, data security becomes even more important. Designing cryptosystems that meet both power constraints and computing constraints of mobile units is very challenging.

In traditional cryptography, encryption and decryption are performed using the same secret key. This method is known as secret key or symmetric cryptography. The main challenge is getting the sender and the receiver to agree on the secret key without anyone else finding out. When the sender and the receiver are physically apart, they agree on the secret key through a trusted third party to prevent the disclosure of the secret key.

An example of secret key cryptography can be found in Global System for Mobile Communication (GSM) networks. In a GSM network, an 128-bit secret key, k_i, is shared between a mobile device and the system which is used for both authentication and cipher key generation. The algorithms for authentication and cipher key generation are A3 and A8 respectively. Both algorithms take in the same input: the subscriber's k_i and a 128-bit random challenge sent by the Mobile Switching Center (MSC), they are combined into a single algorithm known as COMP128 [2]. Several attacks have already been done on the algorithm, which allow the retrieval of the full 128-bit k_i in as few as seven chosen plaintext attacks [21].

Public-key cryptography, however, resolves the key management problem. The theoretical concept of public-key cryptography was developed by Diffie and Hellman [5] in 1976. In a public-key cryptosystem (PKC), each user gets a pair of keys: private key and public key. As implied by the name, the public key is published whereas the private key is kept secret. The private key is used to generate the public key, but the public key cannot be used to determine what the associated private key is. Diffie and Hellman proposed the Diffie-Hellman (DH) key agreement protocol [5] which allows users to exchange secret keys over an insecure channel without any prior shared secret. This protocol provides the basic operation for Internet Key Exchange (IKE) protocol [4]. The security of DH is based on intractability of discrete logarithm (DL) problem in GF(p) and all operations are performed in GF(p). The parameter p in DH key agreement protocol is called a modulus. An increase in modulus by n bits contributes to an increase in security by n bits. Public-key cryptography is mainly applied in encryption/decryption schemes and digital signature schemes. The fourth-generation (4G) mobile network is already being designed to utilize public key cryptography.

For encryption purposes, a sender only needs to know the receiver's public key in order to encrypt a message. This encrypted message can only be decrypted by the designated receiver using the private key. The RSA scheme was developed by Rivest, Shamir and Adleman [27] in 1978. The security is based on intractability of integer factorization of the parameter n. A year later, Rabin developed the Rabin scheme [26] as an alternative to the RSA. The security is based on the intractability of factoring and the computational equivalence of square root and factoring.

Digital signature schemes are designed to provide the digital counterpart to hand-written signatures. They provide data integrity, data origin authentication, and non-repudiation. A digital signature is generated based on the content of the message being signed and some secrets known only to the signer including the private key and the signing key. It must be verifiable by any user in the system without accessing the signer's secret information.

Digital signature schemes are classified according to the underlying mathematical problem which provides the basis for their security in IEEE Standard Specifications for Public Key Cryptography [14]:

1. Discrete Logarithm schemes: security is based on the intractability of the DL problem in a finite field. Examples include ElGamal [6], DSS [20], Schnorr [30] and Nyberg-Rueppel [24, 25] signature schemes.
2. Integer Factorization (IF) schemes: security is based on the intractability of the integer factorization problem. Examples include RSA [27] and Rabin [26] signature schemes.

3. Elliptic Curve (EC) schemes: security is based on the intractability of the elliptic curve DL problem. Examples include Elliptic Curve Digital Signature Algorithm [15, 18, 1].

A sequence generated by linear feedback shift register (LFSR) with a specific initial state is called a characteristic sequence. Cryptosystems that make use of characteristic sequences have the benefit in reducing the size of the modulus while speeding up the computations with the same degree of security as existing cryptosystems. These cryptosystems are desirable as they can minimize computational cost.

In 1994, Smith and Skinner proposed the LUC Public-key cryptosystem [32, 33] which is based on second-order characteristic sequences over GF(p), where p is a prime number. The security is based on the intractability of DL problem in GF(p^2) while all the operations are performed in GF(p). In 1999, Gong and Harn proposed the Gong-Harn Public-key Cryptosystem (GH-PKC) [10, 11] which is based on third-order characteristic sequences over GF(q), where q equals to p or p^2. The security is based on the intractability of DL problem in GF(q^3) while all the operations are performed in GF(q). In 2000, Lenstra and Verheul proposed the XTR Cryptosystem [16, 17] which is based on third-order characteristic sequences over GF(p^2) with a special polynomial. The security is based on the intractability of DL problems in GF(p^6) while all the operations are performed in GF(p^2). Note that both the GH-PKC and the XTR cryptosystem are based on third-order characteristic sequences, and thus, are very similar. The difference between the two cryptosystems is that XTR requires a special polynomial to generate third-order characteristic sequences. In 2003, Giuliani and Gong proposed a cryptosystem which analogues to both the GH and XTR cryptosystems using fifth-order characteristic sequences

over GF(q) where q equals to p or p^2 [8, 9]. The security is based on the intractability of DL problem in GF(q^5) while all the operations are performed in GF(q). Therefore, an increase in p by n bits contributes to an increase in security by $2n$ bits for LUC, $3n$ bits or $6n$ bits for GH, $6n$ bits for XTR and $5n$ or $10n$ bits for the public-key cryptosystem based on quintic finite field extensions [8, 9]. Note that LUC, GH, XTR and the public-key cryptosystem based on quintic finite field extensions are non-standardized public-key algorithms.

Gong and Harn proposed the GH Diffie-Hellman (GH-DH) key agreement protocol in [10] and the GH Digital Signature Algorithm (GH-DSA) in [11]. GH-DH key agreement protocol and GH-DSA, like DH and DSS, belong to the DL class of PKC. They aim to improve the performance over existing DH and DSS systems while maintaining the same level of security. The goal of this research is to implement the GH-DH key agreement protocol and the GH-DSA in software over GF(p) with 1024-bit security. Some design issues that encountered during the course of this implementation is presented in the thesis. Experiments have been done for bandwidth saving problem in the GH-DSA, which provide a certain guidance for theoretically solving this problem. This thesis also includes an implementation of DH and DSS systems. Analysis and comparisons are done based on the author's own implementation as existing commercial implementations of DH and DSS are heavily optimized. Timing analysis is done on a personal computer as well as on a constrained device.

This thesis is organized as follows. Chapter 2 reviews the Diffie-Hellman key agreement protocol and the Digital Signature Standard. It also explains the third-order characteristic sequences and the concept of reciprocal sequences. Three algorithms for computing sequence terms in third-order characteristic sequences are provided in Chapter 3.

Chapter 4 presents the GH-PKC, including the GH Diffie-Hellman key agreement protocol and the GH digital signature algorithm. Chapter 5 explains the design issues for the software design. Chapter 6 provides the details about software implementation including design, testing, parameters, platforms and problems encountered. Timing analysis are shown and discussed in Chapter 7. Finally, Chapter 8 provides concluding remarks and discussion about future research.

Note that RSA and EC cryptosystems are the two other popular public-key cryptosystems to provide key exchange and digital signature. In this thesis, only the DH and DSS are used for comparison because the exponentiation in DH and DSS can be considered as the k^{th} term generated by a first-order characteristic sequence.

Chapter 2

Public-key Cryptosystems and Third-order Characteristic Sequences

First we consider Diffie-Hellman (DH) key agreement protocol [5] and Digital Signature Standard (DSS) [20]. To have 1024-bit security, both DH and DSS require a 1024-bit prime p. Exponentiations are performed using the square-and-multiply algorithm.

The GH-PKC makes use of third-order characteristic sequences. Elements required to generate a third-order characteristic sequence will also be explained in this chapter. In order to understand the arithmetic in the next chapter, the idea of reciprocal sequence [10] will also be illustrated here.

2.1 Diffie-Hellman Key Agreement Protocol

The Diffie-Hellman (DH) key agreement protocol was developed by Diffie and Hellman in 1976 [20]. The protocol allows two users to agree on a secret key over an insecure medium without any prior shared secret.

System parameters for the DH key agreement protocol are public and are used by all users in the system. The system parameters include a prime number, p, and a primitive element, α, in GF(p). A primitive element α of GF(p) for a given p is an integer between 1 and $p-1$ that satisfies the following property: for every number n between 1 and $p-1$ inclusively, there is a power k of α such that $n = \alpha^k \pmod{p}$.

Suppose Alice and Bob want to agree on a shared key using the DH key agreement protocol. First, each user selects his/her private key from the set $\{1, \ldots, p-1\}$ which is co-prime with $p-1$. In other words, Alice and Bob choose their private keys x_A and x_B that satisfy:

$$0 < x_A, x_B < p-1, \gcd(x_A, p-1) = 1 \text{ and } \gcd(x_B, p-1) = 1.$$

Then they derive their public keys y_A and y_B using the system parameters p and α as:

$$y_A = \alpha^{x_A} \pmod{p} \qquad \text{and} \qquad y_B = \alpha^{x_B} \pmod{p}.$$

The private keys are used for public key generation only, and must be kept secret.

To initiate a session, Alice sends her public key to Bob. Bob replies by sending his

public key to Alice. Then Alice and Bob can compute their shared secret key as follows:

$$\begin{aligned} \textit{Alice}: \ {y_B}^{x_A} \pmod p &= (\alpha^{x_B})^{x_A} \pmod p = \alpha^{x_A x_B} \pmod p \\ \textit{Bob}: \ {y_A}^{x_B} \pmod p &= (\alpha^{x_A})^{x_B} \pmod p = \alpha^{x_A x_B} \pmod p. \end{aligned}$$

This shared key secret key is generated using their public keys transmitted through an insecure channel. Figure 2.1 summarizes this protocol.

System Parameters:

p, a prime number
α, a primitive element in GF(p)

	Alice	**Bob**
Secret Key:	x_A, with $0 < x_A < p-1$ and $gcd(x_A, p-1) = 1$	x_B, with $0 < x_B < p-1$ and $gcd(x_B, p-1) = 1$
Public Key:	$y_A = \alpha^{x_A} \ mod \ p$	$y_B = \alpha^{x_B} \ mod \ p$
	$\xrightarrow{y_A}$	
	$\xleftarrow{y_B}$	
Shared Secret Key:	${y_B}^{x_A} \ mod \ p$ $= (\alpha^{x_B})^{x_A} \ mod \ p$	${y_A}^{x_B} \ mod \ p$ $= (\alpha^{x_A})^{x_B} \ mod \ p$

Figure 2.1: DH Key Agreement Protocol

2.2 Digital Signature Standard

Digital Signature Algorithm was proposed by the US National Institute of Standards and Technology (NIST) in August 1991 and was specified as Digital Signature Standard(DSS) in a US Government Federal Information Processing Standards FIBS 186 [20] in May 1994. The DSS can be viewed as a variant of the ElGamal signature scheme [6].

System parameters for DSS are also public and are used by all users in the system. The system parameters include p which is the same as in the previous section. Another system parameter q is required which is a prime factor of $p-1$. According to the National Institute of Standards and Technology (NIST) standards [20], q should be a 160-bit prime. Instead of using the primitive element in the previous section, an element, g, in GF(p) with order q is used. It can be selected in the following way:

For an integer u with $0 < u < p$, if

$$u^{\frac{p-1}{q}} \pmod{p} \neq 1$$

then set g equals to

$$g = u^{\frac{p-1}{q}} \pmod{p}$$

such that

$$g^q \pmod{p} = 1.$$

This is derived from the fact that every number u between 1 and $p-1$ inclusively must satisfy:

$$u^{p-1} \pmod{p} = 1.$$

2.2.1 DSS Signature Generation

Suppose Alice would like to sign a message using DSS and send it to Bob. Alice first selects her private key, x, such that:

$$0 < x < q \text{ and } \gcd(x, q) = 1$$

and computes her public key, y, using the system parameters p and g as:

$$y = g^x \pmod{p}.$$

Then Alice randomly chooses a signing key, k, that satisfies:

$$0 < k < q \text{ and } \gcd(k, q) = 1$$

and computes

$$r = \alpha^k \pmod{p}.$$

Both the private key x and signing key k are used for signature generation only, and must be kept secret. The private key x can be used for a period of time. However, the signing key k must be regenerated for each signature.

Instead of signing the whole message m, she signs the hashed value of m instead. She computes:

$$h = h(m)$$

where $h(\cdot)$ is a hash function, such as MD5 [28] and SHA-1 [19], that Alice and Bob agree upon. Then, Alice solves for t in the signing equation:

$$h = kt + xr \pmod{q}.$$

Then (r, t) is a digital signature of the message m. The signing process is summarized below:

1. Randomly choose k, with $0 < k < q$ and $\gcd(k, q) = 1$.
2. Compute $r = g^k \pmod{p} \pmod{q}$.
3. Compute $h = h(m)$, where $h(\cdot)$ is a hash function and m is the message to be signed.
4. Solve for t in the signing equation $h = xr + kt \pmod{q}$.

Digital signature of message m is (r, t).

2.2.2 DSS Signature Verification

Bob verifies Alice's signature by checking if the signing equation is true. Since x and k are unknown to Bob, he cannot verify the equation simply by checking each parameters. Instead, he put both sides of the equation as powers of g as follows:

$$\begin{aligned} g^h &= g^{rx+kt} \\ &= g^{rx} g^{kt} \\ &= y^r r^t \end{aligned}$$

where g is one of the public system parameters, y is Alice's public key, h is the hashed value of message m and (r, t) is the digital signature of m. These parameters are known to Bob. The above equation involves three exponentiations. To minimize the number of

exponentiation, consider the equation as:

$$\begin{aligned} g^h &= y^r r^t \\ g^h y^{-r} &= r^t \\ g^{ht^{-1}} y^{-rt^{-1}} &= r \end{aligned}$$

which involves only two exponentiations. Bob accepts Alice's signature only if the above equation is true.

Notice that the private and public keys used in the DH key agreement protocol are different from the ones used in DSS. The private key x in the DH key agreement protocol is taken from the set $\{1, \ldots, p-1\}$ and the public key is computed as $\alpha^x \pmod p$, whereas in DSS, the private key x is taken from the set $\{1, \ldots, q-1\}$ and the public key is computed as $g^x \pmod p$. Therefore, each user has to keep two sets of private and public keys for DH key agreement protocol and for DSS respectively.

2.3 Third-order Characteristic Sequences

A third-order characteristic sequence is a sequence generated by an irreducible polynomial $f(x)$ of degree three over GF(p), where p is a prime, using specific initial states. This sequence can also be regarded as a linear feedback shift register (LFSR) sequence generated by $f(x)$.

An irreducible polynomial $f(x)$ of degree three over GF(p) has the form:

$$f(x) = x^3 - ax^2 + bx - 1$$

where the coefficients a and b are elements in GF(p). A third-order characteristic sequence $\{s_k(a,b)\}$, or just $\{s_k\}$ if its context is clear, generated by $f(x)$ is defined as:

$$s_{k+3} = as_{k+2} - bs_{k+1} + s_k, \text{for all } k \geq 0$$

where the initial state is chosen as:

$$s_0 = 3, s_1 = a, s_2 = a^2 - 2b.$$

Thus, s_k denotes the k^{th} term in the sequence. The k^{th} state is denoted as:

$$\underline{s}_k = (s_k, s_{k+1}, s_{k+2}).$$

The period of this characteristic sequence, Q, is a factor of $p^2 + p + 1$ and the maximum period is:

$$Q = p^2 + p + 1.$$

Any polynomial which has the form:

$$f_k(x) = x^3 - s_k(a,b)x^2 + s_{-k}(a,b)x - 1$$

is also an irreducible polynomial with the same period as $f(x)$ if and only if k and Q are co-prime.

2.4 Reciprocal Sequences

Given the irreducible polynomial $f(x)$ in Section 2.3, the reciprocal polynomial is:

$$f^{-1}(x) = x^3 - bx^2 + ax - 1.$$

By choosing the corresponding initial states as given in Section 2.3, the sequence generated by $f^{-1}(x)$ is also a third-order characteristic sequence and it is denoted as $\{s_k(b,a)\}$. This sequence is the reciprocal sequence of $\{s_k(a,b)\}$. The k^{th} term in the reciprocal sequence $\{s_k(b,a)\}$ is the same as the $-k^{th}$ term in the sequence $\{s_k(a,b)\}$. An example of third-order characteristic sequence is shown in Figure 2.2, where

$$s_{k+3} = -s_{k+1} + s_k \text{ and } s_{-(k+3)} = s_{-(k+1)} + s_{-k}, \text{ for } k = 0, 1, \ldots, 30$$

and the initial states are:

$$(s_0, s_1, s_2) = (3, 0, 3) \text{ and } (s_0, s_{-1}, s{-}2) = (3, 1, 1).$$

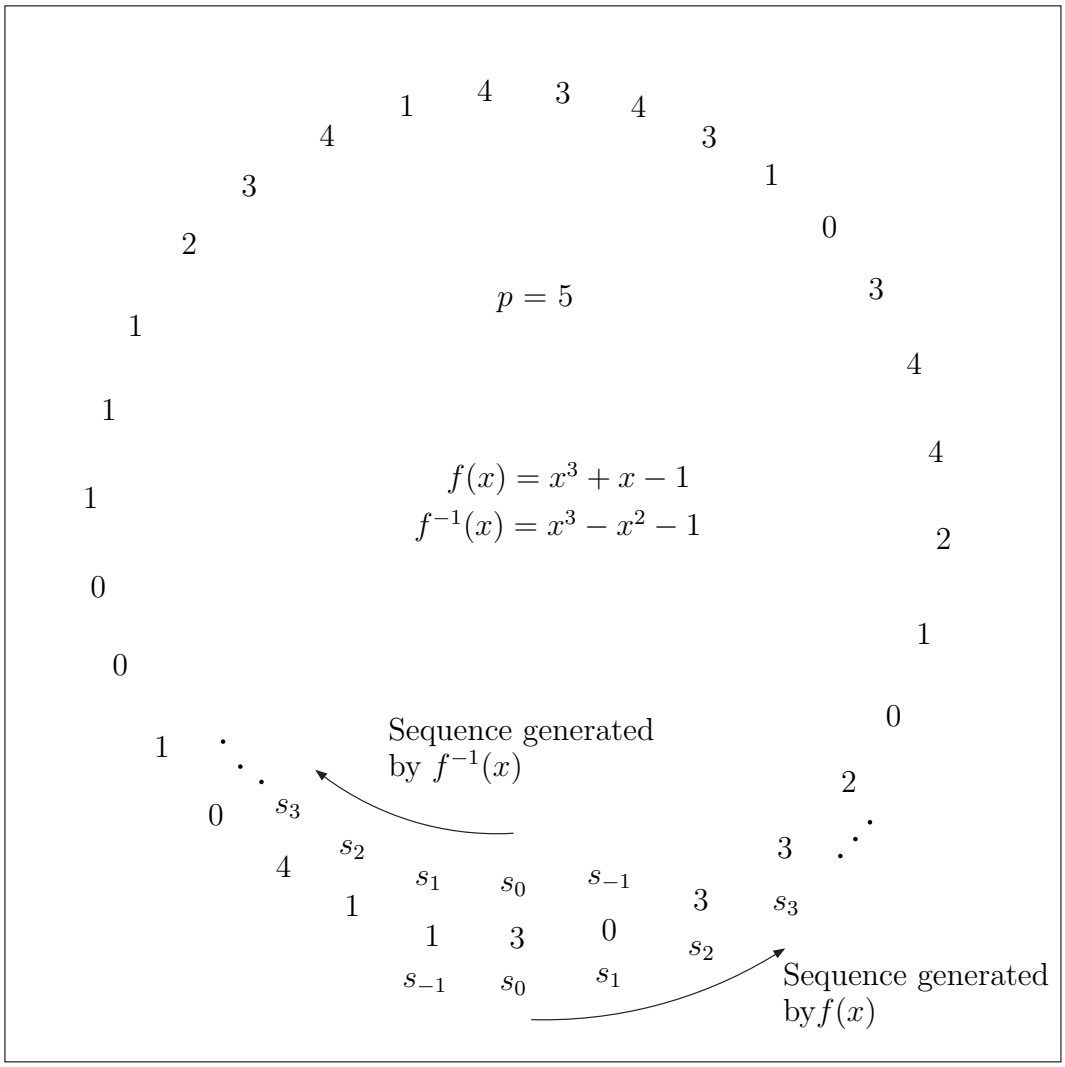

Figure 2.2: A third-order characteristic sequence

Chapter 3

Algorithms for Computing Sequence Terms

Three algorithms for computing the sequence terms in a third-order characteristic sequence given in references [10] and [11] are shown in this chapter.

3.1 Dual-State Fast Evaluation Algorithm (DSEA Algorithm)

The DSEA algorithm is an efficient algorithm for computing the $\pm k^{th}$ terms in a third-order characteristic sequence generated by an irreducible polynomial $f(x)$ over GF(p).

Basically, there are two sets of equations to be used depending on the bits in the binary representation of k.

First, express k in binary representation:

$$k = \sum_{i=0}^{n} k_i 2^{n-i} = k_0 2^n + k_1 2^{n-1} + \cdots + k_n$$

where $n+1$ is the number of bits required to represent k in the binary form and n can be calculated as:

$$n = \lfloor log_2 k \rfloor.$$

A set of variable T_j is defined as follows:

$$T_0 = k_0 = 1, T_j = k_j + 2T_{j-1},\ for\ 1 \leq j \leq n.$$

Since T_0 equals to k_0, which is the most significant bit in the binary representation of k, it must have the value of 1. Using the recursive equation given above to compute T_j, for j from 1 to n, the last value T_n equals to k.

To make it easier to see the terms in the two sets of equations in the algorithm, two variables are defined as:

$$t = T_{j-1},\ t' = T_j.$$

The two sets of equations to be used that depend on the bits in the binary representation of k are:
For $k_j = 0$:

$$\begin{aligned} s_{t'+1} &= s_t s_{t+1} - a s_{-t} + s_{-(t-1)} \\ s_{t'} &= s_t^2 - 2 s_{-t} \\ s_{t'-1} &= s_t s_{t-1} - b s_{-t} + s_{-(t+1)}. \end{aligned}$$

For $k_j = 1$:

$$\begin{aligned} s_{t'+1} &= s_{t+1}^2 - 2s_{-(t+1)} \\ s_{t'} &= s_t s_{t+1} - a s_{-t} + s_{-(t-1)} \\ s_{t'-1} &= s_t^2 - 2s_{-t}. \end{aligned}$$

By having a for-loop for j from 1 to n, depending on the value of k_j, the corresponding set of equations will be chosen in the iteration to compute $(s_{t'+1}, s_{t'}, s_{t'-1})$ terms. Notice that in the two sets of equations, there are some terms that belong to the reciprocal sequence $\{s_k(b,a)\}$ such as s_{-t}. The s_t term used in the j^{th} iteration is the $s_{t'}$ term computed in the $(j-1)^{th}$ iteration. Similarly, the s_{-t} term used in the j^{th} iteration is the $s_{-t'}$ term computed in the $(j-1)^{th}$ iteration. The $s_{-t'}$ term can be computed by interchanging a and b to form the reciprocal polynomial $f^{-1}(x)$ and by interchanging all s_k terms with s_{-k} terms. Therefore, in each iteration, the corresponding set of equations need to be executed twice to obtain the $(s_{t'+1}, s_{t'}, s_{t'-1})$ and $(s_{-(t'+1)}, s_{-t'}, s_{-(t'-1)})$ terms. After the last iteration is performed, the $s_{t'}$ and the $s_{-t'}$ terms are the s_k and the s_{-k} terms respectively.

The corresponding set of equations are executed twice in each iteration and there are n iterations in total. The number of multiplications in the sets of equations corresponding to k_j equals 0 and k_j equals 1 are five and four respectively. On average, the probability that k_j equals 0 or 1 is $\frac{1}{2}$. Therefore, on average, the total number of multiplications in GF(p) to obtain $s_{\pm k}$ is:

$$2 \cdot n \cdot [Pr(k_j = 0) \cdot 5 + Pr(k_j = 1) \cdot 4] = 2 \cdot \lfloor log_2 k \rfloor \cdot \left(\frac{1}{2} \cdot 5 + \frac{1}{2} \cdot 4 \right) = 9 \lfloor log_2 k \rfloor.$$

3.2 Computation of a Previous Sequence Term

In paper [11], it is proven that the third-order characteristic sequence has the properties of duality and redundancy. Three elements in any state of the third-order characteristic sequence are not independent. If any two consecutive elements are known, the third remaining one can be uniquely determined as follows.

Let

$$\Delta = s_{k+1}s_{-(k+1)} - s_1 s_{-1}.$$

Then $s_{\pm(k-1)}$ can be computed as:

$$\begin{aligned} s_{k-1} &= \frac{es_{-(k+1)} - s_{-1}D(e)}{\Delta} \\ s_{-(k-1)} &= \frac{D(e)s_{k+1} - s_1 e}{\Delta} \end{aligned}$$

where

$$\begin{aligned} D(s_k) &= s_{-k} \\ e &= -s_{-1}D(c_1) + c_2 \\ c_1 &= s_1 s_{k+1} - s_{-1}s_k \\ c_2 &= s_k^2 - 3s_{-k} + (b^2 - a)s_{-(k+1)}. \end{aligned}$$

Since Δ appears in the denominator in the above equations, Δ cannot be zero.

3.3 Computation of a Mixed Term

The procedure to compute a mixed term $s_{\pm u(k+v)}$ with known $\underline{s}_{\pm(k-1)}$ state requires the basics of sequence theory. First, compute the sequence term s_{k+v}. Then, construct another irreducible polynomial as:

$$g(x) = x^3 - s_{k+v}x^2 + s_{-(k+v)}x - 1$$

and compute the $\pm u^{th}$ sequence terms generated by $g(x)$ using the DSEA algorithm. This gives $s_{\pm u(k+v)}$ terms in the sequence generated by $f(x)$.

In the paper about GH-DSA[11], it is stated that $s_{\pm(k+v)}$ terms can be computed using the DSEA algorithm. However, it is not true. In order to compute $s_{\pm(k+v)}$ terms, general results of LFSR sequences should be used. The algorithm for computing s_{k+v} terms is given in reference [12].

Define a transitional matrix, matrix A, as:

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & -b \\ 0 & 1 & a \end{bmatrix}.$$

When the k^{th} state is multiplied by matrix A, it gives:

$$\begin{aligned} \underline{s}_k \cdot A &= (s_k, s_{k+1}, s_{k+2}) \cdot \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & -b \\ 0 & 1 & a \end{bmatrix} \\ &= (s_{k+1}, s_{k+2}, s_k - bs_{k+1} + as_{k+2}) \\ &= \underline{s}_{k+1} \end{aligned}$$

which is the next state of the LFSR.

Define a state matrix, matrix M_n, as:

$$M_n = \begin{bmatrix} s_{n-2} & s_{n-1} & s_n \\ s_{n-1} & s_n & s_{n+1} \\ s_n & s_{n+1} & s_{n+2} \end{bmatrix}.$$

The following properties are given in reference [12]:

1. $\underline{s}_v = \underline{s}_0 \cdot A^v = \underline{s}_1 \cdot A^{v-1} = \cdots = \underline{s}_{v-1} \cdot A,\ v = 1, 2, \ldots.$
2. $M_v = M_0 \cdot A^v \Rightarrow A^v = M_0^{-1} \cdot M_v$, if $\det(M_0) \neq 0$.

In general,

$$\begin{aligned} \underline{s}_{k+v} &= \underline{s}_k \cdot A^v = \underline{s}_k \cdot (M_0^{-1} \cdot M_v) \\ &= \underline{s}_k \cdot \left(\begin{bmatrix} s_{-2} & s_{-1} & s_0 \\ s_{-1} & s_0 & s_1 \\ s_0 & s_1 & s_2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} s_{v-2} & s_{v-1} & s_v \\ s_{v-1} & s_v & s_{v+1} \\ s_v & s_{v+1} & s_{v+2} \end{bmatrix} \right), \text{ if } \det(M_0) \neq 0. \end{aligned}$$

In particular, the s_{k+v} term is equal to $\underline{s}_k$ multiplies to the first column of $(M_0^{-1} \cdot M_v)$. Since state $\underline{s}_{k-1}$ is known instead of state $\underline{s}_k$, the s_{k+v} term is equal to $\underline{s}_{k-1}$ multiplied by the middle column of $(M_0^{-1} \cdot M_v)$. To construct the matrix M_v, s_{v-2}, s_{v-1}, s_v, s_{v+1} and s_{v+2} are required. Sequence terms (s_{v-1}, s_v, s_{v+1}) can be obtained using the DSEA algorithm. The terms s_{v-2} and s_{v+2} can be obtained by:

$$\begin{aligned} s_{v+2} &= a s_{v+1} - b s_v + s_{v-1} \\ s_{v-2} &= s_{v+1} - a s_v + b s_{v-1}. \end{aligned}$$

Similarly, it can be done for the $s_{-(k+v)}$ term.

The mixed term algorithm is summarized below:

1. Compute $\underline{s}_{v-1}$ state and its dual generated by $f(x)$ using the DSEA algorithm.
2. Construct Matrix M_0 and compute M_0^{-1}.
3. Compute s_{v-2} and s_{v+2} terms and construct Matrix M_v.
4. Compute s_{k+v} term by multiplying $\underline{s}_{k-1}$ by the middle column of $M_0^{-1} \cdot M_v$.
5. Repeat step 2 to 4 to compute $s_{-(k+v)}$ term.
6. Construct $g(x) = x^3 - s_{k+v}x^2 + s_{-(k+v)}x - 1$ and compute the $\pm u^{th}$ terms generated by $g(x)$ using the DSEA algorithm.

Chapter 4

Gong-Harn Public-Key Cryptosystem

The GH-PKC is explained in this chapter including GH Diffie-Hellman key agreement protocol and GH digital signature algorithm.

4.1 GH Diffie-Hellman (GH-DH) Key Agreement Protocol

The GH-DH key agreement protocol is basically a Diffie-Hellman-like key agreement protocol using sequence terms. The DSEA algorithm was developed to compute these

sequence terms efficiently. A pair of common keys between two users is generated by polynomials using each other's public key pair as coefficients.

The system parameters include a prime number p and an irreducible polynomial $f(x) = x^3 - ax^2 + bx - 1$ over GF(p) which are public and are used by all users in the system.

Suppose Alice and Bob want to agree on a shared key using the GH-DH key agreement protocol. First, they choose their own private keys x_A and x_B that satisfy:

$$0 < x_A, x_B < p^2 + p + 1, \gcd(x_A, p^2 + p + 1) = 1 \; and \; \gcd(x_B, p^2 + p + 1) = 1$$

to ensure the polynomial they generate later is also irreducible over GF(p) with the same period as the period of the sequence generated by $f(x)$.

Alice and Bob compute the $\pm x_A^{th}$ and $\pm x_B^{th}$ sequence terms generated by $f(x)$, namely $s_{\pm x_A}$ and $s_{\pm x_B}$, as their public keys using the DSEA algorithm.

They send their public keys to each other to form polynomials $f_A(x)$ and $f_B(x)$ using the received public keys as the coefficients. The $\pm x_A^{th}$ and $\pm x_B^{th}$ terms in the sequences generated by $f_A(x)$ and $f_B(x)$ respectively are the common key pair between the two users. For example, Alice forms a polynomial:

$$f_A(x) = x^3 - s_{+x_B}x^2 + s_{-x_B}x - 1$$

and generates the common key pair by computing the $\pm x_A^{th}$ terms in the sequence generated by $f_A(x)$, namely $s_{\pm x_A}(s_{x_B}, s_{-x_B})$. Figure 4.1 summarizes this protocol.

System Parameters:

$f(x) = x^3 - ax^2 + bx - 1$, an irreducible polynomial over GF(p), where p is a prime number
$Q = p^2 + p + 1$

	Alice	**Bob**
Secret Key:	x_A, with $0 < x_A < Q$ and $gcd(x_A, Q) = 1$	x_B, with $0 < x_B < Q$ $gcd(x_B, Q) = 1$
Public Key Pair:	(s_{x_A}, s_{-x_A})	(s_{x_B}, s_{-x_B})
	$\xrightarrow{(s_{x_A}, s_{-x_A})}$	
	$\xleftarrow{(s_{x_B}, s_{-x_B})}$	
Polynomial generated using the received public key pair:	$f_A(x) = x^3 - s_{x_B}x^2 + s_{-x_B}x - 1$	$f_B(x) = x^3 - s_{x_A}x^2 + s_{-x_A}x - 1$
Common Key Pair:	$\pm x_A$ terms generated by $f_A(x)$ is $s_{\pm x_A}(s_{x_B}, s_{-x_B})$	$\pm x_B$ terms generated by $f_B(x)$ is $s_{\pm x_B}(s_{x_A}, s_{-x_A})$

$$s_{\pm x_A}(s_{x_B}, s_{-x_B}) = s_{\pm x_B}(s_{x_A}, s_{-x_A})$$

Figure 4.1: GH-DH Key Agreement Protocol

4.2 GH Digital Signature Algorithm (GH-DSA)

The GH-DSA is similar to the ElGamal signature algorithm. In this thesis, Alice is assumed to be the signer and Bob is assumed to be the verifier. The system parameters p, a and b are the same as the last section. Q for the GH-DSA is taken as a factor of $p^2 + p + 1$. Alice chooses her private key, x, that satisfies:

$$0 < x < Q \; and \; \gcd(x, Q) = 1$$

and computes her public key pair as detailed in the last section.

4.2.1 Signing Process

Alice randomly chooses a signing key, k, that satisfies:

$$0 < k < Q \; and \gcd(k, Q) = 1$$

and computes (s_{k-1}, s_k, s_{k+1}) and its dual using the DSEA algorithm, such that s_k is co-prime with Q. If s_k is not co-prime with Q, Alice should select another k.

To sign a message, m, Alice computes the hash value of m:

$$h = h(m)$$

where $h(\cdot)$ is a hash function, such as MD5 [28] and SHA-1 [19], that Alice and Bob agree upon. Then, Alice solves for t in the signing equation:

$$h = kt + xr \pmod{Q}$$

where r equals to the sequence term s_k. Then (r, t) is a digital signature of the message m.

Alice sends Bob (m, r, t), together with (s_x, s_{x+1}), (s_k, s_{k+1}) and their duals. Notice that only (s_k, s_{k+1}) are sent instead of (s_{k-1}, s_k, s_{k+1}). Since the previous sequence term can be computed using the two consecutive sequence terms as described in Section 3.2, it is not necessary for Alice to send all three terms to the verifier. Therefore, only (s_k, s_{k+1}) are sent to minimize the use of bandwidth.

The signing procedure is summarized below:

1. Compute public keys.

2. Randomly choose a signing key k with $0 < k < Q$ and $\gcd(k, Q) = 1$. Compute (s_{k-1}, s_k, s_{k+1}) and its dual using the DSEA algorithm. Set $r = s_k$ if $\gcd(s_k, Q) = 1$.
3. Compute $h = h(m)$ where m is the message and $h(\cdot)$ is a hash function.
4. Compute $t = k^{-1}(h - rx) \pmod{Q}$.

Digital signature of message m is (r, t).

4.2.2 Verification Procedure

In the verification procedure, there are two different cases in verifying Alice's signature that Bob has to choose from depending on whether t and Q are co-prime.

Case 1: $\gcd(t, Q) = 1$.

In the signing equation, k and x are unknown to Bob, therefore, Bob cannot simply verify the equation by checking each parameter. However, Bob has Alice public key pair (s_x, s_{-x}). Instead of verifying each parameter in the signing equation, Bob can put both sides of the signing equation as sequence term indices and verify to see if these sequence terms match:

$$s_{r^{-1}(h-kt)} = s_x \ and \ s_{-r^{-1}(h-kt)} = s_{-x}.$$

Let

$$u = -r^{-1}t \pmod{Q} \ and \ v = -ht^{-1} \pmod{Q}$$

where h is the hash value of m using the same hash function as in the signing process.

Then,

$$s_x = s_{-r^{-1}t(k-ht^{-1})} = s_{u(k+v)} \; and \; s_{-x} = s_{r^{-1}t(k-ht^{-1})} = s_{-u(k+v)}.$$

Bob first computes $s_{\pm(k-1)}$ terms using the algorithm given in Section 3.2. Then he computes u and v as shown above. By using the mixed term algorithm given in Section 3.3, he can compute $s_{\pm u(k+v)}$ sequence terms and verify to see if they are the same as $s_{\pm x}$. If they match, then Bob accepts Alice's signature. Case 1 is summarized as:

1. Compute $s_{\pm(k-1)}$ using (s_k, s_{k+1}) and its duals.
2. Compute $u = -r^{-1}t \pmod{Q}$ *and* $v = -ht^{-1} \pmod{Q}$.
3. Compute $s_{\pm u(k+v)}$.
4. Accept signature if $s_{\pm u(k+v)} = s_{\pm x}$.

Case 2: $\gcd(t, Q) \neq 1$.

If t and Q are not co-prime, $t^{-1} \pmod{Q}$ does not exist; therefore, Bob cannot use the same algorithm as in Case 1. Instead, Bob can verify to see if these sequence terms match:

$$s_{h-rs} = s_{kt} \; and \; s_{-(h-rs)} = s_{-kt}.$$

Let

$$u = -r \pmod{Q} \; and \; v = -r^{-1}h \pmod{Q}$$

where h is the hash value of m using the same hash function as in the signing process.

Then,

$$s_{kt} = s_{u(x+v)} = s_{-r(x-r^{-1}k)} \; and \; s_{-kt} = s_{-u(x+v)} = s_{r(x-r^{-1}k)}.$$

Bob first computes $s_{\pm(x-1)}$ terms using the algorithm given in Section 3.2. Then he computes u, v as shown above. By using the mixed term algorithm given in Section 3.3, Bob can compute $s_{\pm u(x+v)}$ and $s_{\pm kt}$ sequence terms. If these sequence terms match, then Bob accepts Alice's signature. Case 2 is summarized as:

1. Compute $s_{\pm(x-1)}$ using (s_x, s_{x+1}) and its duals.
2. Compute $u = -r \pmod{Q}$ *and* $v = -r^{-1}h \pmod{Q}$.
3. Compute $s_{\pm u(x+v)}$.
4. Compute $s_{\pm kt}$.
5. Accept signature if $s_{\pm u(x+v)} = s_{\pm kt}$.

Similarly to that described at the end of Section 2.2, the private and the public keys used in the GH-DH key agreement protocol are different from the ones used in the GH-DSA. The private key x in GH-DH is taken from the set $\{1, \ldots, p^2 + p\}$, whereas in GH-DSA, the private key x is taken from the set $\{1, \ldots, Q - 1\}$ and Q is a factor of $p^2 + p + 1$. Therefore, each user has to keep two sets of private and public keys for GH-DH key agreement protocol and for GH-DSA.

Chapter 5

Design Issues

Three design issues arise during the process of software design: the storage requirement for recursive iterations in the DSEA algorithm, the percentage of zero Δ and the matrix inverse.

5.1 Storage Requirement for DSEA Algorithm

As stated in Section 3.1, since a for-loop from 1 to n is needed to choose the corresponding set of equations to use depending on the value of k_j, it is necessary to consider the storage requirement for this for-loop. In the two sets of equations as shown in Section 3.1, the six sequence terms to be computed in each iteration are $(s_{\pm(t'+1)}, s_{\pm t'}, s_{\pm(t'-1)})$ which require the use of sequence terms $(s_{\pm(t+1)}, s_{\pm t}, s_{\pm(t-1)})$, which are the six sequence terms generated

in the previous iteration. Therefore, after the sequences terms $(s_{\pm(t'+1)}, s_{\pm t'}, s_{\pm(t'-1)})$ are computed, the terms $(s_{\pm(t+1)}, s_{\pm t}, s_{\pm(t-1)})$ can be discarded as they will not be used in proceeding iterations. Hence, instead of storing the whole sequence, only six values need to be kept after each iteration.

5.2 The Percentage of Zero Δ

As described in Section 4.2.1, the signer, Alice, only sends (s_k, s_{k+1}) and its dual to the verifier, Bob, in order to reduce the use of bandwidth. Bob should computethe $s_{\pm(k-1)}$ terms using the algorithm described in Section 3.2. Alice, as the signer, is responsible for choosing a value for k such that Δ is not zero. It is important to investigate the percentage of Δ that equals zero to see the tradeoff between bandwidth usage and time required to choose a valid k. Since there is no theoretical result about the distribution of zero delta, experiments are performed for this investigation.

Programs written in C++ and Maple are used to compute the value of Δ for all sequence terms generated by all irreducible polynomials of degree three over GF(p) for a given p. When Δ equals zero, it means that:

$$s_{k+1}s_{-(k+1)} = s_1 s_{-1} = ab.$$

If sequence terms are organized in two rows as shown in Figure 5.1, the product $s_{k+1}s_{-(k+1)}$ for a given k is the product of the two sequence terms in the same column. For example, if k equals to 2, the product $s_{k+1}s_{-(k+1)}$ is:

$$s_3 s_{-3} = s_3 s_{Q-3} = s_{Q-3}s_{-(Q-3)}$$

which is the same product for k equals to $(Q-4)$. Therefore, it is sufficient to compute Δ for k from 1 to $(Q-1)/2$ only.

Prime numbers, p, between 5 and 127 are used in the testing and the results are analyzed. Results for prime p between 5 and 17 are provided in Table 5.1- 5.8. The percentage of Δ equals zero for p between 5 and 127 is summarized in Table 5.9.

It is found that for any prime p,

$$\begin{aligned} s_p &= a = s_{-(p+1)} \\ s_{p+1} &= b = s_{-p}. \end{aligned}$$

In other words, if k equals to either $p-1$ or p, the product $s_{k+1}s_{-(k+1)}$ equals to $a \cdot b$ for any prime p. This restriction is added to the signing key selection.

Besides, if either a or b is zero, k will more likely to yield a zero Δ. This is due to the fact that $a \cdot b$ equals zero if either a or b is zero. A zero Δ will result if either s_{k+1} or $s_{-(k+1)}$ equals zero. To avoid this, system parameters should be chosen such that both a and b are non-zero integers. However, if either a or b equals zero, the total number of multiplications in the DSEA algorithm explained in Section 3.1 can be reduced to $7\lfloor log_2 k \rfloor$ and $8\lfloor log_2 k \rfloor$ respectively. This is a tradeoff between computational cost and bandwidth usage. If the computational cost is more expensive than the bandwidth usage, either a or b should be chosen to be zero. In that case, the restriction on k such that Δ does not equal to zero can be removed as the signer sends (s_{k-1}, s_k, s_{k+1}) and its duals to the receiver.

Moreover, from the result, as p increases, the percentage of Δ that equals zero de-

creases in most cases. The percentage of zero Δ for $p = 127$ is only 0.08%. The percentage is low enough to conclude that the cost of re-selecting values for k is lower than the cost associated with increasing the bandwidth usage. Some of the sequences have a short period for certain p, such as $p = 19$, which causes an increase in the percentage of Δ being zero. In choosing system parameters a and b, the sequence period should be considered to minimize the chance of Δ equalling zero.

With the additional restrictions on system parameters and the additional rules on selecting k, it is reasonable to reduce bandwidth usage as a way of minimizing the cost.

5.3 The Matrix Inverse

The algorithm for computing mixed term given in Section 3.3 requires the inverse of Matrix M_0. However, not all matrices are invertible. Matrices are invertible if and only if their determinants do not equal to zero.

The matrix M_0 is:

$$M_0 = \begin{bmatrix} s_{-2} & s_{-1} & s_0 \\ s_{-1} & s_0 & s_1 \\ s_0 & s_1 & s_2 \end{bmatrix} = \begin{bmatrix} b^2 - 2a & b & 3 \\ b & 3 & a \\ 3 & a & a^2 - 2b \end{bmatrix}$$

which is invertible if and only if its determinant is not zero, i.e.:

$$det(M_0) = (b^2 - 2a)[3(a^2 - 2b) - a^2] - b[b(a^2 - 2b) - 3a] + 2[ab - 3 \cdot 3] \neq 0.$$

Since it depends only on a and b, the determinant of M_0 can be guaranteed to be non-zero by choosing a and b correspondingly.

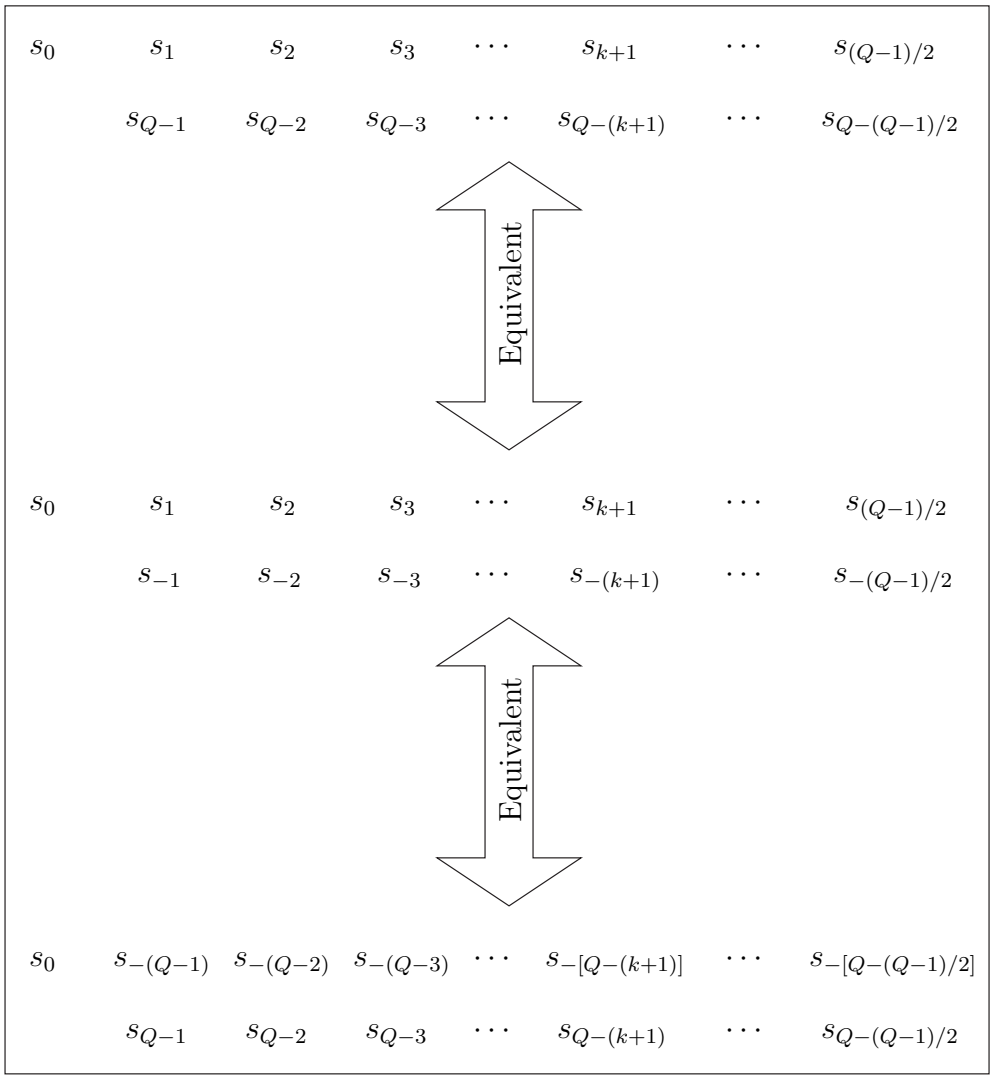

Figure 5.1: Organizing Sequence Terms

Table 5.1: Distribution of Zero Δ for $p = 5$

(a, b)	Seq Period	k values	Percentage of zero Δ
(4, 1)	31	4 5	13.33
(4, 3)	31	4 5	13.33
(0, 2)	31	4 5 7 8 13	33.33
(0, 1)	31	3 4 5 6 10	33.33
(1, 4)	31	4 5	13.33
(3, 1)	31	4 5	13.33
(1, 0)	31	3 4 5 6 10	33.33
(3, 4)	31	4 5	13.33
(2, 0)	31	4 5 7 8 13	33.33
(1, 3)	31	4 5	13.33

Table 5.2: Distribution of Zero Δ for $p = 7$

(a, b)	Seq Period	k values	Percentage of zero Δ
(4, 0)	57	6 7 10 11 17 19 25 26	28.57
(2, 6)	57	6 7 11 17 26	17.86
(4, 3)	19	6 7	22.22
(6, 4)	57	6 7	7.14
(5, 1)	57	6 7	7.14
(3, 1)	19	6 7	22.22
(0, 4)	57	6 7 10 11 17 19 25 26	28.57
(2, 5)	57	6 7	7.14
(1, 0)	57	6 7 10 11 17 19 25 26	28.57
(0, 2)	19	6 7	22.22
(1, 3)	19	6 7	22.22
(6, 2)	57	6 7 11 17 26	17.86
(1, 5)	57	6 7	7.14
(5, 2)	57	6 7	7.14
(3, 4)	19	6 7	22.22
(4, 6)	57	6 7	7.14
(2, 0)	19	6 7	22.22
(0, 1)	57	6 7 10 11 17 19 25 26	28.57

Table 5.3: Distribution of Zero Δ for $p = 11$

(a, b)	Seq Period	k values	Percentage of zero Δ
(5, 8)	133	10 11	3.03
(9,10)	133	10 11	3.03
(8,10)	133	10 11	3.03
(6, 5)	133	10 11	3.03
(8, 0)	133	4 10 11 15 39 40 42 51 54 58 59	16.67
(0, 7)	133	9 10 11 12 17 22 26 30 49 57 64	16.67
(8, 6)	19	6 7	22.22
(4, 2)	133	10 11	3.03
(0, 4)	133	7 10 11 17 28 36 44 49 50 52 64	16.67
(9, 6)	133	10 11	3.03
(8, 5)	133	10 11	3.03
(8, 9)	19	6 7	22.22
(9, 3)	133	10 11	3.03
(1, 7)	133	10 11	3.03
(7, 6)	133	10 11	3.03
(3, 5)	133	10 11	3.03
(4, 6)	7	1 2	66.67
(3, 7)	133	10 11	3.03
(6, 9)	133	10 11	3.03
(10, 9)	133	10 11	3.03
(3, 1)	133	10 11	3.03
(2,10)	19	6 7	22.22
(0, 5)	133	7 8 10 11 24 29 33 36 38 44 63	16.67
(9, 2)	133	10 11	3.03
(4, 0)	133	7 10 11 17 28 36 44 49 50 52 64	16.67
(9, 8)	19	6 7	22.22
(10, 8)	133	10 11	3.03
(6,10)	133	10 11	3.03
(2, 4)	133	10 11	3.03
(7, 3)	133	10 11	3.03
(3, 9)	133	10 11	3.03
(5, 6)	133	10 11	3.03
(6, 8)	19	6 7	22.22
(6, 4)	7	1 2	66.67
(1, 3)	133	10 11	3.03
(7, 1)	133	10 11	3.03
(0, 8)	133	4 10 11 15 39 40 42 51 54 58 59	16.67
(7, 0)	133	9 10 11 12 17 22 26 30 49 57 64	16.67
(5, 3)	133	10 11	3.03
(2, 9)	133	10 11	3.03
(10, 2)	19	6 7	22.22
(6, 7)	133	10 11	3.03
(5, 0)	133	7 8 10 11 24 29 33 36 38 44 63	16.67
(10, 6)	133	10 113	3.03

Table 5.4: Distribution of Zero Δ for $p = 13$

(a, b)	Seq Period	k values	Percentage of zero Δ
(1, 6)	183	12 13 47 59 74	5.49
(2, 8)	183	12 13	2.20
(12, 6)	61	12 13	6.67
(1, 8)	183	12 13 18 63 82	5.49
(9,11)	183	12 13	2.20
(2,12)	61	12 13	6.67
(1, 4)	183	12 13	2.20
(11,10)	183	12 13	2.20
(7, 3)	61	12 13	6.67
(7,12)	183	12 13	2.20
(2, 1)	183	12 13	2.20
(6,10)	61	12 13	6.67
(6, 1)	183	12 13 47 59 74	5.49
(10, 0)	183	12 13 46 47 59 61 73 74	8.79
(6, 4)	183	12 13 47 59 74	5.49
(4, 8)	61	12 13	6.67
(4, 5)	183	12 13 46 61 73	5.49
(12, 0)	183	12 13 46 47 59 61 73 74	8.79
(5, 3)	61	12 13	6.67
(2,10)	183	12 13 46 61 73	5.49
(10, 5)	183	12 13	2.20
(3,10)	61	12 13	6.67
(10, 6)	61	12 13	6.67
(8, 2)	183	12 13	2.20
(12, 9)	183	12 13	2.20
(9, 6)	183	12 13	2.20
(5,11)	61	12 13	6.67
(2, 4)	183	12 13 19 76 85	5.49
(7, 6)	183	12 13	2.20
(0, 4)	61	12 13	6.67
(1, 2)	183	12 13	2.20
(0,10)	183	12 13 46 47 59 61 73 74	8.79
(6,12)	61	12 13	6.67
(12, 5)	183	12 13 47 59 74	5.49
(9, 5)	183	12 13	2.20
(2, 3)	61	12 13	6.67
(12, 7)	183	12 13	2.20
(10, 3)	61	12 13	6.67
(4, 6)	183	12 13 47 59 74	5.49
(8, 1)	183	12 13 18 63 82	5.49

Table 5.5: Distribution of Zero Δ for $p = 13$ (cont.)

(a, b)	Seq Period	k values	Percentage of zero Δ
(3, 7)	61	12 13	6.67
(5,10)	183	12 13	2.20
(9,12)	183	12 13	2.20
(8, 4)	61	12 13	6.67
(11, 9)	183	12 13	2.20
(10,11)	183	12 13	2.20
(10, 2)	183	12 13 46 61 73	5.49
(4, 0)	61	12 13	6.67
(6, 9)	183	12 13	2.20
(5, 4)	183	12 13 46 61 73	5.49
(12, 2)	61	12 13	6.67
(4, 1)	183	12 13	2.20
(3, 5)	61	12 13	6.67
(6, 7)	183	12 13	2.20
(5,12)	183	12 13 47 59 74	5.49
(11, 5)	61	12 13	6.67
(0,12)	183	12 13 46 47 59 61 73 74	8.79
(4, 2)	183	12 13 19 76 85	5.49
(3, 2)	61	12 13	6.67
(5, 9)	183	12 13	2.20

Table 5.6: Distribution of Zero Δ for $p = 17$

(a, b)	Seq Period	k values	Percentage of zero Δ
(1, 5)	307	16 17	1.31
(8, 6)	307	5 16 17 101 107	3.27
(6,11)	307	16 17	1.31
(1, 3)	307	16 17	1.31
(13,10)	307	16 17	1.31
(14, 7)	307	16 17	1.31
(1,11)	307	16 17	1.31
(12, 7)	307	16 17 45 92 138	3.27
(4,14)	307	16 17	1.31
(13, 6)	307	16 17	1.31
(5, 6)	307	16 17 19 32 52	3.27
(12, 4)	307	2 16 17 50 53	3.27
(0, 3)	307	5 13 14 16 17 36 42 51 54 61 68 101 107 111 116 132 146	11.11
(13, 0)	307	6 7 12 16 17 20 29 49 70 72 73 85 103 118 125 135 143	11.11
(8, 1)	307	16 17	1.31
(11, 8)	307	16 17	1.31
(5, 1)	307	16 17	1.31
(4,10)	307	16 17	1.31
(5, 9)	307	16 17	1.31
(13, 9)	307	16 17	1.31
(9,12)	307	16 17	1.31
(0, 9)	307	7 16 17 21 22 24 66 79 83 88 94 106 117 131 135 142 143	11.11
(2, 8)	307	16 17	1.31
(11, 9)	307	16 17	1.31
(1,12)	307	16 17 38 48 87	3.27
(16, 8)	307	16 17	1.31
(5, 3)	307	16 17	1.31
(11, 2)	307	16 17	1.31
(2,15)	307	16 17 45 92 138	3.27
(3, 8)	307	11 16 17 90 102	3.27
(8,11)	307	16 17	1.31
(6, 8)	307	5 16 17 101 107	3.27
(3,13)	307	16 17	1.31
(13, 7)	307	16 17	1.31
(1,13)	307	16 17	1.31
(7, 3)	307	16 17	1.31
(15, 9)	307	16 17	1.31
(15, 4)	307	16 17	1.31
(15,14)	307	16 17	1.31
(6, 7)	307	16 17	1.31

Table 5.7: Distribution of Zero Δ for $p = 17$ (cont.)

(a, b)	Seq Period	k values	Percentage of zero Δ
(14, 8)	307	16 17	1.31
(16,13)	307	16 17	1.31
(1, 8)	307	16 17	1.31
(10, 4)	307	16 17	1.31
(11, 6)	307	16 17	1.31
(2,16)	307	16 17	1.31
(9,14)	307	16 17	1.31
(15,16)	307	16 17	1.31
(6,14)	307	16 17 19 32 52	3.27
(8, 0)	307	12 16 17 39 40 60 65 72 78 82 85 105 112 114 115 123 129	11.11
(10, 2)	307	16 17	1.31
(10, 7)	307	16 17	1.31
(0,13)	307	6 7 12 16 17 20 29 49 70 72 73 85 103 118 125 135 143	11.11
(8, 3)	307	11 16 17 90 102	3.27
(9, 5)	307	16 17	1.31
(3, 1)	307	16 17	1.31
(2,12)	307	16 17 26 127 151	3.27
(0,10)	307	2 16 17 28 41 43 44 50 53 91 99 110 120 128 133 141 150	11.11
(0,11)	307	9 16 17 24 29 61 73 78 103 111 112 114 117 126 132 136 142	11.11
(2, 6)	307	16 17 62 93 149	3.27
(3, 0)	307	5 13 14 16 17 36 42 51 54 61 68 101 107 111 116 132 146	11.11
(15, 2)	307	16 17 45 92 138	3.27
(13, 3)	307	16 17	1.31
(9,13)	307	16 17	1.31
(10,13)	307	16 17	1.31
(9, 1)	307	16 17	1.31
(4,12)	307	2 16 17 50 53	3.27
(2,11)	307	16 17	1.31
(13,16)	307	16 17	1.31
(7,13)	307	16 17	1.31
(12, 9)	307	16 17	1.31
(7,14)	307	16 17	1.31
(6, 5)	307	16 17 19 32 52	3.27
(3, 5)	307	16 17	1.31
(8,14)	307	16 17	1.31
(16, 2)	307	16 17	1.31
(13, 1)	307	16 17	1.31
(9, 0)	307	7 16 17 21 22 24 66 79 83 88 94 106 117 131 135 142 143	11.11
(11, 1)	307	16 17	1.31
(6,13)	307	16 17	1.31

Table 5.8: Distribution of Zero Δ for $p = 17$ (cont.)

(a, b)	Seq Period	k values	Percentage of zero Δ
(8,16)	307	16 17	1.31
(7, 6)	307	16 17	1.31
(7,10)	307	16 17	1.31
(14, 9)	307	16 17	1.31
(3, 7)	307	16 17	1.31
(8, 2)	307	16 17	1.31
(7,12)	307	16 17 45 92 138	3.27
(14, 4)	307	16 17	1.31
(12, 1)	307	16 17 38 48 87	3.27
(14,15)	307	16 17	1.31
(2,10)	307	16 17	1.31
(12, 2)	307	16 17 26 127 151	3.27
(16,15)	307	16 17	1.31
(9,15)	307	16 17	1.31
(9,11)	307	16 17	1.31
(4,15)	307	16 17	1.31
(0, 8)	307	12 16 17 39 40 60 65 72 78 82 85 105 112 114 115 123 129	11.11
(6, 2)	307	16 17 62 93 149	3.27
(10, 0)	307	2 16 17 28 41 43 44 50 53 91 99 110 120 128 133 141 150	11.11
(14, 6)	307	16 17 19 32 52	3.27
(11, 0)	307	9 16 17 24 29 61 73 78 103 111 112 114 117 126 132 136 142	11.11
(1, 9)	307	16 17	1.31

Table 5.9: The Average Percentage of Zero Δ

p	Average percentage of zero Δ
5	21.33
7	18.12
11	11.02
13	4.89
17	2.85
19	3.82
23	2.86
29	1.90
31	1.33
37	1.79
41	0.54
43	0.58
47	0.77
53	0.62
59	0.29
61	0.68
67	0.83
71	0.21
73	0.26
79	0.58
83	0.26
89	0.13
97	0.16
101	0.11
103	0.14
107	0.28
109	0.22
113	0.15
127	0.08

Chapter 6

Software Implementation

The software design, testing process and problems encountered are explained in this chapter. Implementation is done in C++, Java and RIM Blackberry Java Development Environment (JDE). Complete code listings are provided in Appendices A.1 to A.3.

6.1 Software Design

The first step in the program is to set up the system parameters. This involves setting the values of a, b and p, with a and b being the coefficient of the irreducible polynomial $f(x)$ over GF(p) with period Q, where Q is a factor of $p^2 + p + 1$. Since the system parameters should remain constant, they can be hard-coded in the program.

The program will then display a menu for the user to choose which operation to perform: to compute a shared key pair, to sign a message, to verify a signature or to terminate the program. Flowcharts of these operations except program termination are illustrated in Figure 6.1, Figure 6.2 and Figure 6.3.

6.1.1 GH-DH Shared Key Computation

The program will prompt the initiator, Alice, to choose her private key from 0 to $Q = p^2 + p + 1$. After Alice has entered the secret key, x, the program will check to see if x is valid by testing if it is co-prime with $p^2 + p + 1$ and if x is less than $p^2 + p + 1$. If any of these cases fails, the program will prompt Alice to choose another value for the private key. After validating the choice of private key, the program will express x in its binary form and compute the $\pm x^{th}$ terms of the sequence generated by $f(x)$ according to the DSEA algorithm. The pair (s_x, s_{-x}) is Alice's public key pair.

In setting up a session with Bob, Alice will send her public key pair to Bob and Bob will response by sending his public key pair to Alice. The program will prompt Alice to enter the received public key pair. These received public keys will be used to set up another irreducible polynomial $f_A(x)$ over GF(p) and the $\pm x^{th}$ terms of the sequence generated by $f_A(x)$ will be the shared key pair between Alice and Bob. This can be done by setting a and b equal to the keys in the received key pair and by using the DSEA algorithm for computing the $\pm x^{th}$ terms of the sequence generated by $f(x)$. This is the key pair shared between Alice and Bob.

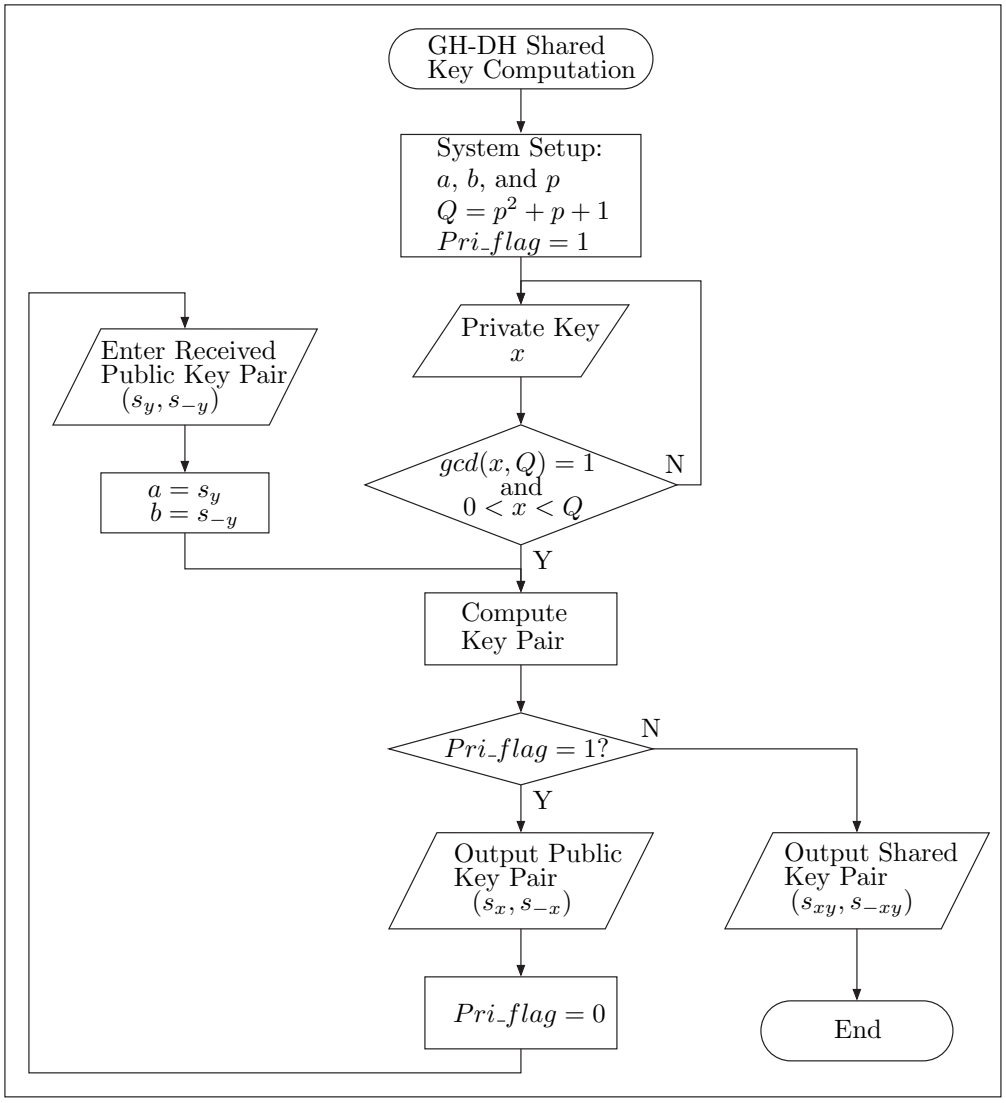

Figure 6.1: Flowchart Diagram of GH-DH Shared Key Computation

6.1.2 GH-DSA Signature Generation

Since signature generation requires the public key pair of the user, the program will prompt the signer, Alice, to choose her private key as in the first part of Section 6.1.1. However, in Case 2 of signature verification process as explained in Section 4.2.2, the verifier, Bob, needs to compute $s_{\pm(x-1)}$ using (s_x, s_{x+1}) and its duals. Therefore, Alice should select her private key x such that Δ does not equal to zero. The program will compute Δ for the x and will ask Alice to choose another x if the condition is not satisfied.

After computing the public key pair, the program will prompt Alice to enter a value for the signing key, k, with the restrictions described in Section 4.2.1. The program computes (s_{k-1}, s_k, s_{k+1}) and its duals after a valid k value is entered. However, this is not the final validation of the signing key choice yet.

As describe in the signing equation in Section 4.2.1, the parameter r equals to the sequence term s_k and it should be co-prime with Q. Therefore, the program will perform the co-prime test on r and Q after s_k is computed. To compute a previous sequence term as describe in Section 3.2, Δ must be a non-zero number. This algorithm would be used in signature verification as Bob has to compute $s_{\pm(k-1)}$ using (s_k, s_{k+1}) and its duals. Therefore, in the signing process, the program should also compute Δ and check to see if it equals zero. If any of these tests fails, the program will prompt Alice to select another value for k.

The program will then ask Alice to enter the message that she needs to sign. The hash function is only implemented in Java since two commonly used hash functions MD5 [28] and SHA-1 [19] are available in java.security package. In other programming

environments, the hashed value, h, is set to be the same as the message, m. The program will compute the parameter t according to the signing equation. If t equals zero, x will be compromised as the term with t in the signing equation disappears. Therefore, t must be a non-zero number. If t is zero, the program will prompt Alice to select another value for k.

After the tests on r, t and Δ, the choice of k is now validated, and the program will output (m, r, t) as the signed message. It will also output (s_x, s_{x+1}), (s_k, s_{k+1}) and their duals.

6.1.3 GH-DSA Signature Verification

The program will prompt the verifier, Bob, to entered all the received data including (m, r, t), (s_x, s_{x+1}), (s_k, s_{k+1}) and theirs dual. Depending whether t is co-prime with Q, the program first computes either $s_{\pm(k-1)}$ using (s_k, s_{k+1}) and its dual or $s_{\pm(x-1)}$ using (s_x, s_{x+1}) and its dual. It will then determine the corresponding u and v as described in Section 4.2.2 and compute either $s_{\pm u(k+v)}$ or $s_{\pm u(x+v)}$ terms using the mixed term algorithm given in Section 3.3.

If t is co-prime with Q, then the program will compare to see if $s_{\pm u(k+v)}$ equals $s_{\pm x}$. If t is not co-prime with Q, then the program will compute $s_{\pm kt}$ terms using the mixed term algorithm. Then, it will compare to see if $s_{\pm u(x+v)}$ equals $s_{\pm kt}$. Again, the signature is valid if and only if these sequence terms match.

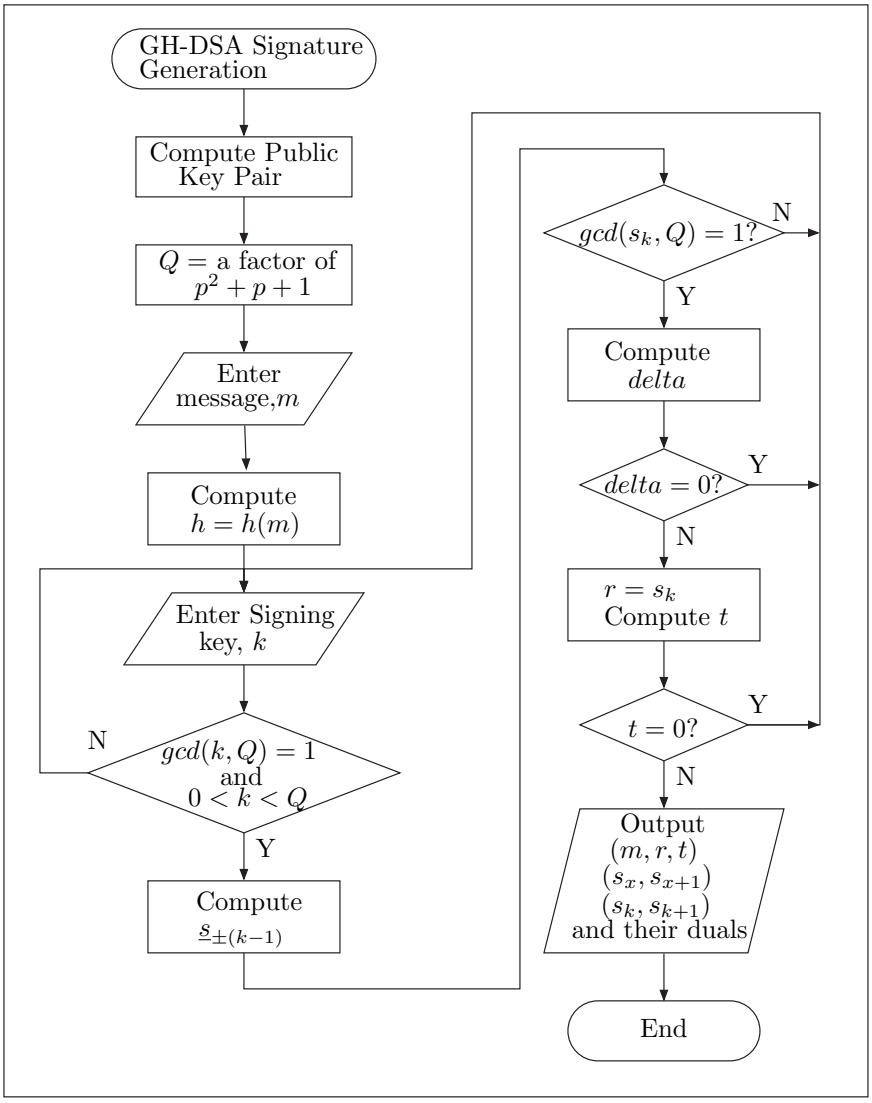

Figure 6.2: Flowchart Diagram of GH-DSA Signature Generation

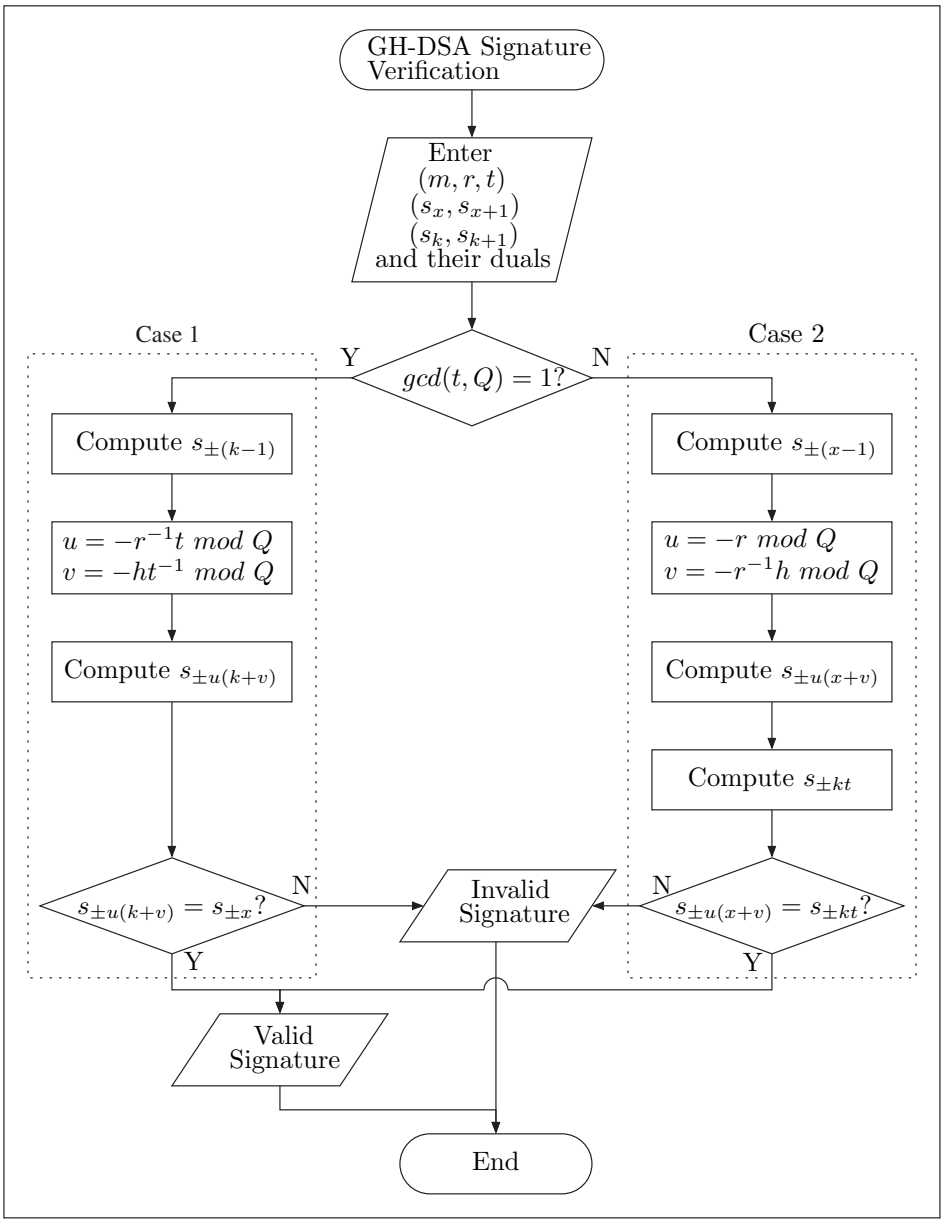

Figure 6.3: Flowchart Diagram of GH-DSA Signature Verification

6.2 Testing and Parameters

Two cases, toy case and real system case, are used to verify the coding. The testing and parameter selection of these two cases are described below.

6.2.1 Toy Case

In the initial phase of testing, an irreducible polynomial $f(x)$ over a small field is used. Working in small field makes it possible to verify all the sequence term computations including the intermediate ones.

In order to verify that all computed sequence terms are correct, a program is written in C++ to generate all the terms in one period of the third-order characteristic sequence as shown at the end of this section. Sequence terms are computed using a recursive formula:

$$s_{j+3} = as_{j+2} - bs_{j+1} + s_j, \ for\ i = 0, 1, 2, \ldots$$

which is derived from $f(x)$. This program can display any $\pm k^{th}$ terms in the sequence.

An irreducible polynomial in GF(5) is chosen for the initial phase of testing:

$$f(x) = x^3 + x - 1.$$

The maximum period of the third-order characteristic sequence generated by any

irreducible polynomial $f(x)$ over GF(5) is:

$$Q = 5^2 + 5 + 1 = 31.$$

All computed sequence terms, such as public key pairs, shared key pairs and r in the signing equation are compared to the sequence terms generated by the recursive formula to ensure there is no mismatch.

Sequence term Generation

```
/*************************************************************************
 * This program calculates the period of a third-order characteristic
 * sequence generated by a given irreducible polynomial.
 *
 * Susana Sin
 ************************************************************************/

#include<iostream.h>

int LFSR(int s[],int n,int p, int g[]) {
  bool flag = 0;
  int i = 0;

  while (flag == 0) {
    s[i + n] = (s[i] * g[0] + s[i + 1] * g[1] + s[i + 2] * g[2]) % p;
    if (s[i + n] < 0) s[i + n] = s[i + n] + p;
    if ((i >= 3) && (s[i +n- 2] == s[0]) && (s[i +n- 1] == s[1]) && (s[i+n] == s[2])) flag = 1;
    i++;
  }
  return i; // period of sequence
}

int main(){
  const int n = 3;
  const int p = 5;
  int a = 0, b = 1;
  int q = p * p + p + 1, per = 0, k = 1;
  int g[n] = {1, -b, a};
  int s[q + n];

  /*Initial State*/
  s[0] = 3;
  s[1] = a;
  s[2] = (a * a - 2 * b) % p;
  if (s[2] < 0) s[2] = s[2] + p;
```

```
  per = LFSR(s, n, p, g);

  cout << "Period of sequence generated by f(x) = x^3 - " << a << "x^2 + ";
  cout << b << "x - 1 over GH(" << p << ") is " << per << endl << endl;

  cout << "To display the +/-(k-1)th state of the sequence: (Enter 0 to stop)" << endl;
  while (k > 0) {
    cout << "Enter value of k" << endl << "k = ";
    cin >> k;
    if (k <= 0) break;
    if (k > per) k = k % per;
    cout << "(s" << (k - 1) << ", s" << k << ", s" << (k + 1) << ") = (";
    cout << s[k - 1] << ", " << s[k] << ", " << s[k + 1]<< ")" << endl;
    cout << "(s" << -(k - 1) << ", s" << -k << ", s" << -(k + 1) << ") = (";
    cout << s[per-(k-1)] << ", " << s[per-k] << ", " << s[per-(k+1)]<< ")" << endl;
  }
  return 0;
}
```

6.2.2 Real System Case

Since the security of the GH-PKC is based on the intractability of DL problem in GF(p^3), to implement a GH-PKC with 1024-bit security, a 340-bit prime p is required. The parameter Q in GH-DSA is equivalent to the parameter q in the DSS standardized by NIST [20], which is standardized to be 160-bit prime factor of $p-1$. The parameter Q in GH-DSA should also be chosen as a 160-bit prime factor of p^2+p+1. In that case, the verification process can be simplified to only Case 1 as t is always co-prime with Q. The condition on x such that Δ does not equal to zero is not required in the signing process. The 340-bit prime p should be chosen such that p^2+p+1 has a 160-bit prime factor which is Q.

System parameters are chosen as:

$$
\begin{aligned}
p &= 25241001428020650913199864753466204394427825281223816408128163843843641958926288184400247294075952092 91\\
a &= 10096784624666345343732361659954789777913228641532071493304907762091482797330771799383971091151487089 51\\
b &= 20621602264418475981502454995422784810870872365985454817408829350029390623706895406373921929388361626 83\\
Q &= 164705219395020291376758884936962412458513495611 1.
\end{aligned}
$$

To test this "Real System" it is not practical to verify all the generated sequence terms as there are too many terms in one period. As all the sequence terms were verified in the "Toy Case", there is no need to verify these terms again here. Instead, to see if the program is correctly implemented, two copies of the program are executed at the same time to simulate two user Alice and Bob, namely Program A and Program B respectively. By feeding random private keys x_A and x_B to Program A and Program B respectively, the programs should return their public key pairs. After the public key pairs have been generated, the public key pair from Program B is entered into Program A to simulate Alice receiving the public key from Bob. Similarly, copying the public key pair from Program A to Program B simulates Bob receiving the public key from Alice. Then each program outputs a common key pair that is the common key pair shared between Alice and Bob, and the output key pairs in the two programs are verified to be the same. Similar testing would be done to the signature scheme.

6.3 Platforms

The GH-PKC is implemented on a personal computer (PC) and on a RIM Blackberry device. Specifications of these platforms are given in this section.

6.3.1 PC

A Pentium 4 PC with 2.4 GHz processor and 512 MB of RAM is used for the implementation. C++ and Java versions are implemented and are ran under Windows XP.

6.3.2 Blackberry

A RIM Blackberry 6210 wireless handheld is chosen as a constrained device for the implementation. This handheld is a data and voice-enabled wireless handheld which operates on 900/1900 MHz GSM/GPRS wireless network. The Blackberry consists of a 32-bit microprocessor with 16MB flash memory plus 2MB SRAM.

A docking cradle is used to connect the device to a Universal Serial Bus (USB) port of a PC. Software for Microsoft Windows is provided by RIM to transfer information between the device and the PC.

New application can be developed using RIM's Blackberry Java Development Environment (JDE) which is freely available [22]. It provides a integrated development

environment and simulation tools for building Java 2 Micro Edition (J2ME) applications. After the application is successfully compiled in Blackberry JDE, a .cod file is generated. The .cod can be loaded to the handheld through the docking cradle by using the following command in the command prompt:

JavaLoader -usb load *filename*.cod

where *filename* is the name of the application. To delete the application from the handheld, use the following command:

JavaLoader -usb erase [-f] *filename*.cod

where the -f option forces removal of the application even if it is in use.

6.4 Problems Encountered

To facilitate the handling and calculation of these large integers, ZZ, BigInteger and CryptoInteger data types are used in C++, Java and RIM Blackberry Java Development Environment (JDE) respectively. These data type provides basic mathematical operators as well as other mathematical functions such as GCD($\cdot$) for use with large integers. Different problems are encountered in different programming environments.

6.4.1 Implementation in C++

The NTL library [31], written by Victor Shoup, is used in this implementation. This C++ library provides data structures and algorithms for arbitrary length integers, vectors, matrices and polynomials over finite fields. ZZ class represents large signed integer, ZZ_p class represents large integer mod p and vec_ZZ_p class represents an array of ZZ_p. Before ZZ_p and vec_ZZ_p classes can be used, the modulus, p, should be first initialized:

```
ZZ p = to_ ZZ("5");        // 5 is in decimal
ZZ_p::init(p);             // p is used to initialize the modulus for ZZ_ p class allows
                           // arithmetics done in mod p
ZZ_p x = to_ZZ_p("3");
vec_ ZZ p s;
s.SetLength(6);
```

6.4.2 Implementation in Java

The BigInteger class is used in standard Java and can be found in java.math application program interface (API) in Java^{TM} Platform. It provides data structures and algorithms for arbitrary length integers. There are several constructors for BigInteger:

```
BigInteger x = new BigInteger("3");   // 3 is in decimal
BigInteger [] s = new BigInteger[6];
```

Binary arithmetic methods in BigInteger class, however, do not include modulo operations. In order to perform a modulo operation, the mod(·) method should be used.

6.4.3 Implementation in RIM Blackberry JDE

RIM Blackberry JDE does not support the BigInteger class. Instead of using BigInteger class, CryptoInteger class is used and can be found in the net.rim.device.api.crypto API. A modulo operation is available for any binary operation. CryptoIntegers are constructed as follows:

```
CryptoInteger x = new CryptoInteger("03");   // 03 is in hexadecimal
CryptoInteger [] s = new CryptoInteger[6];
```

6.4.4 Licensing Problem on Blackberry Handheld

Unfortunately, the net.rim.device.api.crypto API is a controlled API, and in order to use controlled APIs in an application on the handheld, the application must be signed using keys provided by RIM before the application can be loaded on to the handheld. Moreover, the CryptoInteger class is one of the classes that make use of Certicom technology. A Certicom license is required to use these classes, and registration with RIM alone does not allow access to these classes. Fortunately, signing application is not required to run application using the JDE simulator. Analysis is performed in the simulator instead of the handheld to avoid licensing issues.

Without running the application on the handheld, it is not possible to analyze the amount of power consumed by the application. Because the process time varies directly with the power consumption, analysis is done based on the processing time.

Chapter 7

Analysis

Time analysis is performed on PC (Java only) and Blackberry simulator. The results are provided in Table 7.1-6. Key length affects the process time since both the DSEA algorithm and the square-and-multiply algorithm look at the binary representation of a key bit-by-bit. Therefore, choosing a reasonable length of keys is essential for time analysis.

7.1 Key Agreement Protocol

Private keys with average and maximum lengths are used in the analysis as the length of the private keys affect the processing time to a great extent. The GH-DH private key x are chosen between 0 and $p^2 + p + 1$, resulting in a maximum and average length

of 680-bit and 340-bit x respectively for a 340-bit p. The DH private key x are chosen between within 0 and $p-1$, resulting in a maximum and average 1024-bit and 512-bit x respectively for a 1024-bit p.

From the process time results, it can be seen that GH-DH key agreement protocol with average and maximum private key lengths on both PC and Blackberry simulator are about 60% and 40% faster than DH key agreement protocol. This is expected as the lengths of the private key in the GH-DH key agreement protocol is only two-third of that in the DH key agreement protocol.

Table 7.1: Key Agreement Process Time on PC in Java using BigInteger

	GH-DH Key		**DH Key**	
	340-bit x	680-bit x	512-bit x	1024-bit x
	70ms	150ms	110ms	261ms
	70ms	151ms	120ms	250ms
	90ms	150ms	110ms	250ms
	70ms	150ms	121ms	250ms
	70ms	150ms	111ms	260ms
	80ms	150ms	121ms	250ms
	80ms	150ms	120ms	251ms
	70ms	150ms	110ms	251ms
	70ms	150ms	120ms	251ms
	80ms	160ms	111ms	250ms
	80ms	151ms	130ms	250ms
	70ms	150ms	110ms	250ms
	70ms	150ms	110ms	250ms
	70ms	150ms	120ms	250ms
	80ms	151ms	110ms	250ms
	70ms	160ms	120ms	251ms
	80ms	160ms	120ms	250ms
	70ms	151ms	120ms	261ms
	70ms	150ms	110ms	250ms
	70ms	160ms	110ms	251ms
Maximum	90ms	160ms	130ms	261ms
Minimum	70ms	150ms	110ms	250ms
Average	74ms	152ms	116ms	252ms

Table 7.2: Key Agreement Process Time on Blackberry in JDE using CryptoInteger

	GH-DH Key		**DH Key**	
	340-bit x	680-bit x	512-bit x	1024-bit x
	8480ms	17840ms	11800ms	23550ms
	8590ms	17260ms	11730ms	23870ms
	8470ms	17260ms	11730ms	23550ms
	8970ms	17300ms	11700ms	23550ms
	8470ms	17750ms	11670ms	24080ms
	8520ms	17560ms	12100ms	23510ms
	8490ms	17320ms	11700ms	23880ms
	8530ms	17320ms	11730ms	23560ms
	9080ms	17300ms	11730ms	23570ms
	8500ms	17700ms	11710ms	23920ms
	8520ms	17330ms	11700ms	23580ms
	8490ms	17310ms	11720ms	23940ms
	8540ms	17330ms	11720ms	23540ms
	8550ms	17340ms	11690ms	23820ms
	8510ms	17810ms	11720ms	23550ms
	8900ms	17380ms	11700ms	23570ms
	8460ms	17400ms	11920ms	23850ms
	8540ms	17390ms	11720ms	23560ms
	8530ms	17720ms	11690ms	23840ms
	8650ms	17350ms	11700ms	23620ms
Maximum	9080ms	17840ms	12100ms	24080ms
Minimum	8460ms	17620ms	11670ms	23410ms
Average	8590ms	17449ms	11745ms	23691ms

7.2 Signing Procedure

The key length of the signing key k affects the digital signature generation time. Signing key with average and maximum lengths are considered in the analysis. The GH-DSA signing key k are chosen between 0 and Q, where Q is a 160-bit prime factor of p^2+p+1 with a 340-bit p. The DSS signing key k are chosen between 0 and q, where q is a 160-bit prime factor of $p-1$ with a 1024-bit p. Since both Q and q are 160-bit, same signing keys can be used for both cases with maximum length being 160-bit and average length being 80-bit.

From the results, the GH-DSA signature generation is only slightly faster than the DSS signature generation on both PC implementation and Blackberry implementation. Although improvement cannot be seen clearly, GH-DSA signature generation time is still comparable with the DSS signature generation time.

The difference in the two signature schemes is the way r in which is computed. The GH-DSA uses the DSEA algorithm to compute r whereas the DSS uses the square-and-multiply algorithm to compute r. In the DSEA algorithm, for each bit of binary representation of k there are on average 9 multiplications in GF(p) where p is a 340-bit prime. In the square-and-multiply algorithm, however, there is only 1 multiplication in GF(p), where p is a 1024-bit prime. To compare the complexity between the two algorithms, the number of multiplications with the same length of operands is examined instead. Each 1024-bit operand requires three 340-bit variables to store the number. In order to perform a multiplication on two 1024-bit numbers based on 340-bit long operands, a total of 9 multiplications in GF(p) with 1024-bit p is required. Both algorithms require the

same number of multiplications with 340-bit operands. However, the modulus length in the DSEA algorithm is only one-third that of the square-and-multiply algorithm. This improvement can be seen more easily in a hardware implementation.

In the software implementations, operations are done using facilities provided in BigInteger class in Java and CryptoInteger class in Blackberry JDE as explained in Section 6.4.2 and 6.4.3. The performance of these classes play an important role in the processing time.

Table 7.3: Signature Generation Time on PC in Java using BigInteger

	GH-DSA Sign		**DSS Sign**	
	80-bit k	160-bit k	80-bit k	160-bit k
	10ms	20ms	20ms	20ms
	20ms	20ms	20ms	20ms
	10ms	20ms	10ms	30ms
	10ms	20ms	10ms	20ms
	10ms	20ms	10ms	20ms
	10ms	20ms	10ms	20ms
	10ms	20ms	10ms	20ms
	10ms	30ms	10ms	20ms
	10ms	20ms	10ms	20ms
	10ms	20ms	10ms	20ms
	20ms	20ms	10ms	20ms
	10ms	20ms	10ms	20ms
	10ms	20ms	20ms	20ms
	10ms	20ms	10ms	20ms
	10ms	20ms	10ms	20ms
	10ms	20ms	10ms	20ms
	10ms	10ms	20ms	20ms
	10ms	10ms	10ms	20ms
	10ms	30ms	10ms	20ms
	10ms	20ms	10ms	20ms
Maximum	20ms	30ms	20ms	30ms
Minimum	10ms	10ms	10ms	20ms
Average	11ms	20ms	12ms	21ms

Table 7.4: Signature Generation Time on Blackberry in JDE using CryptoInteger

	GH-DSA Sign		**DSS Sign**	
	80-bit k	160-bit k	80-bit k	160-bit k
	1050ms	2030ms	1170ms	2280ms
	1080ms	2110ms	1170ms	2240ms
	1050ms	2080ms	1170ms	2230ms
	1100ms	2070ms	1170ms	2220ms
	1110ms	2070ms	1160ms	2240ms
	1100ms	2080ms	1160ms	2220ms
	1030ms	2100ms	1150ms	2260ms
	1080ms	2080ms	1140ms	2230ms
	1100ms	2070ms	1130ms	2230ms
	1050ms	2090ms	1120ms	2230ms
	1100ms	2070ms	1150ms	2270ms
	1050ms	2080ms	1120ms	2230ms
	1100ms	2070ms	1130ms	2230ms
	1110ms	2100ms	1130ms	2230ms
	1080ms	2080ms	1130ms	2230ms
	1100ms	2030ms	1130ms	2220ms
	1100ms	2080ms	1120ms	2270ms
	1110ms	2070ms	1140ms	2230ms
	1100ms	2090ms	1130ms	2230ms
	1080ms	2040ms	1130ms	2220ms
Maximum	1110ms	2110ms	1170ms	2280ms
Minimum	1030ms	2030ms	1120ms	2220ms
Average	1084ms	2075ms	1143ms	2237ms

7.3 Signature Verification

The signatures generated in the previous section are used as test data for signature verification. From the results, the GH-DSA signature verification is slightly faster than the DSS signature verification on the Blackberry implementation and is about 20% faster on the PC implementation.

The verifier needs to compute $s_{\pm u(k+v)}$ as described in Section 4.2.2 for the GH-DSA and compute $g^{ht^{-1}}$ and $y^{-rt^{-1}}$ as described in Section 2.2 for the DSS.

The computation of $s_{\pm u(k+v)}$ terms in GH-DSA is done using the mixed term algorithm described in Section 3.3. The mixed term algorithm makes use of the DSEA algorithm twice: one to compute the $\underline{s}_{\pm(v-1)}$ states generated by $f(x)$ and the other one to compute the $\pm u^{th}$ terms generated $g(x)$. Parameters u and v are computed as described in Section 4.2.2 and they are both 160-bit long.

The exponentiations $g^{ht^{-1}}$ and $y^{-rt^{-1}}$ in DSS are done using the square-and-multiply algorithm. Since g is an element in GF(p) with order q, the exponents should be modulo q. As a result, the exponents are 160-bit numbers.

Both the GH-DSA and the DSS signature verification schemes take twice as long as their corresponding signing procedure with a 160-bit signing key as the signing procedures utilize the DSEA algorithm and the square-and-multiply algorithm only once.

As described in the previous section, the DSEA algorithm has a lower complexity compared to the square-and-multiply algorithm. Improvement can be seen more easily

in a hardware implementation.

Table 7.5: Signature Verification Time on PC in Java using BigInteger

	GH-DSA Verify		**DSS Verify**	
	80-bit k	160-bit k	80-bit k	160-bit k
	40ms	40ms	50ms	50ms
	30ms	30ms	50ms	50ms
	40ms	40ms	41ms	40ms
	40ms	30ms	40ms	50ms
	40ms	40ms	51ms	40ms
	30ms	41ms	40ms	40ms
	40ms	30ms	50ms	50ms
	30ms	40ms	40ms	50ms
	30ms	30ms	40ms	40ms
	40ms	40ms	50ms	40ms
	30ms	31ms	40ms	40ms
	40ms	30ms	40ms	41ms
	40ms	40ms	50ms	40ms
	40ms	30ms	40ms	40ms
	40ms	40ms	40ms	40ms
	40ms	40ms	40ms	40ms
	30ms	40ms	51ms	40ms
	40ms	40ms	50ms	40ms
	30ms	30ms	50ms	40ms
	40ms	40ms	50ms	50ms
Maximum	40ms	41ms	51ms	50ms
Minimum	30ms	30ms	40ms	40ms
Average	37ms	36ms	45ms	43ms

Table 7.6: Signature Verification Time on Blackberry in JDE using CryptoInteger

	GH-DSA Verify		**DSS Verify**	
	80-bit k	160-bit k	80-bit k	160-bit k
	4020ms	4220ms	4540ms	4540ms
	4060ms	4230ms	4520ms	4520ms
	4290ms	4250ms	4530ms	4530ms
	4310ms	4220ms	4530ms	4560ms
	4290ms	4240ms	4520ms	4600ms
	4320ms	4230ms	4570ms	4750ms
	4230ms	4250ms	4520ms	4560ms
	4300ms	4240ms	4570ms	4510ms
	4280ms	4250ms	4540ms	4540ms
	4260ms	4250ms	4510ms	4570ms
	4240ms	4250ms	4530ms	4520ms
	4230ms	4260ms	4600ms	4510ms
	4240ms	4280ms	4520ms	4560ms
	4230ms	4240ms	4530ms	4530ms
	4270ms	4240ms	4570ms	4510ms
	4240ms	4240ms	4530ms	4540ms
	4260ms	4260ms	4530ms	4540ms
	4240ms	4240ms	4630ms	4550ms
	4250ms	4240ms	4720ms	4520ms
	4270ms	4280ms	4550ms	4520ms
Maximum	4320ms	4280ms	4720ms	4750ms
Minimum	4230ms	4220ms	4510ms	4510ms
Average	4263ms	4246ms	4553ms	4549ms

Chapter 8

Conclusions and Future Work

8.1 Summary and Conclusions

The GH-DH key agreement protocol and the GH-DSA are implemented in software in C++ and Java over GF(p) with 1024-bit security. Timing analysis is done with Java implementation on PC and on the Blackberry simulator. The main contributions of this thesis are summarized as follows.

1. This is the first software implementation of the GH-DH key agreement protocol and GH-DSA to see the feasibility of implementing the system in software. The GH-DH and the GH-DSA aimed to improve the performance over existing DH and DSS systems by decreasing the size of modulus to speed up the computations while maintaining the same level of security. This is desirable especially on constrained

device as there is limited power and time for computations. Blackberry is chosen as a constrained platform for comparison in this thesis.

2. The implementation of GH-DH and GH-DSA are done faithfully according to the algorithms provided in the original papers on GH-PKC [10, 11] except for the algorithm used in computing the $s_{\pm(k+v)}$ terms. To compute the $s_{\pm(k+v)}$ terms, the general result of LFSR sequences is used as discussed in Section 3.3 instead of using the DSEA algorithm as stated in the paper [11].

3. Design issues and the choice of system parameters are summarized below:

   (a) In each step of the iteration in the DSEA algorithm provided in Section 3.1, sequence terms $(s_{\pm(t'+1)}, s_{\pm t'}, s_{\pm(t'-1)})$ are computed using the six sequence terms generated in the previous iteration, $(s_{\pm(t+1)}, s_{\pm t}, s_{\pm(t-1)})$. Therefore, instead of storing all the sequence terms computed in all iterations, only the six newly generated ones $(s_{\pm(t'+1)}, s_{\pm t'}, s_{\pm(t'-1)})$ are kept as they are the only ones used in the next iteration.

   (b) In order to reduce the bandwidth usage, a signer has to choose a value for the signing key k such that Δ does not equal to zero as discussed in Section 5.2. Since there is no theoretical result available about the distribution of zero Δ, experiments are performed to determine the tradeoff between bandwidth usage and the time required to choose a valid k. Based on the experimental data, k cannot be taken as either $p-1$ or p as these values always cause Δ to be zero. Besides, the probability of Δ being zero is minimized by choosiong

both a and b to be non-zero integers. However, by choosing either a or b to be zero, the number of multiplications in the DSEA algorithm explained in Section 3.1 can be reduced to $7\lfloor log_2 k \rfloor$ and $8\lfloor log_2 k \rfloor$ respectively. This is a tradeoff between computational cost and bandwidth usage. Moreover, as p increases, the percentage of Δ being zero decreases. The percentage is low enough to conclude that the cost of re-selecting values for k is lower than the cost of increasing the bandwidth usage.

(c) The algorithm for computing mixed term given in Section 3.3 requires the inverse of Matrix M_0. A matrix is invertible if and only if its determinant does not equal zero. Since the determinant of matrix M_0 depends on only a and b, matrix M_0 is guaranteed to be invertible by choosing a and b correspondingly.

(d) The parameter Q in GH-DSA is a factor of $p^2 + p + 1$. The paper on GH-DSA [11] does not specify the size of Q. It includes two cases for signature verification depending on whether t is co-prime with Q. The parameter Q is equivalent to the parameter q in DSS, which is a standard 160-bit prime factor of $p - 1$ as set by NIST. Therefore Q in GH-DSA is chosen to be a 160-bit prime factor of $p^2 + p + 1$. Since Q is a prime, signature verification is simplified to Case 1 as t is always co-prime with Q. The condition on x such that Δ does not equal to zero as discussed in Section 6.1.2 is removed as only Case 2 of the signature verification requires the computation of $s_{\pm(x-1)}$ terms. The system parameter p is chosen to be a 340-bit prime such that $p^2 + p + 1$ has a 160-bit prime factor.

4. Problems are identified in the thesis. The first problem is in facilitating the handling and calculations of large integers, and different data classes are used in different coding environments. Another problem is the licensing issues. In order to use CryptoInteger class on the Blackberry handheld, a Certicom license needs to be obtained and registration needs to be completed with RIM. To avoid licensing issues, analysis is done on the Blackberry simulator instead.

5. Comparisons are done based on the author's own implementations of DH and DSS utilizing square-and-multiply method to perform exponentiation without any optimization. A performance improvement of the GH-DH key agreement protocol over the DH key agreement protocol can be seen clearly in the software implementation as the length of the private key in the GH-DH key agreement protocol is reduced to only two-third of that of the DH key agreement protocol. Processing time of GH-DSA is comparable to that of DSS although improvement cannot be seen easily in the software implementation. The performance of the BigInteger and the CryptoInteger classes play an important role in the processing time as well.

8.2 Recommendations for Future Work

Recommendations for future research are listed below.

- **Software optimization:** The software implementation of GH-DH key agreement protocol and GH-DSA can be further optimized for better performance. One can also implement a data class to facilitate the handling and calculations of large

numbers so that the GH-PKC can be loaded on to any platforms such as mobile headsets and other constrained devices.

- **Exponentiation algorithm for DH and DSS:** Square-and-multiply method based on binary representation of k is used to perform exponentiation in the author's own implementation of DH and DSS without any optimization. Tricks can be utilized to alternate the binary representation of k in order to speed up the exponentiation algorithm by decreasing the number of multiplications, such as using non-adjacent form (NAF) and minimal hamming weight representation [13]. Moreover, since the exponentiations in DH and DSS are done using fixed bases α and g respectively, pre-computations can be done once and re-used for many exponentiations. This idea was first introduced by Brickell, Gordon, McCurley and Wilson and it is known as the BGMW method [3]. Future research can focus on comparing GH-DH and GH-DSA to DH and DSS using improved exponentiation algorithms.

- **Bandwidth saving:** In order to minimize the bandwidth usage, the signing key k should be chosen such that Δ does not equal zero. There is no theoretical result available about the distribution of zero Δ. One can theoretically prove the probability of Δ being zero.

- **Comparisons with other cryptosystems:** Another extension to this research is to compare GH-PKC to RSA and EC cryptosystems. RSA is already available commercially on mobile headsets including Ericsson [7] and Motorola [29] and EC-DSA is already available commercially on the Blackberry handheld [23]. Therefore,

future research would require software optimization for the mobile headsets and the Blackberry platform to allow for better comparisons.

- **Hardware implementation:** The modulus length in the GH-PKC is only one-third of that of existing DH and DSS with the same level of security. This is beneficial in hardware implementation as the hardware complexity can be reduced significantly. Future research can focus on the improvement of GH-PKC over DH and DSS in hardware implementations.

Bibliography

[1] G. Agnew, R. Mullin and S. Vanstone, An implementation of elliptic curve cryptosystems over $\mathbb{F}_{2^{155}}$, IEEE Journal on Selected Areas in Communications, vol. 11, pp. 804-813, 1993.

[2] M. Briceno, I. Goldberg and D. Wagner, An implementation of COMP128, 1998. This material is available at URL: http://www.iol.ie/~kooltek/a3a8.txt.

[3] E. F. Brickell, D. M. Gordon, K. S. McCurley and D.B. Wilson, Fast exponentiation with precomputations, *The Proceedings of Eurocrypto '1992*, Lecture Notes in COmputer Science, vol. 658, Springer-Verlag, 1992.

[4] R. Canetti and H. Krawczyk, Security analysis of IKE's signature-based key-exchange protocol, *Advances in Cryptology, The Proceedings of Crypto '2002*, Lecture Notes in COmputer Science, vol. 2442, Springer-Verlag, pp. 143-161, 2002.

[5] W. Diffie and M. E. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644-654, Nov. 1976.

[6] T. ElGamal, A public-key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory*, vol. 31, pp. 469-472, 1985.

[7] Ericsson Press Release, RSA Security and Ericsson work together to secure wireless communications markets leaders to address security requirements for smartphones, January 13, 2000. This material is also available at URL:http://www.ericsson.com/press/archive/2000Q1/20000113-0047.html.

[8] K. J. Giuliani and Guang Gong, Efficient key agreement and signature schemes using compact represenations in GF(p^{10}), to apprear in *the Proceedings of the IEEE Symposium on Information Theory 2004*, June 28-July 2, 2004, Chicago.

[9] K. J. Giuliani and G. Gong, Analogues to the Gong-Harn and XTR Cryptosystems, Department of C&O, University of Waterloo, Technical Report CORR 2003-34. This material is also available at URL: http://www.cacr.math.uwaterloo.ca/techreports/2003/corr2003-34.ps.

[10] G. Gong and L. Harn, Public-key cryptosystems based on cubic finite field extensions, *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2061-2065, Nov. 1999.

[11] G. Gong, L. Harn, and H. Wu, The GH public-key cryptosystem, *The Proceedings of the Eighth Annual Workshop on Selected Areas in Cryptography (SAC) 2001*, Toronto, pp. 301-316, Aug. 2001.

[12] S.W. Golomb, *Shift Register Sequences*, California: Aegean Park Press, pp 34-35,

1982.

[13] D. M. Gordon, A sufvey of fast exponentiation methods, *Journal of Algorithm 27*, pp. 129-146, 1998.

[14] IEEE, IEEE 1363/D13: Standard Specifications for Public Key Cryptography, Nov. 12, 1999.

[15] N. Koblitz, Elliptic curve cryposystems, *Mathematics of Computation*, 1987.

[16] A. K. Lenstra and E. R. Verheul, The XTR public key system, *Advances in Cryptology, The Proceedings of Crypto '2000*, Lecture Notes in Computer Science, Springer-Verlag, vol. 1880, pp. 1-19, 2000.

[17] A. K. Lenstra and E. R. Verheul, Key improvements to XTR, *The Proceedings of AsiaCrypto '2000*, Lecture Notes in Computer Science, vol. 1976, 2000.

[18] V. Miller, Use of elliptic curves in cryptography, *Advances in Cryptology, The Proceedings of Crypto '1985*, Lecture Notes in Computer Science, Springer-Verlag, vol. 218, pp. 417-426, 1986.

[19] NIST, FIPS PUB 180-2: Secure Hash Standard, Aug. 2002. This material is also available at URL: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf.

[20] NIST, FIPS PUB 186-2: Digital Signature Standard (DSS), Jan. 2000. This material is also available at URL: http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf.

[21] J. R. Rao, P. Rohatgi, H. Scherzer and S. Tinguely, IBM Watson Research Center Partitioning Attacks: Or how to rapidly clone some GSM cards, *The Proceedings of 2002 IEEE Symposium on Security and Privacy*, pp. 29-39, May 2002. This material is also available at URL: http://www.research.ibm.com/intec/gsm.ps.

[22] RIM, Blackberry JDE 3.7. This software can be downloaded at URL: http://blackberry.com/developers/na/java/start/download.shtml.

[23] RIM Press Release, Research In Motion and Certicom to Enable Trusted Mobile Commerce Over Wireless Networks, July 24, 2000. This material is also available at URL: http://www.rim.net/news/press/2000/pr-24_07_2000.shtml.

[24] K. Nyberg and R. Rueppel, Message recovery for signature schemes based on the discrete logarithm problem, *Designs, Codes and Cryptography, 7*, pp. 61-81, 1996.

[25] K. Nyberg and R. Rueppel, A new signature scheme based on the DSA giving message recovery, *The First ACM Conference on Computer and Communications Security*, pp. 58-61, 1993.

[26] M. Rabin, Digitalized signatures and public-key functions as intractable as factorization, Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.

[27] R. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key crypsostems", *The Proceeding of the Communications of the ACM*, vol. 21, Issue 2, pp. 120-126, 1978.

[28] R. Rivest, RFC 1321: The MD5 Message-Digest Algorithm, Massachusetts Institute of Technology (MIT) Laboratory for Computer Science and RSA Security, April 1992. This material is also available at URL: http://www.ietf.org/rfc/rfc1321.txt.

[29] RSA Security, RSA Security Teams Up with Global Communications Giant Motorola, March 26, 2001. This material is also available at URL: http://www.rsasecurity.com/company/news/releases/pr.asp?doc_id=890.

[30] C.P. Schnorr, Efficient identification and signatures for smart cards, *Advances in Cryptology, The Proceedings of Crypto 1998*, Springer-Verlag, pp. 239-251, 1990.

[31] V. Shoup, Number Theory Library (NTL) Version 5.3. This library can be downloaded at URL: http://www.shoup.net/ntl.

[32] P. Smith and C. Skinner, A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms, *The Proceeding of Asiacrypt 1994*, pp. 298-306, Nov. 1994.

[33] P. Smith and C. Skinner, LUC public key encryption: a secure alternative to RSA, *Dr. Dobb's Journal*, vol. 18, no. 1, pp. 44-49, Jan. 1993.