LEBANESE AMERICAN UNIVERSITY

HARDENING THE ELGAMAL CRYPTOSYSTEM IN THE SETTING OF THE SECOND GROUP OF UNITS

BY

SUZAN MOHAMAD FANOUS

A thesis

submitted in partial fulfillment of the requirements

for the Degree of Master of Science in Computer Science

School of Arts and Sciences

August 2011

LEBANESE AMERICAN UNIVERSITY

School of Arts and Sciences - Beirut Campus

Thesis approval Form (Annex III)

Student Name: Suga Fanous I.D. #: 200803843

Thesis Title : Hardening the El Gamal Cryptosystem in the setting of the second group of units

Program : Master in Computer Science

Division/Dept : Computer Science & Mathematics

School : **School of Arts and Sciences**

Approved by: Ramzi Haraty

Thesis Advisor:

Member : Abdul-Nasser Kassar [redacted]

Member : Azzam Mourad <

Date 10/8/2011

(This document will constitute the first page of the Thesis)

THESIS/PROJECT COPYRIGHT RELEASE FORM

LEBANESE AMERICAN UNIVERSITY

By signing and submitting this license, I Suzan Mohamad Fanous grant the Lebanese American University (LAU) the non-exclusive right to reproduce, translate (as defined below), and/or distribute my submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video. I agree that LAU may, without changing the content, translate the submission to any medium or format for the purpose of preservation. I also agree that LAU may keep more than one copy of this submission for purposes of security, backup and preservation. I represent that the submission is my original work, and that I have the right to grant the rights contained in this license. I also represent that my submission does not, to the best of my knowledge, infringe upon anyone's copyright. If the submission contains material for which I do not hold copyright, I represent that I have obtained the unrestricted permission of the copyright owner to grant LAU the rights required by this license, and that such third-party owned material is clearly identified and acknowledged within the text or content of the submission. IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN LAU, I REPRESENT THAT I HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT. LAU will clearly identify my name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to my submission.

Name: Suzan Fanous

Signature [redacted] Date: 10, Aug-2011

PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.
This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Suzan Fanous

Signature: [redacted]

Date: 10, Aug-2011

ACKNOWLEDGMENTS

This research would not have been possible without the help and assistance of many persons.

First I would like to express my gratitude to my supervisor Dr. Ramzi Haraty who was always supporting me.

I am also deeply grateful to Dr. Abdul Nasser Kassar for his assistance.

Thanks go also to Dr. Azzam Murad for being a member of my committee.

Finally, special thanks go also to Dr. Samer Habre who treated us like sons and daughters.

DEDICATION

I would like to thank my great God, who like him there is no one to thank.

I thank God for I was blessed to learn the meaning of sacrifice from one of the closest persons to my heart, someone who dedicated his entire life for my well being. Last what I can say: Thank you "Dad".

Special thanks, to the one who has embraced me with her caring love like no one else. Throughout the years, you have enlightened my life and my words may never express my feelings towards you, but hope a simple "I love you" will do.

Special thanks to my family especially to my sister who was always with me, moment by moment.

Special thanks to my best friend Sanaa who was always supporting and inspiring me. Thank you Sanaa!

Finally, I dedicate this book to my Home Land "My Pride".

Hardening the ElGamal Cryptosystem in the Setting of the Second Group of Units

Suzan Mohamad Fanous

Abstract

The purpose of this thesis is to give up to date discussion of the principles, techniques and algorithms of interest in cryptographic practice. We choose to emphasize on the most practical and applied cryptographic system, El-Gamal scheme.

This thesis discusses modifications we made on El-Gamal scheme, mainly extending the scheme to work on the second group of units of Zn and the second group of units of the quotient ring of polynomials over a field with cyclic second group of units.

The arithmetic needed in this new setting is described. Algorithms are given. Advantages and efficiency of the new method are pointed out. Finally the security of the new scheme is studied and suggestions for further work are introduced.

Keywords: Second group of units of Z_n, Second group of units of $Z_n[x]/< h(x) >$, generator, ElGamal Cryptosystem, Baby Step Giant Step attack algorithm, Private key.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF CHARTS

Chapter One

Introduction

1.1 Problem Statement:

ElGamal Public Key Cryptosystem is one of the most practical cryptosystems nowadays. This cryptosystem works in the setting of any cyclic group G. This group G should be carefully chosen. Hence, when choosing G two conditions should be satisfied:

1. The operations in G are easy to perform.
2. The Discrete Logarithm problem in G should be infeasible.

ElGamal cryptosystem was extended in the setting of the following cyclic groups:

1. The domain of Gaussian integers.
2. Over the group of units of $Z_p[x]/< x^2 >$
3. The group of units of $Z_p[x]/< h(x) >$ where h(x) is an irreducible polynomial.
4. The group of units of $Z_p[x]/< h_1(x)h_2(x) \dots h_r(x) >$ where $h_i(x)$ are irreducible polynomials of pair-wise relatively prime degrees.

As known, the more secure and efficient the cryptosystem is the more practical and useful it becomes. So, there is always a need to enhance ElGamal and end up with a more secure and efficient cryptosystem.

1.2 Scope of the thesis:

In this work, we extend the ElGamal cryptosystem to work in the setting of a new group G; the second group of units. We ended up with three modified cases:

1. The second group of units of Zn when the first group of units is also cyclic.
2. The second group of units of Zn when the first group of units is not cyclic.
3. The second group of units of Z2[x]/<h(x)> where h(x) is an irreducible polynomial.

A study of the values of n for which the second group of units is cyclic was performed. We ended up with six cases; in three of them both the second group and the first group of units were cyclic whereas in the other three cases only the second group of units was cyclic.

Then, the algorithms for constructing the second group of units in each case were provided. Next, the modified ElGamal cryptosystem in each case was introduced and a study of its efficiency and security was pointed out to.

1.3 The organization of the thesis:

The thesis is organized as follows: Chapter one is an introductory chapter. Chapter two contains some basic cryptographic concepts. In chapter three a useful math background was provided. Chapter 4, describes some previous work that was performed on the ElGamal cryptosystem. Chapters 5 and 6 provide a description of the proposed cryptosystem. Chapter 7, contains the experimental results of testing the efficiency and security of the extended cryptosystems. Finally, Chapter 8 is a concluding chapter that concludes the thesis and recommends some future work.

Chapter Two

Fundamentals of Cryptography

2.1 Introduction:

Cryptography is the way of transmitting sensitive data and information such that only those who it's intended for can read and process. So, the aim of using cryptography is to hide information. In fact, this goal can't be achieved when some algorithms are used to reveal this information.

It should be noted that, the most striking development in the history of cryptography came in 1976 when Diffie and Hellman published New Directions in Cryptography [1].

Cryptanalysis is the way of deciphering the decrypted sensitive data.

2.2 Cryptography Definitions [2]

Encryption is the method of transforming a ***plaintext*** into a ***ciphertext***.

When the plaintext is transformed into ciphertext both human and machine won't be able to process it until it is decrypted.

This allows the transmission of confidential information over insecure channels without unauthorized disclosure. As data is stored on a computer, it's protected by access controls. But when it's sent over a network, it would be in a much more vulnerable state.

A system that provides encryption and decryption is referred to as a ***cryptosystem.***

The cryptosystem uses an encryption algorithm. Encryption algorithms apply complex mathematical formulas on the plaintext. They use a secret value called a key to encrypt and decrypt a message. An illustration is shown in Figure 2.1.

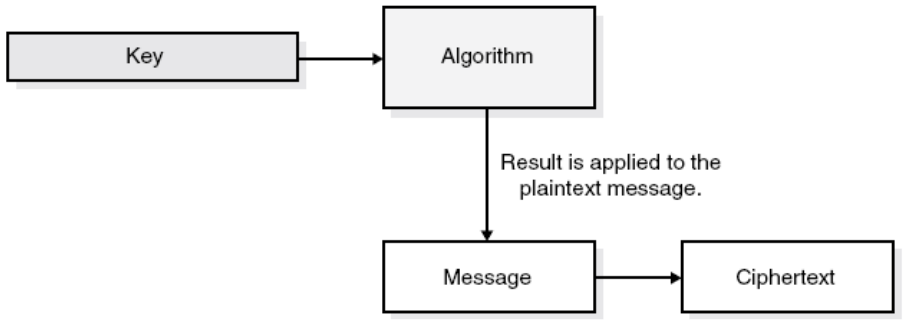

Figure 8-4 The key is inserted into the mathematical algorithm and the result is applied to the message, which ends up in ciphertext.

Figure 2.1 An Encryption Scheme

Cryptography Definitions

• **Cryptanalysis** is the method of getting plaintext from ciphertext without a key or without breaking the encryption.

• **Cryptology** The science that studies both cryptography and cryptanalysis.

• **Ciphertext** is the Data that is in encrypted or unreadable format.

• **Encipher** is to transform data into an unreadable format

• **Decipher** is to transform data into a readable format

• **Keyspace** is the set of all possible values from where a key is constructed.

• **Work factor** is the time and all the resources needed to decipher a message.

2.3 Types of Ciphers

Mainly, we have two types of ciphers: substitution and transposition.

1. The ***substitution cipher*** is based on replacing characters, or blocks of characters with different characters.
2. The ***transposition cipher*** moves the text in a certain way. It rearranges bits, characters, or blocks of characters to hide the original message. Hence it uses permutation instead of replacement.

2.3.1 Substitution Cipher

A substitution cipher is referred to as shifting alphabet. So, each letter is shifted a certain number of places beyond it in the original alphabet.

Example: in the Caesar Cipher, each letter is shifted to three places beyond it and is then replaced with the letter in that position.

There exist other types of substitutions where shifting takes place with more than one alphabet.

2.3.2 Transposition Cipher

In a ***transposition cipher***, letters are rearranged or scrambled they are not replaced with other letters from the alphabet. Also, a key is needed to determine to what positions are the letters moved.

Figure 2.2, shows a simple example on transposition cipher, but when introduced with complex mathematical functions transposition becomes difficult to break.

Nowadays, ciphers use both substitutions and permutations on the messages.

Such ciphers are vulnerable to attacks like the *frequency analysis.* It's clear that some words in each language are used more than other words.

Since such words are used frequently in the language, this would help attackers guess the transformation between plaintext and the ciphertext and hence to find the key that was used in the transformation.

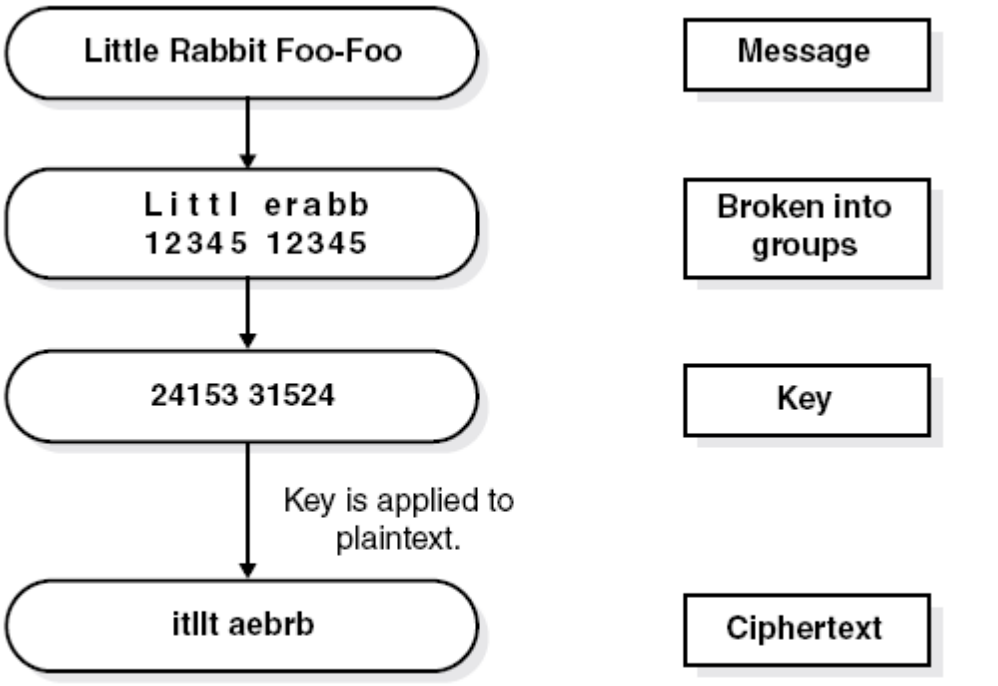

Figure 8-7 Transposition cipher

Figure 2.2 Transposition Cipher

2.4 Public Key Cryptography

Simply, this cryptosystem uses two types of keys, public and private, that are usually generated by an asymmetric algorithm that allows key distribution. Another key, the secrete key, is generated by a symmetric algorithm and is then used for the bulk encryption.

It should be noted that, in symmetric cryptography two users use the same key. On the other hand, asymmetric cryptography enables the two users to use different keys. But it should be noted that it is too slow compared to symmetric cryptography.

So to have the best of both types, a hybrid approach is used in a complementary way. This would allow a high level of security in an acceptable amount of time.

The hybrid approach works as follows: the symmetric algorithm is used at the first place to create the keys that would be used for encryption. Next, the asymmetric algorithm generates the keys used for key distribution.

It should be noted that the secret key is used for encrypting the messages you want to send .But the receiver needs the secrete key too in order to decipher the message. Hence, this key should not travel unprotected or else an attacker would be able decrypt the message and find your information.

So, this key should be encrypted too. To encrypt the secrete key an asymmetric algorithm is used. In fact, an asymmetric algorithm takes more time than the symmetric one due to math complexity. That's why we use it on the key and we usually use the fast symmetric algorithm on the message itself since its length is going to be longer than the length of the key.

Chapter Three

Mathematical Background

In this chapter, basic definitions from number theory, abstract algebra and finite fields are involved, to facilitate the understanding of the material.

Definition 1: [3]

Given n>0, the congruence relation (≡) is defined for two integers a, b in Z by:

$a \equiv b \ (mod\ n)$ if n| (a-b)

Definition 2: [4]

Two integers a, b are relatively prime if and only if GCD (a, b) = 1.

Definition 3: Let d be the gcd (a, n) [5]

1. If d=1, then $ax \equiv b \ (mod\ n)$ has a solution $x = a^{\emptyset(n)-1}.b$
2. If d≠ 1 , then $ax \equiv b \ (mod\ n)$ has a solution x if and only if d|b.

Definition 4: Chinese Remainder Theorem [6]

The system of congruencies:

$$x \equiv a_1 \ mod\ m_1$$

$$x \equiv a_2 \ mod\ m_2$$

$$x \equiv a_i \ mod\ m_i$$

Has a unique solution $mod\ m_1 m_2 \dots m_i$ when $m_1, m_2, \dots, m_i$ are pair-wise relatively prime.

Definition 5: [7]

A ring (R, +, ×) is defined as the set R with two binary operations" + (addition) and × (multiplication) on R", that satisfies the following axioms:

1. (R, +) is an Abelian group with identity "0".
2. The operation " × " is associative. That's a× $(b \times c) = (a \times b) \times c$ $for\ all\ a, b, c \in R$.
3. There is a multiplicative identity denoted by 1, with $1 \neq 0$ such that $1 \times a = a \times 1 = a$ for all a $\epsilon\ R$.
4. The operation " × " is distributive over +. That's $a \times (b + c) = (a \times b) + (a \times c)$ and $(b + c) \times a = (b \times a) + (c \times a)$ for all a, b, c $\epsilon\ R$.

Definition 6:

The set of units in a ring R forms a group under multiplication called the group of units of R.

Definition 7: Given the two elements a and b of the ring R: [8]

"a" is called a unit if there exits b in R such that a × b = 1, where 1 is the multiplicative identity of R.

Definition 8: [8]

The group of units of Zn is the multiplicative group

$Z_n^* = \{a \in Z_n \ni \gcd(a, n) = 1\}$.

If n is a prime, then: $Z_n^* = \{1,2,3,..,n-1\}$.

Definition 9:

The ring is a commutative ring if $a \times b = b \times a$ for all a, b ϵ R.

Definition 10: [9]

The Euler Phi function $\emptyset(n)$ is the number of integers between 1 and n that are relatively prime with n.

For $n = p_1^{a_1} p_2^{a_2} \ldots p_i^{a_i}$ the Euler Phi function is:

$$\emptyset(n) = (p_1 - 1)p_1^{a_1 - 1}(p_2 - 1)p_2^{a_2 - 1} \ldots (p_i - 1)p_i^{a_i - 1}$$

Definition 11:

The order of an element $a \in Z_n^*$ is the least positive integer t such that $a^t \equiv 1\ (mod\ n)$.

Theorem 1:

If $r \equiv s\ (mod\ \emptyset(n)),\ then\ a^r \equiv a^s\ (mod\ n)$ for all integers a. In other words, when working modulo n, exponents can be reduced modulo $\emptyset(n)$.

Definition 12:

If the order of an element $a \in Z_n^*$ is $\emptyset(n)$, then a is said to be a generator or a primitive element of Z_n^*. If Z_n^* has a generator then it is said to be cyclic.

Properties of a generator of Z_n^* :

1. Z_n^* has a generator "a" if and only if n = 2, 4, $p^t\ or\ 2p^t$ where p is an odd prime and t $\geq$ 1.
2. For each element b in the group, b= a^i .

3. If a is a generator of Z_n^*, then $Z_n^* = \{a^i \ (mod\ n) | \ 0 \leq i \leq \emptyset(n) - 1\}$.
4. If Z_n^* is cyclic then it has $\emptyset(\emptyset(n))$ generators.

Algorithm 1: (Finding a generator of Z_n^*)

---

1. Select an integer a, $2 \leq a \leq p - 2$.
2. Write $\emptyset(p) = p - 1 = p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$
3. For i=1 to k

   3.1 Compute $a^{\frac{p-1}{p_i}} \ mod\ p$

   3.2 If $a^{\frac{p-1}{p_i}} \ mod\ p$ is congruent to 1 mod p, then return to step 1.
4. Return with a as a generator.

Homomorphism and Isomorphism:

A function f from a group G to group G1 is homomorphism if the following condition is satisfied: For all elements a, b in the group G: [10]

f (a.b) = f (a) f (b)

A one to one onto homomorphism is called an isomorphism.

Let f: G → G1 be an onto homomorphism from G to G1, then G1 is cyclic if G is cyclic.

Also if G is isomorphic to G1, $G \cong G1$ and G1 is cyclic then so is G.

Note that every finite cyclic group of order n is isomorphic to Zn and every infinite cyclic group is isomorphic to Z.

Theorem 2:

Let G1, G2, G3, .., Gr be groups and let $G_1 \times G_2 \times ... \times G_r$ be their external product i.e., the group defined on the Cartesian product whose operation is defined point-wise.

Then, $G_1 \times G_2 \times ... \times G_r$ is cyclic if and only if |Gi| and |Gj| are relatively prime when $i \neq j$.

Lemma 1:

Let m and n be two positive integers, then $Z_m \times Z_n \cong Z_{mn}$ if and only if gcd(m, n) = 1. Also if n = n1.n2…nr then $Z_n \cong Z_{n_1} \times Z_{n_2} \times ... \times Z_{n_r}$ if and only if n1, n2,.., nr are pair-wise relatively prime numbers.

Theorem 3: (Fundamental Theorem of Finite Abelian Groups)

Every finite Abelian group is a direct product of cyclic groups of prime power order

Theorem 4:

The group of units of Zn is given by:

1. $U(Z_2) \cong \{0\}$
2. $U(Z_4) \cong Z_2$
3. $U(Z_{2^a}) \cong Z_{2^a} \times Z_{2^{a-2}}$ when a ≥ 3.
4. $U\left(Z_{p^a}\right) \cong Z_{p-1} \times Z_{p^{a-1}}$

Definition 13:

An irreducible polynomial is a polynomial that can't be written as a product of two polynomials with positive degrees.

Definition 14:

Given R is a commutative ring; a polynomial over R has the form:

$f(x) = a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0 \; where \; a_i \epsilon \; R \; and \; n \geq 0.$

a_i is called the *coefficient* of x^i in f(x).

Degree of f(x) is the highest power of x in f(x).

Polynomial ring R[x]: is the commutative ring formed by the set of all polynomials having coefficients from R.

The operations used in R[x] are the standard polynomial addition and multiplication.

Chapter Four

Literature Review

4.1 The Classical ElGamal Public Key Encryption Scheme:

It is the most popular and widely used cryptosystem. It is described in the setting of the multiplicative group Z_p^*. For more details see [11] and [12].

The multiplicative group of Z_p^* = {1, 2, .., p-1 } is a cyclic group generated by a generator from the group.

The following algorithms show how the ElGamal cryptosystem functions:

Algorithm 2: (Key Generation)

A should do the following:

1. Generate a large random prime p and find the generator α of Z_p^*.
2. Select a random integer a, $1 \leq a \leq p-2$ and compute $\alpha^a \ mod\ p$.
3. A's public Key is (p, α, α^a) and A's private Key is a.

Algorithm 3: (Encryption)

B should do the following:

1. Obtain A's authentic public key (p, α, α^a).
2. Represent the message as an integer m in the range {0, 1, 2,…, p-1}.
3. Select a random integer k, where $2 \leq k \leq p-2$.
4. Compute $\gamma \equiv \alpha^k \ mod\ p \ and \ \delta \equiv m.(\alpha^a)^k \ mod\ p$
5. Send the ciphertext c = (γ, δ) to A.

Algorithm 4: (Decryption)

A should do the following:

1. Use the private Key a to compute $\gamma^{p-1-a}\ mod\ p$.
2. Recover the message m by computing $c \equiv \gamma^{-a}.\delta\ mod\ p$

4.2 ElGamal Public Key Encryption Scheme over Finite Fields:

4.2.1 Introduction:

In [13], Smith and Gallian determined the structure of the group of units of the quotient ring $F_q[x]/< f(x) >$, where f (x) is a polynomial in $F_q[x]$. Then, according to this structure El-Kassar et al. [14] gave a characterization of quotient ring of polynomials over finite fields with cyclic group of units.

In [15], El-Gamal cryptosystem was extended in the setting of $Z_p[x]/< x^2 >$.

Later on in [16], El-Gamal cryptosystem was extended to the setting of quotient ring of polynomials having a cyclic group of units U ($Z_p[x]/< h(x) >$) where p is an odd prime. In [17], this modification was tested and evaluated according to reliability, efficiency and security.

Two cases for h(x) were studied [14]:

Case 1: h (x) = x^2 (hence h(x) is reducible polynomial of degree n=2)

Case 2: h(x) = $h_1(x)h_2(x) \dots h_r(x)$ (h(x) is a product of irreducible polynomials $h_i(x)$ of degrees d_i)

The elements of U ($Z_p[x]/< h(x) >$) can be described as follows:

U ($Z_p[x]/< h(x) >$) = $\{a_0 + a_1x + \cdots + a_{n-1}x^{n-1} | \ a_0, a_1, \dots, a_{n-1} \epsilon \ Z_p\}$.

4.2.2 Case 1: h(x) = x^2

The order of U ($Z_p[x]/x^2$) is $\emptyset(x^2) = p(p-1)$ and its elements can be described as:

$U(Z_p[x]/x^2) = \{c + d\,x \mid 1 \leq c \leq p-1 \text{ and } 0 \leq d \leq p-1\}$

The extended El-Gamal cryptosystem in this setting is given in the following three algorithms.

To generate the public and private keys, entity should use algorithm 5:

Algorithm 5: (Key Generation)

---

1. Generate a large random prime p and find $\emptyset(x^2)$.
2. Find the generator α (x) of U ($Z_p[x]/x^2$), that is write U ($Z_p[x]/x^2$) as:

   U ($Z_p[x]/x^2$) = $\{1, \alpha(x), \alpha^2(x), \dots, \alpha^{p^2-p-1}(x)\}$
3. Select a random integer a, $2 \leq a \leq \emptyset(x^2) - 1$.
4. Compute $\alpha^a(x)\ mod(x^2)$.
5. A's public key is (p, x^2, $\alpha(x), \alpha^a(x)$); A's private key is a.

In order to encrypt m(x) $\epsilon\ U(Z_p[x]/< x^2 >)$, entity B should use algorithm 6.

Algorithm 6: (Encryption algorithm)

---

1. Get the public key.
2. Select a random integer k, $2 \leq k \leq \emptyset(x^2) - 1$.
3. Represent the message as a polynomial m(x) $\epsilon\ U(Z_p[x]/< x^2 >)$.
4. Compute $\gamma(x) = \alpha^k(x) mod(x^2)\ and\ \delta(x) = m(x).(\alpha(x)^a)^k mod(x^2)$.

5. Send the ciphertext $(\gamma(x), \delta(x))$ to A.

For decryption, B uses the following algorithm.

Algorithm 7: (Decryption scheme)

---

1. Uses the private key a to compute $\gamma(x)^{p^2-p-a} mod(x^2)$.
2. Recover the plaintext m(x) by computing $\gamma(x)^{p^2-p-a}.\delta(x)\ mod(x^2)$.

4.2.3 Case 2: h(x) = $\mathbf{h_1(x)h_2(x)}$ ... $\mathbf{h_r(x)}$ and p=2

The powers d_i of each of the $h_i(x)$ are pair-wise relatively prime and the order of U ($Z_2[x]/< h(x) >$) is $\emptyset\big(h(x)\big) = (2^{d_1} - 1)(2^{d_2} - 1)..(2^{d_i} - 1)$.

For generating the keys, entity A should use algorithm 8:

Algorithm 8: (Key Generation)

---

1. Choose pair-wise relatively prime numbers d_1, d_2 ,.., d_r.
2. Find irreducible polynomials $h_1(x), h_2(x), \dots, h_r(x)$ over Z_2 of degrees d_j.
3. Form h(x)= $h_1(x)h_2(x) \dots h_r(x)$
4. Find $\emptyset$(h(x)) $= (2^{d_1} - 1)(2^{d_2} - 1)..(2^{d_i} - 1)$.
5. Find the generator α (x) of U (Z_p[x]/$< h(x) >$).
6. Select a random integer a, $2 \leq a \leq \emptyset(h(x)) - 1$.
7. Compute $\alpha^a(x)\ mod(h(x))$.
8. A's public key is (h(x), $\alpha(x), \alpha^a(x)$); A's private key is a.

In order to encrypt m(x) $\epsilon\ U(Z_p[x]/< h(x) >)$, entity B should use algorithm 9.

Algorithm 9: (Encryption Scheme)

---

1. Obtain A's public key.
2. Select a random integer k, $2 \leq k \leq \emptyset(h(x)) - 1$.
3. Represent the message as a polynomial m(x) $\epsilon\, U(Z_2[x]/< h(x) >)$.
4. Compute $\gamma(x) = \alpha^k(x) mod(h(x))$ & $\delta(x) = m(x).(\alpha(x)^a)^k mod(h(x))$.
5. Send the ciphertext $(\gamma(x), \delta(x))$ to A.

For decryption, entity B uses the following algorithm.

Algorithm 10: (Decryption scheme)

---

1. Uses the private key a to compute $\gamma(x)^{\emptyset(h(x))-a} mod(h(x))$.
2. Recover the plaintext m(x) by computing $\gamma(x)^{-a}.\delta(x)\ mod(h(x))$.

Chapter Five

El-Gamal Cryptosystem over the second group of units

5.1 The Second Group of Units of Z_n

Theorem 5: (Fundamental Theorem of Abelian Groups) Every finite Abelian group is a direct product of cyclic groups of prime power order.

Let R be a finite commutative ring with identity. By the fundamental theorem of finite Abelian groups, the group of units, U (R), is isomorphic to the direct product of cyclic groups, say $U(R) \cong Z_{n_1} \times Z_{n_2} \times ... \times Z_{n_i}$. Hence, the multiplicative group U (R) supports a ring structure by defining the operations $\oplus \, and \otimes$ on U (R) that make $(U(R), \oplus, \otimes)$ a ring isomorphic to $Z_{n_1} \oplus Z_{n_2} \oplus ... \oplus Z_{n_i}$. The ring $Z_{n_1} \oplus Z_{n_2} \oplus ... \oplus Z_{n_i}$ will be denoted by R^2.

Definition 15: In [18], the second group of units of Z_n is defined as the group of units of the ring $U(Z_n)$:

$U^2(Z_n) = U(U(Z_n)) \cong U(R^2)$**.**

The paper considered the problem of determining the values of n that make $U^2(Z_n)$ cyclic. The result was as follows:

Theorem 6: Let p and q be odd prime integers and α be a positive integer. Then, $U^2(Z_n)$ is cyclic iff one of the following is true:

1. $n = 2^{\alpha}.3.p$,where $\alpha = 1,2 \; or \; 3$;
2. n = 15 ;
3. n=3.p, where p=4k+3 and 2k+1=q^{α};

4. $n = 2.3^{\alpha}, 2^{2}.3^{\alpha}, 2^{3}.3^{\alpha} or\ 2^{4}.3$;

5. n= 2 , 4, 8, or 16;

6. n= 5 or $2p^{\alpha}+1$, where$2p^{\alpha}+1$ is a prime integer;

7. n=3^{α};

5.1.1 Classification: These results can be classified into two cases:

Case 1: Both U (Z_n) and $U^2(Z_n)$ are cyclic:

As known U (Z_n) is cyclic when n = 2, 4, $p^{\alpha} or\ 2p^{\alpha}$ where $\alpha \geq 1$ and p is an odd prime integer. So out of the previous seven cases for n, only the following belong to case one:

1. $n = 2.3^{\alpha}$ since n is in the form of $2p^{\alpha}\ where\ p = 3$;
2. $n = 2\ or\ 4$
3. n = 5 or $2p^{\alpha}+1$ since $2p^{\alpha}+1$ is a prime integer, so it's in the form of $p^{\alpha}\ where\ \alpha = 1$;
4. n =3^{α} since n is in the form of $p^{\alpha}\ where\ p = 3$;

Case 2: $U^2(Z_n)$ is cyclic whereas, U (Z_n) is not cyclic:

1. $n = 2^{\alpha}.3.p$, where $\alpha = 1,2\ or\ 3$;
2. n = 15;
3. n = 3.p, where p=4k+3 and 2k+1= q^{α};
4. $n = 2^{2}.3^{\alpha}, 2^{3}.3^{\alpha} or\ 2^{4}.3$;
5. n = 4 or 16;

5.2 Construction of $U^2(Z_n)$ in case 1

5.2.1 Construction: to construct the second group of units $U^2(Z_n)$, we follow the following steps:

1. Form the group of units $U(Z_n)$, where $U(Z_n)=\{a \in Z_n : \gcd(a,n) = 1\}$. The order of $U(Z_n)$ is $\varphi(n)$.
2. Find a generator r of $U(Z_n)$.
3. Write $U(Z_n)$ in the form: $\{r^0, r^1, \dots, r^{\varphi(n)-1}\}$.
4. Find $U^2(Z_n)$, $U^2(Z_n) = \{r^i : \gcd(i, \varphi(n)) = 1\}$.
5. Write $U^2(Z_n)$ as $U^2(Z_n) = \{x, x \equiv r^i \ mod \ n\}$.

Example 1: n = 23, U (Z_{23}) = {1, 2, 3, …, 22} and $\varphi(23) = 22$. The generator of U (Z_{23}) is r =5.

U_{23}	1	2	3	4	5	6	7	8	9	10	11
5^i	5^0	5^2	5^{16}	5^4	5^1	5^{18}	5^{19}	5^6	5^{10}	5^3	5^9

$U(Z_{23})$	12	13	14	15	16	17	18	19	20	21	22
5^i	5^{20}	5^{14}	5^{21}	5^{17}	5^8	5^7	5^{12}	5^{15}	5^5	5^{13}	5^{11}

Table 5.1 Elements of U_{23}

$U^2(Z_{23}) = \{5^i : \gcd(i, 22) = 1\} = \{5^1, 5^{19}, 5^3, 5^9, 5^{21}, 5^{17}, 5^7, 5^{15}, 5^5, 5^{13}\}$

Then $U^2(Z_{23}) = \{5, 10, 20, 17, 11, 21, 19, 15, 7, 14\}$. The order of $U^2(Z_{23})$ is $\varphi(\varphi(23)) = 10$.

5.2.2 Operations: In [4], the operations over the ring (U (Z_n), . , $\otimes$) are defined as follows:

1. *Addition operation*: x . y = xy mod n.

2. *Multiplication operation*: $x \otimes y = x^{\log_r y}(mod\ n)$ where r is the generator of U (Z_n).

3. *The power operation:* Let θ be an element of $U^2(Z_n)$ such that $\theta = r^k$, r is the generator of U (Z_n), then $\theta^N = (r)^{k^N}$.

Identity: The identity of the second group of units of Z_n is r [4], where r is the generator of U (Z_n).

5.3 Construction of $U^2(Z_n)$ in case 2

Lemma 2: Let n and m be any two positive integers, then $Z_{mn} \cong Z_n \times Z_m\ iff\ (m,n) = 1$.

Also, if $n = n_1.n_2 \ldots n_r\ then\ Z_n \cong Z_{n_1} \times Z_{n_2} \times \ldots \times Z_{n_r}$ iff $n_1, n_2 \ldots, n_r$ are pairwise relatively prime.

Lemma 3:

1. $U(Z_2) \cong \{0\}$.
2. $U(Z_4) \cong Z_2$.
3. $U\left(Z_{2^k}\right) \cong Z_2 \oplus Z_{2^{k-2}} for\ k \geq 3$.
4. $U\left(Z_{p^n}\right) \cong Z_{p^{n-1}} \oplus Z_{p-1}$

Theorem 7: If $R = R_1 \oplus R_2 \ldots \oplus R_i \Rightarrow U(R) = U(R_1) \times U(R_2) \ldots U(R_i)$.

We will work on the case where n= 3p:

$U(Z_n) \cong U(Z_3) \times U(Z_p) \cong Z_2 \times Z_{p-1} \Rightarrow R^2 = (Z_2 \oplus Z_{p-1})$.

$U^2(R) \cong U(R^2) = U(Z_2 \oplus Z_{p-1}) \cong U(Z_2) \times U(Z_{p-1}) \cong U(Z_{p-1})$.

Definition 16: Isomorphism Functions:

1. The isomorphism function f:$U(Z_{mn}) \to U(Z_m) \times U(Z_n) \ni (m,n) = 1$ is:

for every a ∈ $U(Z_{mn})$, f(a) = (a (mod n), a (mod m)).

2. The isomorphism function $f_1 : U(Z_n) \to (Z_{n-1})$ is $f_1 : a \to \log_r a$ where r is the generator of $U(Z_n)$.

5.3.1 Construction of $U^2(Z_n)$:

1. Construct $U(Z_n)$, $U(Z_3)$ and $U(Z_p)$.

2. Find G = {(a (mod 3), a (mod p)): a ∈ $U(Z_n)$}.

3. Find the generator r of $U(Z_3)$ and $r_1$the generator of $U(Z_p)$.

4. Write G in the form:

$\{\left(r^t (mod\ 3), r_1^{t^1} (mod\ p)\right) : 0 \le t \le \varphi(3) - 1\ and\ 0 \le t^1 \le \varphi(p) - 1\}$.

5. Form the set $Z_2 \oplus Z_{p-1} = \{(t, t^1)\}$ and find G1, the set of its invertible elements. Note that: (a, b) is invertible in $Z_2 \oplus Z_{p-1}$ if a is invertible in Z_2 and b is invertible in Z_{p-1}.

6. $U^2(Z_n)$ is the set of elements in $U(Z_n)$ corresponding to elements in G1.

Example 2:

Let p = 11, $U(Z_{33}) \cong U(Z_3) \times U(Z_{11}) \cong Z_2 \oplus Z_{10}$.

$U(Z_{33}) = \{1,2,4,5,7,8,10,13,14,16,17,19,20,23,25,26,28,29,31,32\}$.

$U(Z_{11}) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, 2 is a generator of $U(Z_{11})$.

$U(Z_3) = \{1, 2\}$, 2 is a generator of $U(Z_3)$.

$U(Z_{33})$	1	2	4	5	7	8	10
$U(Z_3) \times U(Z_{11})$	(1,1)	(2,2)	(1,4)	(2,5)	(1,7)	(2,8)	(1,10)
$(2^i, 2^j)$	$(2^0, 2^0)$	$(2^1, 2^1)$	$(2^0, 2^2)$	$(2^1, 2^4)$	$(2^0, 2^7)$	$(2^1, 2^3)$	$(2^0, 2^5)$
$Z_2 \oplus Z_{10}$	(0,0)	(1,1)	(0,2)	(1,4)	(0,7)	(1,3)	(0,5)

$U(Z_{33})$	13	14	16	17	19	20	23
$U(Z_3) \times U(Z_{11})$	(1,2)	(2,3)	(1,5)	(2,6)	(1,8)	(2,9)	(2,1)
$(2^i, 2^j)$	$(2^0, 2^1)$	$(2^1, 2^8)$	$(2^0, 2^4)$	$(2^1, 2^9)$	$(2^0, 2^3)$	$(2^1, 2^6)$	$(2^1, 2^0)$
$Z_2 \oplus Z_{10}$	(0,1)	(1,8)	(0,4)	(1,9)	(0,3)	(1,6)	(1,0)

$U(Z_{33})$	25	26	28	29	31	32
$U(Z_3) \times U(Z_{11})$	(1,3)	(2,4)	(1,6)	(2,7)	(1,9)	(2,10)
$(2^i, 2^j)$	$(2^0, 2^8)$	$(2^1, 2^2)$	$(2^0, 2^9)$	$(2^1, 2^7)$	$(2^0, 2^6)$	$(2^1, 2^5)$
$Z_2 \oplus Z_{10}$	(0,8)	(1,2)	(0,9)	(1,7)	(0,6)	(1,5)

Table 5.3 Table of isomorphism between U_{33} and $U_3 \times U_{11}$

Invertible elements in $Z_2 \oplus Z_{10}$ are: {(1, 1), (1, 3), (1, 9), (1, 7)}. Then:

$U^2(Z_{33}) = \{2, 8, 17, 29\}$.

5.3.2 Operations:

1. Multiplication: $(U^2(Z_n), \odot)$

a $\odot$ $b = f^{-1}(f(a) \otimes f(b))$ where f :$U(Z_n) \to U(Z_3) \times U(Z_p)$.

Suppose that: f (a) = (c, d) and f (b) = (e, g).

$f(a) \otimes f(b) = (c, d) \otimes (e, g) = (c \otimes_3 e, d \otimes_p g)$.

Define:$\otimes_3$: x $\otimes_3$y = $x^{\log_r y}$, where r is the generator of $U(Z_3)$.

$\otimes_p$: x $\otimes_p$ y = $x^{\log_{r_1} y}$, where $r_1$is the generator of $U(Z_p)$.

2. Power operation: f :$U(Z_n) \to U(Z_3) \times U(Z_p)$.

Let a ϵ $U^2(Z_n) \ni f(a) = (c, d) = (r^t, {r_1}^{t_1})$, then $a^N = f^{-1}(r^{t^N}, {r_1}^{t_1^N})$.

Proof: $a^2 = a \odot a = f^{-1}\big((r^t, {r_1}^{t_1}) \otimes (r^t, {r_1}^{t_1})\big) = f^{-1}(r^t \otimes_3 r^t, {r_1}^{t_1} \otimes_p {r_1}^{t_1})$

$= f^{-1}(r^{t^2}, {r_1}^{t_1^2})$

For induction purpose, assume that the result is true for N-1:

So,$a^{N-1} = f^{-1}\left(r^{t^{N-1}}, {r_1}^{t_1^{N-1}}\right)$then,

$a^N = a \odot a^{N-1} = \left(f^{-1}(r^t, {r_1}^{t_1})\right) \odot \left(f^{-1}\left(r^{t^{N-1}}, {r_1}^{t_1^{N-1}}\right)\right)$

$= f^{-1}\left((r^t, {r_1}^{t_1}) \otimes \left(r^{t^{N-1}}, {r_1}^{t_1^{N-1}}\right)\right)$

$= f^{-1}(r^t \otimes_3 r^{t^{N-1}}, {r_1}^{t_1} \otimes_p {r_1}^{{t_1}^{N-1}}) = f^{-1}(r^{t^N}, {r_1}^{t_1^N})$

Identity: The identity of $U^2(Z_n)$, is the element a ϵ $U^2(Z_n) \ni a = f^{-1}(r, r_1)$, where r is the generator of $U(Z_3)$and $r_1$is the generator of $U(Z_p)$.

5.4 ElGamal Cryptosystem over $U^2(Z_n)$ for case 1:

Finding a generator of $U^2(Z_n)$ for case 1:

Algorithm 11 *(Generator of $U^2(Z_n)$)*

---

1. Find a generator $\theta 1$ of $U(Z_n)$.

2. Write the order of $U^2(Z_n)$ as $p_1^{\alpha_1} p_2^{\alpha_2} \dots p_i^{\alpha_i}$.

3. Select a random integer s, $0 \leq s \leq \varphi(n) - 1 \ni (s,\ \varphi(n)) = 1$.

4. For j = 1 to i, do:

    4.1 Compute $\theta 1^{\frac{N}{p_j}}$ mod n.

    4.2 If $\theta 1^{s^{(\frac{N}{p_j})}}$ mod n = $\theta 1$, then go to step 3.

5. Return s

For key generation, entity A does the following:

Algorithm 12*(Key Generation)*

---

1. Find a generator $\theta 1$ of $U(Z_n)$.

2. Find s using Algorithm 1.

3. Compute the order of $U^2(Z_n)$ using $Order = \varphi(\varphi(n))$.

4. Select a random integer a,$2 \leq a \leq \varphi\big(\varphi(n)\big) - 1$, compute $f = s^a \ (mod\ \varphi(n))$.

5. A's public key is $(n, \theta 1,\ s,\ f)$ and A's private key is a.

B encrypts a message m for A using the algorithm below:

Algorithm 13 *(Encryption)*

---

1. B obtains A's authentic public key $(n, \theta 1,\ s,\ s^a)$.

2. Represent the message as an integer in $U^2(Z_n)$.

3. Select a random integer k, where $2 \leq k \leq \varphi\big(\varphi(n)\big) - 1$.

4. Compute $q = s^k\ (mod\ \varphi(n))$, $r = f^k (mod\ \varphi(n))$, $\gamma = \theta^k = \theta 1^q\ (mod\ n)$ and $\delta \equiv m^r\ (mod\ n)$.

5. Send the ciphertext c = (q, δ) to A.

To recover the plaintext m from c, A should do the following:

Algorithm 14 *(Decryption)*

---

1. Use the private key "a" to compute $b = \varphi(\varphi(n))$ - a.

2. Recover the message by computing $t = q^b\big(mod\ \varphi(n)\big) and\ \delta^t\ (mod\ n)$.

The following theorem proves that the decryption formula $\gamma^{-a}.\delta\ (mod\ n)$ allows the recovery of the message m. This proof is for case 1.

Theorem 8: Given a generator θ of $U^2(Z_n)$ such that $\theta = \theta 1^s$, where $\theta 1$ is the generator of $U(Z_n)$. Define $\gamma \equiv \theta^k\ (mod\ n)$ and $\delta \equiv m.(\theta^a)^k\ (mod\ n)$. If $s \in U^2(Z_n)$ such that s $\equiv \gamma^{-a}.\delta\ (mod\ n)$, then s = m.

Proof: $\gamma \equiv \theta 1^{s^k} (mod\ n) \Rightarrow$

$$\gamma^{-a} \equiv \theta 1^{s^{-a.k}} (mod\ n)\ and\ \delta \equiv m.(\theta^a)^k \equiv m.\theta 1^{s^{a.k}} (mod\ n).$$

$$Hence, s \equiv \gamma^{-a}.\delta \equiv \theta 1^{s^{-a.k}}.m.\theta 1^{s^{a.k}} (mod\ n) \equiv \theta 1^{s^{-a.k}}.\theta 1^{s^{a.k}}.m$$

$$\equiv \left(\theta 1^{s^{-ak}}\right)^{\log_{\theta 1}\theta 1^{s^{ak}}}.m \equiv \theta 1^{s^{-ak}.s^{ak}}.m$$

$$\equiv \theta 1^{s^0}.m \equiv \theta 1.m \equiv m\ (mod\ n).$$

But, m and s belong to the same complete residue system modulo n and s ≡ m (mod n) ⇒ m = s.

Example 4: In this example, we will work on the case n = 3^{α}.

Let $\alpha = 3$, θ = 5 is a generator of U (Z_{27}) and $\theta_1 = 2$ is a generator of $U^2(Z_{27})$. A uses a = 3 and calculates $f = \theta^a = 2^3 \equiv 5^{11^3} \equiv 11\ (mod\ 27)$.

So, A′spublic key is $(n = 27, \theta = 5,\ s = 11, s^a = 11^3 mod\ \big(\varphi(27)\big) = 17)$.

To encrypt m = 5, B selects an integer k = 4 and finds $q = s^k = 11^4\ \big(mod\ \varphi(27)\big) = 7$, $r = f^k = 17^4 (mod\ \varphi(27)) = 1$ and $\delta \equiv 5^r = 5\ (mod\ 27)$. B sends $(s^k = 7, \delta = 5)$ to A.

Finally, A computes $b = \varphi\big(\varphi(n)\big) - a = 2$ and $t = q^b \big(mod\ \varphi(27)\big) = 7^2 (mod\ 6) = 1$. Then finds $\delta^t\ mod(27) = 5^1 = 5\ (mod\ 27) = m$

5.5 ElGamal Cryptosystem over $U^2(Z_n)$ for case 2:

Finding a generator of $U^2(Z_n)$ for case 2:

Suppose that n = 3.p. Note that the generator of $U(Z_3)\ is\ 2$.

Algorithm 15*(Generator of $U^2(Z_n)$)*

1. Find a generator $\theta 1$ of $U(Z_p)$.

2. Compute the order of $U^2(Z_n)$, $\varphi(\varphi(p))$.

3. Write the order of $U^2(Z_n)$ as $p_1^{\alpha_1} p_2^{\alpha_2} \dots p_i^{\alpha_i}$.

4. Select a random integer s, $0 \leq s \leq \varphi(p) - 1 \ni (s,\ \varphi(p)) = 1$.

5. For j = 1 to i, do:

   4.1 Compute $\theta 1^{\frac{N}{p_j}}$ mod p.

   4.2 If $\theta 1^{s^{(\frac{N}{p_j})}}$ mod p = $\theta 1$, then go to step 4.

6. Use the Chinese Remainder Theorem to find θ, and s, i.e. by solving the system of congruencies:

$x \equiv 2\ (mod\ 3)\ and\ x \equiv \theta 1^s\ (mod\ p)$.

7. Return s.

Suppose that n =3.p. For key generation, A follows the steps below:

Algorithm 16*(Key Generation)*

1. Find a generator $\theta 1$ of $U(Z_n)$.

2. Find s using Algorithm 15.

3. Compute the order of $U^2(Z_n)$ using $Order = \varphi(\varphi(p))$.

4. Select a random integer a, $2 \leq a \leq \varphi(\varphi(p)) - 1$, compute $f = s^a\ (mod\ \varphi(p))$.

5. A's public key is $(p, \theta 1,\ s,\ f)$ and A's private key is a.

B encrypts a message m for A using the algorithm below:

Algorithm 17 *(Encryption)*

---

1. B obtains A's authentic public key $(p, \theta 1,\ s,\ s^a)$.

2. Represent the message as an integer in $U^2(Z_p)$.

3. Select a random integer k, where $2 \leq k \leq \varphi(\varphi(p)) - 1$.

4. Compute $q = s^k\ (mod\ \varphi(p))$, $r = f^k (mod\ \varphi(p))$, $\gamma = \theta^k = \theta 1^q\ (mod\ p)$ and $\delta \equiv m^r\ (mod\ p)$.

5. Send the ciphertext c = (q, δ) to A.

To recover the plaintext m from c, A should do the following:

Algorithm 18 *(Decryption)*

---

1. Use the private key "a" to compute $b = \varphi(\varphi(p))$ - a.

2. Recover the message by computing $t = q^b(mod\ \varphi(p))and\ \delta^t\ (mod\ p)$.

Theorem 9: Suppose that $n = 3.p$ and let θ be a generator of $U^2(Z_n)$ such that $\theta = (2,\theta_1{}^s)$, where $\theta 1$ is the generator of $U(Z_p)$. m $\epsilon\, U(Z_p)$. Define $\gamma \equiv \theta^k\ (mod\ n)$ and $\delta \equiv m.(\theta^a)^k\ (mod\ n)$. If $s \in U(Z_p)$ such that s $\equiv \gamma^{-a}.\delta\ (mod\ n)$, then s = m.

Proof: $\gamma \equiv (2, \theta 1^{s^k})\ (mod\ p) \Rightarrow \gamma^{-a} \equiv (2, \theta 1^{s^{-a.k}})\ (mod\ p)$

$$and\ \delta \equiv m.(\theta^a)^k \equiv m.(2, \theta 1^{s^{a.k}})(mod\ p).$$

$$Hence, s \equiv \gamma^{-a}.\delta \equiv (2, \theta 1^{s^{-a.k}}).m.\left(2, \theta 1^{s^{a.k}}\right)(mod\ p)$$

$$\equiv \left(2, \theta 1^{s^{-a.k}}\right).\left(2, \theta 1^{s^{a.k}}\right).m$$

$$\equiv (2, \theta 1^{s^{-ak}.s^{ak}}).m \equiv (2, \theta 1^{s^0}).m$$

$$\equiv (2, \theta 1).m \equiv m\ (mod\ n)$$

But, m and s belong to the same complete residue system modulo n and s ≡ m (mod n) ⇒ m = s.

Example 5: In this example, we will work on n=3.p from case 2.

Let p=11 so, $\varphi(n) = \varphi(33) = 18$.

$\theta = 29$ is a generator of $U^2(Z_{33})$,

$\theta_1 = 2$ is a generator of $U(Z_{11})$, s = 7.

A uses a=3 and finds $f = s^a = 7^3 \equiv 3\big(mod\varphi(11)\big)$. So, A′s public key is

$(p = 11, \theta 1 = 2,\ s = 7, s^a = 3)$.

To encrypt the message m = 5, B selects a random integer k = 2 and computes $s^k = 7^2\big(mod\ \varphi(11)\big) = 9, \gamma \equiv (2, \theta 1^{s^k} mod\ 11) \equiv (2,6)\ and\ \delta \equiv 5.(2,6)$.

Finally, A computes $b = 1\ and\ \gamma^1.\delta \equiv (2,6).5.(2,6) \equiv 5.(2,6).(2,6) \equiv 5.(2, 2^{9.9}) \equiv 5.(2,2) \equiv 5$.

Chapter Six

ElGamal Cryptosystem over Second Group of Units of $Z_2[x]/< h(x) >$

6.1 The Second Group of Units of $Z_2[x]/< h(x) >$

Suppose that h(x) is an irreducible polynomial of degree n, then $Z_2[x]/< h(x) > = \{a_0 + a_1x + \cdots + a_{n-1}x^{n-1}: a_0, a_1, \dots, a_{n-1} \epsilon Z_2\}$ is a field whose elements are the polynomials in $Z_2[x]$ with degree less than that of h(x).

The order of $Z_2[x]/< h(x) >$ is 2^n, its nonzero elements form a cyclic group$U(Z_2[x]/< h(x) >)$ of order $\varphi\big(h(x)\big) = 2^n - 1$. See [3] for more details.

[4] Contains a study of the values of n that make $U^2(Z_2[x]/< h(x) >)$ a cyclic group.

Theorem 10: $U^2(Z_2[x]/< h(x) >)$ is cyclic iff one of the following conditions is satisfied:

1. $2^n = q^\alpha + 1$ where n, the degree of h(x), is prime (q is an odd prime and $\alpha > 0$).
2. $2^n = q + 1$ where, q is a Mersenne prime.

6.2 Construction of $U^2(Z_2[x]/< h(x) >)$:

Since $U(Z_2[x]/< h(x) >)$ is cyclic for any value of n, and then if n satisfies Theorem 3 both of $U(Z_2[x]/< h(x) >)$ and $U^2(Z_2[x]/< h(x) >)$ would be cyclic groups.

Hence to construct $U^2(Z_2[x]/< h(x) >)$:

1. Form the group of units $U(Z_2[x]/< h(x) >)$.
2. Find a generator r of $U(Z_2[x]/< h(x) >)$.
3. Write $U(Z_2[x]/< h(x) >)$ in the form: $\{r^0, r^1, \dots, r^{\varphi(h(x))-1}\}$.
4. Find $U^2(Z_2[x]/< h(x) >) = \{r^i : gcd\ (i, 2^n - 1) = 1\}$.
5. Write $U^2(Z_2[x]/< h(x) >)\ as:$

   $U^2(Z_2[x]/< h(x) >) = \{x, x \equiv r^i\ mod\ h(x)\}$.

Example 6: Given h(x) = $x^3 + x + 1$.

$Z_2[x]/< x^3 + x + 1 > = \{0,1, x, x^2, 1 + x, 1 + x^2, x + x^2, 1 + x + x^2\}$.

$U(Z_2[x]/< x^3 + x + 1 >) = \{1, x, x^2, 1 + x, 1 + x^2, x + x^2, 1 + x + x^2\}$ and

r = 1+ x is a generator.

$U(Z_2[x]/< x^3 + x + 1 >)$	1	x	x^2	1+x
$(1 + x)^i$	$(1 + x)^0$	$(1 + x)^5$	$(1 + x)^3$	$(1 + x)^1$

$U(Z_2[x]/< x^3 + x + 1 >)$	$1 + x^2$	$x + x^2$	$1 + x + x^2$
$(1 + x)^i$	$(1 + x)^2$	$(1 + x)^6$	$(1 + x)^4$

Table 6.2 Elements of $U(Z_2[x]/< x^3 + x + 1 >)$

$U^2(Z_2[x]/< x^3 + x + 1 >) = \{(x + 1)^i : \gcd(i, 2^3 - 1) = 1\}$.

$U^2(Z_2[x]/< x^3 + x + 1 >) = \{x, x^2, 1 + x, 1 + x^2, x + x^2, 1 + x + x^2\}$.

6.3 ElGamal Cryptosystem over $U^2(Z_2[x]/< h(x) >)$

In a previous work, ElGamal cryptosystem was employed in the setting of

$U(Z_2[x]/< h(x) >)$.

For more details on this work see [15]. In this work we extend ElGamal scheme to the setting of the second group of units of $(Z_2[x]/< h(x) >)$.

Finding the generator of $U^2(Z_2[x]/< h(x) >)$:

Algorithm 19*(Generator of $U^2\big((Z_2[x]/< h(x) >)\big)$*

---

1. Find a generator $\theta 1(x)$ of $U\big((Z_2[x]/< h(x) >)\big)$.

2. Write the order of $U^2\big((Z_2[x]/< h(x) >)\big)$ as $\varphi(h(x) = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_i^{\alpha_i}$.

3. Select a random integer s, $0 \leq s \leq \varphi(h(x)) - 1 \ni (s,\ \varphi(h(x))) = 1$.

4. For j = 1 to i, do:

   4.1 Compute $\theta 1(x)^{\frac{N}{p_j}} \bmod h(x)$.

   4.2 If $\theta 1(x)^{s^{(\frac{N}{p_j})}} \bmod h(x) = \theta 1(x)$, then go to step 3.

5. Return s.

At the first place in order to generate the corresponding public and private keys, entity A follows the steps below:

Algorithm 20 *(Key Generation)*

---

1. Select an irreducible polynomial h(x) of degree n.

2. Find a generator $\theta 1(x)$ of $U(Z_2[x]/< h(x) >)$.

2. Find s using Algorithm 19.

3. Compute the order of $U^2(Z_2[x]/< h(x) >)$ using $Order = \varphi(\varphi(h(x)))$.

4. Select a random integer a, $2 \leq a \leq \varphi\big(\varphi(h(x))\big) - 1$, compute

$f = s^a \ (mod\ \varphi(h(x)))$.

5. A's public key is $(n, h(x), \theta 1(x),\ s,\ f)$ and A's private key is a.

B encrypts a message m(x) for A using the algorithm below:

Algorithm 21 *(Encryption)*

---

1. B obtains A's authentic public key $(n, h(x), \theta 1(x),\ s,\ f)$.

2. Choose a message m(x) from $U^2\big((\boldsymbol{Z_2}[\boldsymbol{x}]/< h(x) >)\big)$.

3. Select a random integer k, where $2 \leq k \leq \varphi\big(\varphi(h(x))\big) - 1$.

4. Compute $q = s^k \ (mod\ \varphi(h(x)))$, and $r = f^k (mod\ \varphi(h(x)))$, $\gamma(x) = \theta(x)^k = \theta 1(x)^q \ \big(mod\ h(x)\big)\ and\ \delta(x) \equiv m(x)^r \ (mod\ h(x))$.

5. Send the ciphertext c = $(q, \delta(x))$ to A.

To recover the plaintext m from c, A should do the following:

Algorithm 22 *(Decryption)*

---

1. Use the private key "a" to compute $b = \varphi(\varphi(h(x)))$ - a.

2. Recover the message by computing $t = q^b\big(mod\ \varphi(h(x))\big)\ and\ \delta(x)^t \ (mod\ h(x))$.

Theorem 10: Given a generator $\theta(x)$ of $U^2(Z_2[x]/< h(x) >)$ such that $\theta(x) = \theta 1(x)^s$, where $\theta 1(x)$ is the generator of $U(Z_2[x]/< h(x) >)$. Define $\gamma(x) \equiv \theta(x)^k \ (mod\ h(x))$ and $\delta(x) \equiv m(x).(\theta(x)^a)^k \ (mod\ h(x))$.

If $s(x) \in U^2\big((\boldsymbol{Z_2}[\boldsymbol{x}]/< h(x) >)\big)$ such that s(x) $\equiv \gamma(x)^{-a}.\delta(x) \ (mod\ h(x))$, then s(x) = m(x).

Proof: $\gamma(x) \equiv \theta 1(x)^{s^k}\big(mod\ h(x)\big) \Rightarrow \gamma(x)^{-a} \equiv \theta 1(x)^{s^{-a.k}}\big(mod\ h(x)\big)\ and$

$$\delta(x) \equiv m(x).(\theta(x)^a)^k \equiv m(x).\theta 1(x)^{s^{a.k}}\big(mod\ h(x)\big).$$

$$s(x) \equiv \gamma(x)^{-a}.\delta(x) \equiv \theta 1(x)^{s^{-a.k}}.m(x).\theta 1(x)^{s^{a.k}}(mod\ h(x)$$

$$\equiv \theta 1(x)^{s^{-a.k}}.\theta 1(x)^{s^{a.k}}.m(x) \equiv \left(\theta 1(x)^{s^{-ak}}\right)^{\log_{\theta_1}\theta 1^{s^{ak}}}.m(x)$$

$$\equiv \theta 1(x)^{s^{-ak}.s^{ak}}.m(x) \equiv \theta 1(x)^{s^0}.m(x) \equiv \theta 1(x).m(x)$$

$$\equiv m(x)\ (mod\ h(x))$$

Since m(x) and s(x) are in the same complete residue system modulo h(x) and

s(x) ≡ m(x) (mod h(x)) ⇒ m(x) = s(x).

Example 7: Take n=3 and choose $h(x) = x^3 + x + 1$.

$Z_2[x]/< x^3 + x + 1 >= \{0,1,x,x^2,x+1,1+x^2,x+x^2,1+x+x^2\}$.

$U(Z_2[x]/< x^3 + x + 1 >) = \{1,x,x^2,x+1,1+x^2,x+x^2,1+x+x^2\}$.

$U^2(Z_2[x]/< x^3 + x + 1 >) = \{x,x^2,x+1,1+x^2,x+x^2,1+x+x^2\}$.

A finds a generator $\theta(x) = x$ of $U^2(Z_2[x]/< x^3 + x + 1 >)$, $\theta 1(x) = x + 1$ and s = 5.

Then A selects an integer a = 4 and computes $f = s^a = 5^4 (mod\ \varphi(7)) = 1$.

Then A's public key is $(n = 3, h(x) = x^3 + x + 1, \theta 1(x) = 1 + x, s = 5, f = 2)$.

To encrypt the message m(x) =x^2, B selects a random integer k = 3 and computes,

$r = s^k (mod\ \varphi(7)) = 5^3 = 5\ (mod\ 6)$,

$q = f^k (mod\ \varphi(7)) = 1^3\ mod\ 6 = 1$ and,

$\delta(x) = m(x)^q = (x^2)^1 = x^2 (mod x^3 + x + 1)$

And sends the ciphertext $(r, \delta(x))$ to A.

To decrypt the message A computes b = 6-4 = 2 and $t = r^b\ mod\ 6 = 5^2\ mod\ 6 = 1$.

Then computes $\delta(x)^t = (x^2)^1 = x^2 (mod\ x^3 + x + 1)$.

Chapter Seven

Testing and Evaluation

7.1 Implementation of the Modified ElGamal Cryptosystem:

In this section, we test and evaluate the modified cryptosystems by implementing modified algorithms. This is done using Mathematica 7.0 as a programming language and an hp computer with 1.73 GHZ CPU and 1014 MB RAM.

Using Mathematica 7.0, we have written programs for the following algorithms:

1. ElGamal with n in the form 2.3^{α}.
2. ElGamal with n in the form 4.3^{α}.
3. ElGamal with n in the form 3^{α}.
4. ElGamal with n in the form 3.p.
5. ElGamal with n in the form 3.2^{α}.p.
6. ElGamal with n in the form $2.p^{\alpha}+1$.
7. ElGamal over the polynomial case.

After running the programs, it was clear that all the programs have generated a public and private key. Then a message is encrypted and is sent to a decryption scheme which recovered the message.

It should be noted that, working in the second group of units is more secure than that of the first group of units, since if the hacker found the private key he has to perform

complicated multiplication and power operations and sometimes he has to work with isomorphism functions to find the plaintext.

Chart 7.1 and Table 7.1 show the results that were obtained after running the Mathematica programs for 20 times.

El-Gamal Cryptosystem Evaluation			
Algorithm	***Key Generation***	***Encryption***	***Decryption***
n=3^a	*972.158*	*5116.84*	*2801.47*
n=2.3^a	*1941.4*	*10302.2*	*5669.9*
n=2p^a+1	*2.4*	*2.35*	*7736.1*
n=4.3^a	*2651.25*	*14160.9*	*2.13333*
n=3.p	*2.3*	*6.81421*10^(-15)*	*6.81421*10^(-15)*
n=3.4.p	*2.35*	*0.75*	*5.14996*10^(-15)*
Poly case	*937.5*	*20.4*	*3.15*

Table 7.1 ElGamal Evaluation

Comparing these algorithms with each other, we conclude the following:

1. All programs are reliable; they can encrypt and decrypt any message.

2. For the irreducible polynomial case, it worked well but it took a considerable time to find an irreducible polynomial of high degree. Moreover, it takes more time to generate the public and private key than to decrypt or encrypt a message. Finally, it takes a considerable time to find some of the elements of $U(Z_2[x]/< h(x) >)$ when the degree of h(x) is high.

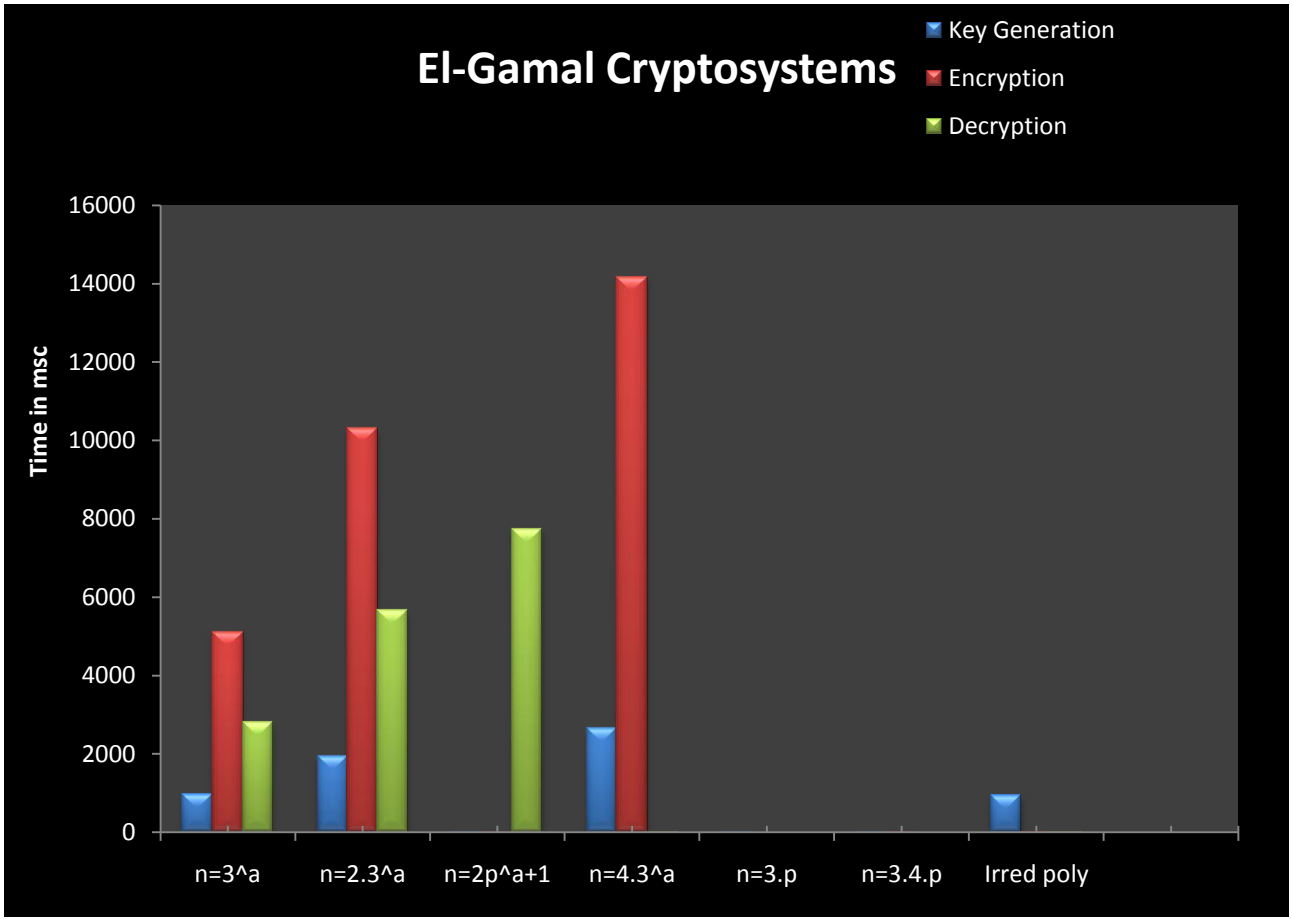

Figure 7.1 Testing results of the modified ElGamal cryptosystem

3. For cases 2, 3 and 4, the time needed for encryption is more than that needed for generating public and private keys and for decryption. But it works well even for a=10000.

4. Cases 5, 6 and 7 are more efficient than others since they work for every prime number even for primes consisting of 13 digits and they require less time for encryption, decryption and key generation.

5. The case where n = 3p is the most efficient one since it takes the least time for encryption, key generation and decryption.

6. In case 7, the time needed for decryption was the much more than that needed for key generation and encryption.

7.2 The Discrete Logarithm Problem: [11]

The security of the El-Gamal cryptosystem depends on the intractability of the discrete logarithm problem.

Discrete Logarithm Problem:

Let G be a finite cyclic group of order n. Let a be a generator of G, and $\beta \in G$. The Discrete Logarithm of β to the base a, denoted by $\log_a \beta$, is the unique integer x, $0 \leq x \leq n-1$,such that $\beta = a^x$.

7.3 The baby-step Giant-Step algorithm: [11]

To attack this cryptosystem we have to solve the discrete logarithm problem. The most popular attack algorithm is:

Let $m = \lceil \sqrt{n} \rceil$, where n is the order of a. If $\beta = a^x$, then one can write x = im + j, where $0 \leq i, j < m$. Hence, $a^x = a^{im} a^j$, which implies $\beta(a^{-m})^i = a^j$.

Algorithm 23: Baby-Step Giant-Step algorithm

INPUT: a generator of a cyclic group G of order n, and an element $\beta \in G$

OUTPUT: the discrete logarithm x= $\log_a \beta$.

1. Set $m = \lceil \sqrt{n} \rceil$.

2. Construct a table with entries (j, a^i) for $0 \leq j < m$. Sort this table by the second component.
3. Compute a^{-m} and set $\gamma \leftarrow \beta$.
4. For i from 0 to m-1 do the following

   4.1 Check if γ is the second component of some entry in the table,

   4.2 If $\gamma = a^j$ then return x=im+j

   4.3 Set $\leftarrow \gamma . a^{-m}$.

7.4 Implementation of the Attack Algorithm:

Here is a list of the implemented attack algorithms:

1. Baby Giant with n = 2.3^{α}
2. Baby Giant with n = 3^{α}.
3. Baby Giant with n = 4.3^{α}.
4. Baby Giant with n = $2.p^{\alpha}$+1
5. Baby Giant with n = 3.p
6. Baby Giant with n = 3.p. 2^{α}
7. Irreducible Polynomial Baby Giant.

Experimental Results:

In order to attack any protocol that uses ElGamal public key encryption scheme we have to solve the discrete logarithm problem. We enhanced the Baby Step Giant Step algorithm to work with the modified algorithms.

To test the security of the algorithms, we implemented attack schemes and applied them on the modified algorithms.

After running these attack algorithms, we observed the following:

1. All the attack programs are reliable so that they can hack an encrypted message by finding the private key.

2. In all the cases, the time needed to attack the modified cryptosystem is approximately the same.

3. The most difficult to attack is the Irreducible Polynomial case. This is due to the fact that mathematically it is complex and needs considerable computing time to find the modulus of a given polynomial with respect to a certain irreducible polynomial.

4. We were not able to run the programs on large values of p and large powers since it would take a considerable time to generate some of the elements of the second group of units in each case.

   The following table shows the results of running the programs 100 times in each case:

Baby Step Giant Step Attack Evaluation	
Algorithm	***Time needed in atto sec***
n=3^a	*3.04119)*
n=2.3^a	*3.07689*
n=2p^a+1	*4.34042*
n=4.3^a	*3.20547*
n=3.p	*5.0357*
n=3.4.p	*2.96404)*
Polynomial Case	*11.3508*

Table 7.2: Evaluation results of Baby Giant algorithm

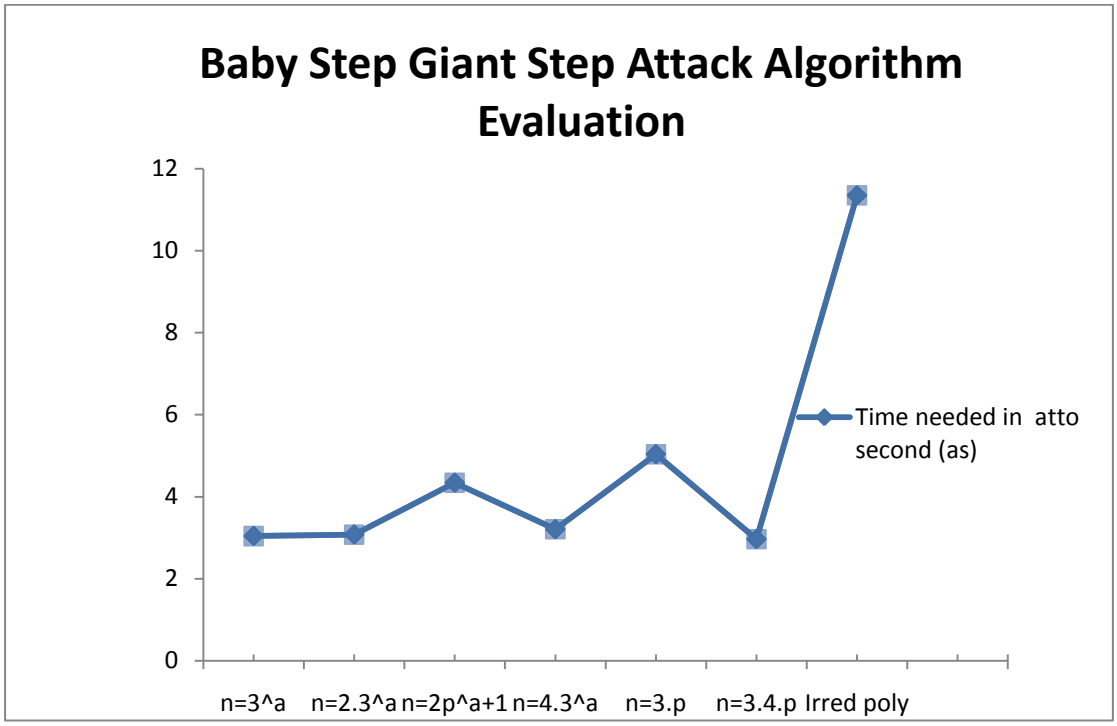

Chart 7.2: Baby Giant algorithm evaluation

Chapter Eight

Conclusion and Further Work

In this chapter, a brief summary of what was done throughout this thesis is provided. Some suggestions for further work are also pointed out to in the other section of this chapter.

We started with a simple introduction to cryptography and its keywords. Then, we introduced some mathematical background that clarifies some important definitions for keywords used throughout the chapters of the thesis.

Then, in the literature review chapter, we concentrated on the most practical and used public key cryptosystems nowadays, ElGamal Cryptosystem. We described the classical case that was previously used and the extensions performed on this cryptosystems.

Next, we introduced a new modification considering two important mathematical concepts:

"The Second Group of Units of Z_n and the Second group of Units of

$Z_2[x]/< h(x) >$ where h(x) is an irreducible polynomial".

In these two chapters algorithms for Key Generation, Encryption and Decryption were illustrated. Examples to clarify these modifications were also provided. And proofs that the new modifications do really recover the plaintext from the ciphertext were included.

Testing and evaluation of the modified cryptosystems to prove their efficiency and security was studied in the next chapter. Computer programs are created using Mathematica 7.0 as a programming language. And results obtained were also included in that chapter. These programs were used to prove the reliability of the new cryptosystems.

We did then move to attacking the Modified cryptosystem using Baby-Step-Giant-Step algorithm. Results obtained after running this attack algorithm were presented.

8.1 Further Work:

For future work, we would investigate the case of n= 8.3^{α} and 16.3 where a different algorithms would be used (since 4 and (2.3^{α}) are not relatively prime, same for 4 and (4.3^{α})).

We can also work on the case where h(x)= x^2 or on the case where h(x) is a product of irreducible polynomials whose degrees are pairwise relatively prime.

We could also work more on attacking the Modified ElGamal Cryptosystem using the Pohling Hellman attack algorithm.

Finally, the RSA cryptosystem could be also extended in the setting of The second group of Units of Z_n and $Z_2[x]/< h(x) >$.

Chapter 9

Bibliography and References

[1] W. Diffie and M. E. Hellman, "New Directions in Cryptography", IEEE Trans. on Info. Theory, IT-22, pp. 472-492, 1978.

[2] http://www.cccure.org/Documents/Cryptography/cisspallinone.pdf

[3] J. A. Gallian, *Contemporary Abstract Algebra*, 4th ed. Boston: Houghton Mifflin Company, 1998.

[4] M. Chatila, "New Public Key Encryption Scheme In Principle Ideal Domain", Master's Thesis, Dept. Of Science, Beirut Arab Univ., Beirut, Lebanon, 1998.

[5] A. R. Kenneth, *Elementary Number Theory and its applications*. New Jersey: AT & T Bell Laboratories in Murray Hill, 1975.

[6] David Joyner and Richard kreminski and Joann Turisco, *Applied Abstract Algebra*. USA: Johns Hopkins University press, 2004

[7] Thomas W. Hungerford, *Graduate Texts in Mathematics*. New York: Springer-Verlag New York Inc. 1974.

[8] Pierre Antoine Grillet, *Graduate Texts in Mathematics*, 2nd ed. New York: Springer Science + Business Media, 2007.

[9] Paul B. Garrett, *Abstract Algebra*. USA: Taylor and Francis Group, 2008.

[10] http://www.math.cornell.edu/~mec/2008-2009/Victor/part6.htm

[11] A. J. Menezes and Van Oorshot and P. C. S. A. Vanstone, *Handbook of Applied Cryptography*. Massachusetts: CRC press, 1997.

[12] T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms", in *Advances in Cryptology Proc. of CRYPTO84* (LNCS 196), 1985, pp. 10-18.

[13] J. L. Smith and J. A. Gallian, "Factoring Finite Factor Rings", Mathematics Magazine 58, pp. 93-95, 1985.

[14] A. N. El-Kassar and H. Chihadi and D. Zentout, "Quotient rings of Polynomials over Finite Fields with Cyclic Group of Units", in *Proc. International Conf. on research Trends in Science and Technology*, 2002, pp. 257-266.

[15] A. N. El-Kassar and R. Haraty, "ElGamal Public Key Cryptosystem Using Reducible Polynomials Over Finite Field", in *Proc. ISCA 13th International Conf. on Intelligent and Adaptive Systems and Software Engineering*, ISCA 2004, Nice, France, 2004, pp. 189-194.

[16] A. N. El-Kassar and R. Haraty, "ElGamal Public Key Cryptosystem in Multiplicative Groups of Quotient Rings of Polynomials over Finite Field", Computer Science and Information Systems journal, vol. 2, ISSN: 1820-0214, June 2005.

[17] H. Otrok, "Security Testing and Evaluation of Cryptographic Algorithms", Master's Thesis, Dept. of Mathematics and Computer Science, Lebanese American Univ., Beirut, Lebanon, 2003.

[18] A. N. El-Kassar and H. Chihadi, "Generalized Group of Units", Math. Balkanica (N.S.) 20, no. 3-4, pp. 275–286, 2006.

Chapter 10
Appendices

Appendix I : U^2 (Zn) is cyclic while U (Zn) is not

Appendix I – a : n = 3. p .2^a

```
GeneratorOfU[p_] :=
 Module[{ϕ, k, j, ok, t},
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
  l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n1 = RandomInteger[{2, p – 1}];
   If[GCD[n1, p] == 1, k = n1, Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]

GeneratorOfU2[p_] :=
 Module[{phi, l, f, l1, k, c},
  r = GeneratorOfU[p];
  c = 0;
  p1 = {};
  phi = EulerPhi[p];
  ϕ = EulerPhi[EulerPhi[p]];
  l = FactorInteger[ϕ];
```

```
 f = Length[l];
 For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
 l1 = Length[p1];
 For[k = 1, k ≤ ϕ, k++, Label[b];
  n1 = RandomInteger[{1, phi - 1}];
  If[GCD[n1, p - 1] == 1, s = n1, Goto[b]];
  For[j = 1, j ≤ Length[l1], j++,
   If[PowerMod[r, s^(ϕ/p1[[j]]), p] == r, c = 1]];
  If[c == 0, t = ChineseRemainder[{2, PowerMod[r, s, p], 3}, {3, p, 4}];
   Break[]]; c = 0]; {s, t, r}]

CreateKeyPairsElGamal[p_] :=
Module[{s, f},
  n = 3 * p * 4;
  Print["Picking n= ", n];
  b = GeneratorOfU2[p];
  θ = b[[2]];
  s = b[[1]];
  θ1 = b[[3]];
  a = RandomInteger[{1, EulerPhi[EulerPhi[p]] - 1}];
  Print["Picking a = ", a];
  f = PowerMod[s, a, EulerPhi[p]];
  publicKey = {p, θ1, s, f};
  privateKey = a;
  Print["Public Key is ", {p, θ1, s, f}];
  Print["Private Key is = ", privateKey]; {publicKey, privateKey}]

MessageToIntegersElGamal[m_String] := Module[ {l},
 l = {};
  l = ToCharacterCode[m]]
```

```
IntegersToMessageElGamal[integers_List] :=
 Module[{m},
  m = FromCharacterCode[integers]]

EncryptElGamal[message_, publicKey_List] :=
 Module[{s, f},
  p = publicKey[[1]];
  θ1 = publicKey[[2]];
  s = publicKey[[3]];
  f = publicKey[[4]];
  messageIntegers = MessageToIntegersElGamal[message];
  Print[messageIntegers];
  Map[EncryptIntegerElGamal[#, p, s, f, θ1] &, messageIntegers]]


EncryptIntegerElGamal[messageIntegers_, p_, s_, f_, θ1_] :=
 Module[{k},
  k = RandomInteger[{1, EulerPhi[EulerPhi[p]] - 1}];
  c = PowerMod[s, k, EulerPhi[p]];
  γ = PowerMod[θ1, c, p];
  c1 = PowerMod[f, k, EulerPhi[p]];
  δ = PowerMod[messageIntegers, c1, p];
  cipherlist = {c, δ}]


DecryptElGamal[cipherIntegers_List, privateKey_] :=
 Module[{a},
  a = privateKey;
  messageIntegers = DecryptIntegerElGamal[cipherIntegers, a, θ1];
  Print[messageIntegers];
  IntegersToMessageElGamal[messageIntegers]]


DecryptIntegerElGamal[cipherInteger_List, a_, θ1_] :=
 Module[{w, e, x, y},
  e = EulerPhi[EulerPhi[p]];
  b = e - a;
  mult[{x_, y_}] := PowerMod[y, PowerMod[x, b, EulerPhi[p]], p];
  w = Map[mult, cipherInteger]; w]
```

```
{t, {PuKey, PriKey}} = Timing[CreateKeyPairsElGamal[10 570 841]];
Print[t];

{t, cipherText} = Timing[EncryptElGamal["  d", PuKey]]

Timing[DecryptElGamal[cipherText, PriKey]]
```

Appendix I – b : n = 3. p

```
GeneratorOfU[p_] :=
 Module[{ϕ, k, j, ok, t},
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n1 = RandomInteger[{2, p – 1}];
   If[GCD[n1, p] == 1, k = n1, Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]
GeneratorOfU2[p_] :=
 Module[{phi, l, f, l1, k, c},
  r = GeneratorOfU[p];
  c = 0;
  p1 = {};
  phi = EulerPhi[p];
```

```
 ϕ = EulerPhi[EulerPhi[p]];
 l = FactorInteger[ϕ];
 f = Length[l];
 For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
 l1 = Length[p1];
 For[k = 1, k ≤ ϕ, k++, Label[b];
  n1 = RandomInteger[{1, phi - 1}];
  If[GCD[n1, p - 1] == 1, s = n1, Goto[b]];
  For[j = 1, j ≤ Length[l1], j++,
   If[PowerMod[r, s^(ϕ/p1[[j]]), p] == r, c = 1]];
  If[c == 0, t = ChineseRemainder[{2, PowerMod[r, s, p]}, {3, p}]; Break[]];
  c = 0]; {s, t, r}]

CreateKeyPairsElGamal[p_] :=
Module[{s, f},
  n = 3 * p;
  Print["Picking n= ", n];
  b = GeneratorOfU2[p];
  θ = b[[2]];
  s = b[[1]];
  θ1 = b[[3]];
  a = RandomInteger[{2, EulerPhi[EulerPhi[p]] - 1}];
  Print["Picking a = ", a];
  f = PowerMod[s, a, EulerPhi[p]];
  publicKey = {p, θ1, s, f};
  privateKey = a;
  Print["Public Key is ", {p, θ1, s, f}];
  Print["Private Key is = ", privateKey]; {publicKey, privateKey}]

MessageToIntegersElGamal[m_String] := Module[ {l},
 l = {};
  l = ToCharacterCode[m]]
```

```
EncryptElGamal[message_, publicKey_List] :=
 Module[{s, f},
  p = publicKey[[1]];
  θ1 = publicKey[[2]];
  s = publicKey[[3]];
  f = publicKey[[4]];  messageIntegers = MessageToIntegersElGamal[message];
  Print[messageIntegers];
  Map[EncryptIntegerElGamal[#, p, s, f, θ1] &, messageIntegers]]

EncryptIntegerElGamal[messageIntegers_, p_, s_, f_, θ1_] :=
 Module[{k},
  k = RandomInteger[{2, EulerPhi[EulerPhi[p]] - 1}];
  c = PowerMod[s, k, EulerPhi[p]];
  γ = PowerMod[θ1, c, p];
  c1 = PowerMod[f, k, EulerPhi[p]];
  δ = PowerMod[messageIntegers, c1, p];
  cipherlist = {c, δ}]


DecryptElGamal[cipherIntegers_List, privateKey_] :=
 Module[{a},
  a = privateKey;
  messageIntegers = DecryptIntegerElGamal[cipherIntegers, a, θ1];
  Print[messageIntegers];
  IntegersToMessageElGamal[messageIntegers]]

DecryptIntegerElGamal[cipherInteger_List, a_, θ1_] :=
 Module[{w, e, x, y},
  e = EulerPhi[EulerPhi[p]];
  b = e - a;
  mult[{x_, y_}] := PowerMod[y, PowerMod[x, b, EulerPhi[p]], p];
  w = Map[mult, cipherInteger]; w]

{t, {PuKey, PriKey}} = Timing[CreateKeyPairsElGamal[2 760 727 302 517]];
Print[t];

{t, cipherText} = Timing[EncryptElGamal["  d", PuKey]]

Timing[DecryptElGamal[cipherText, PriKey]]
```

Appendix I – c : n = 4.3^a

```
GeneratorOfU[α_] :=
 Module[{ϕ, k, j, ok, t},
  p = 3^α;
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n = RandomInteger[{2, p – 1}];
   If[GCD[n, p] == 1, k = n, Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]
GeneratorOfU2[α_] :=
 Module[{p, phi, l, f, l1, k, c},
  p = 3^α;
  r = GeneratorOfU[α];
  Print["r=", r];
  c = 0;
  p1 = {};
  phi = EulerPhi[p];
  ϕ = EulerPhi[EulerPhi[p]];
  l = FactorInteger[ϕ];
  f = Length[l];
```

```
  For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
  l1 = Length[p1];
  For[k = 1, k ≤ ϕ, k++, Label[b];
   n = RandomInteger[{1, phi - 1}];
   If[GCD[n, p - 1] == 1, s = n, Goto[b]];
   For[j = 1, j ≤ Length[l1], j++,
    If[PowerMod[r, s^(ϕ/p1[[j]]), p] == r, c = 1]];
   If[c == 0, t = ChineseRemainder[{3, PowerMod[r, s, p]}, {4, p}]; Break[]];
   c = 0]; {s, t, r}]
CreateKeyPairsElGamal[α_] :=
Module[{w, s, f},
  p = 3^α;
  n = 4 * p;
  Print["Picking n= ", n];
  b = GeneratorOfU2[α];
  θ = b[[2]];
  s = b[[1]];
  θ1 = b[[3]];
  a = RandomInteger[{1, EulerPhi[EulerPhi[p]] - 1}];
  Print["Picking a = ", a];
  f = PowerMod[s, a, EulerPhi[p]];
  publicKey = {p, θ1, s, f};
  privateKey = a;
  Print["Public Key is ", {p, θ1, s, f}];
  Print["Private Key is = ", privateKey]; {publicKey, privateKey}]

MessageToIntegersElGamal[m_String] := Module[ {l},
 l = {};
  l = ToCharacterCode[m]]
```

```
IntegersToMessageElGamal[integers_List] :=
 Module[{m},
  m = FromCharacterCode[integers]]

EncryptElGamal[message_, publicKey_List] :=
 Module[{s, p, f},
  p = publicKey[[1]];
  θ1 = publicKey[[2]];
  s = publicKey[[3]];
  f = publicKey[[4]];
  messageIntegers = MessageToIntegersElGamal[message];
  Print[messageIntegers];
  Map[EncryptIntegerElGamal[#, p, s, f, θ1] &, messageIntegers]]

EncryptIntegerElGamal[messageIntegers_, p_, s_, f_, θ1_] :=
 Module[{k},
  k = RandomInteger[{1, EulerPhi[EulerPhi[p]] - 1}];
  c = PowerMod[s, k, EulerPhi[p]];
  γ = PowerMod[θ1, c, p];
  c1 = PowerMod[f, k, EulerPhi[p]];
  δ = PowerMod[messageIntegers, c1, p];
  cipherlist = {c, δ}]

DecryptElGamal[cipherIntegers_List, privateKey_] :=
 Module[{a},
  a = privateKey;
  messageIntegers = DecryptIntegerElGamal[cipherIntegers, a, θ1];
  Print[messageIntegers];
  IntegersToMessageElGamal[messageIntegers]]
```

```
DecryptIntegerElGamal[cipherInteger_List, a_, θ1_] :=
 Module[{w, e, x, y},
  e = EulerPhi[EulerPhi[p]];
  b = e - a;
  mult[{x_, y_}] := PowerMod[y, PowerMod[x, b, EulerPhi[p]], p];
  w = Map[mult, cipherInteger]; w]

{t, {PuKey, PriKey}} = Timing[CreateKeyPairsElGamal[7000]];
Print[t];

{t, cipherText} = Timing[EncryptElGamal["  d", PuKey]]

Timing[DecryptElGamal[cipherText, PriKey]]
```

Appendix II : Polynomial Case

Appendix II - a : To Find Irreducible polynomials of order n in Z2

```
p = 2; h = {}; hirr = {};
For[b = 0, b < p, b++, For[c = 0, c < p, c++,
    For[d = 0, d < p, d++, For[e = 0, e < p, e++,
      For[f = 0, f < p, f++, For[g = 0, g < p, g++, For[n = 0, n < p, n++,
          h = Append[h, 1 + b x + c x^2 + d x^3 + e x^4 + f x^5 + g x^6 + n x^7]]]]]]]];
L = Length[h];
For[i = 1, i ≤ L, i++, m1 = 0;
  For[j = 0, j < p, j++, If[Mod[h[[i]] /. x -> j, p] == 0, m1 = 1]];
  If[m1 == 0, hirr = Append[hirr, h[[i]]]]];
Print["The irreducible polynomials are: ", hirr];
```

Appendix II – b : n = 7 and p = 2

```
Generate[p_, L_, H_] := Module[{A, i, j, k},
  A = {};
  For[i = 0, i ≤ H – 1, i++,
   For[j = 0, j ≤ H – 1, j++, For[k = 0, k ≤ H – 1, k++, For[l = 0, l ≤ H – 1, l++,
       For[m = 0, m ≤ H – 1, m++, For[n = 0, n ≤ H – 1, n++, For[o = 0, o ≤ H – 1, o++,
          A = Append[A, i + j x + k x^2 + l x^3 + m x^4 + n x^5 + o x^6]]]]]]]; A]
GeneratorAlpha[n1_] := Module[{v, v1, α, l, q, flag, k, n, Low, ig},
  n = 2^n1 – 1;
  v = FactorInteger[n];
  α = {};
  α1 = {};
  v1 = {};
  z = {};
  u2 = {};
  k = 0;
  ig = 0;
  Label[b];
  Low = ig;
  ig = Low + 2;
  α = Generate[2, Low, ig];
  For[i = 2, i ≤ Length[α],
   flag = 1; l = 0;
   While[flag == 1 && l <= Length[v] – 1,
    l = l + 1;
    q = v[[l]][[1]];
    If[PolynomialRemainder[α[[i]]^(n / q), 1 + x^2 + x^7, x, Modulus → 2] == 1,
     flag = 0]]; If[flag == 1, k = α[[i]]; Break[]]; i++];
  If[flag == 0, Goto[b]]; k]
```

```
GeneratorOfU2[n1_] :=
 Module[{phi, l, f, l1, k, c},
  n = 2^n1 - 1;
  r = GeneratorAlpha[n1];
  c = 0;
  p1 = {};
  l = FactorInteger[n];
  f = Length[l];
  For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
  l1 = Length[p1];
  For[k = 1, k ≤ n, k++, Label[b];
   n2 = RandomInteger[{1, n - 1}];
   If[GCD[n2, n] == 1, s = n, Goto[b]];
   For[j = 1, j ≤ Length[l1], j++,
    If[PowerMod[r, s^(n/p1[[j]]), 1 + x^2 + x^7] == r, c = 1]];
   If[c == 0, t = PolynomialMod[r^s, {1 + x^2 + x^7, 2}]; Break[]]; c = 0];
  {s, t, r}]

CreateKeyPairsElGamal[n1_] :=
 Module[{l, s, f, publicKey, privateKey},
  n = 2^n1 - 1;
  b = GeneratorOfU2[n1];
  Print["b = ", b];
  s = b[[1]];
  θ1 = b[[3]];
  a = RandomInteger[{1, EulerPhi[n] - 1}];
  Print["Picking a = ", a];
```

```
 f = PowerMod[s, a, n];
 publicKey = {n1, θ1, s, f};
 privateKey = a;
 Print["Public Key is ", publicKey];
 Print["Private Key is ", privateKey]; {publicKey, privateKey}]


EncryptElGamal[message_, publicKey_] :=
 Module[{s, f},
  n1 = publicKey[[1]];
  θ1 = publicKey[[2]];
  s = publicKey[[3]];
  f = publicKey[[4]];
  Map[EncryptIntegerElGamal[#, n1, s, f, θ1] &, message]]

EncryptIntegerElGamal[message_, n1_, s_, f_, θ1_] :=
 Module[{k},
  n = 2^n1 - 1;
  k = RandomInteger[{1, EulerPhi[n] -1}];
  c = PowerMod[s, k, n];
  γ = PolynomialMod[θ1^c, {1 + x^2 + x^7, 2}];
  c1 = PowerMod[f, k, n]; Print["c1 = ", c1];
  δ = PolynomialMod[message^c1, {1 + x^2 + x^7 , 2}]; Print["δ = ", δ];
  ciperlist = {c, δ}]


DecryptElGamal[ciperIntegersList_List, privateKey_] :=
 Module[{a, s, f, messageIntegers},
  a = privateKey;
  messageIntegers = DecryptIntegerElGamal[ciperIntegersList, a, θ1];
  messageIntegers]

DecryptIntegerElGamal[ciperInteger_List, a_, θ1_] :=
 Module[{w, z, y},
  e = EulerPhi[n];
  b = e - a;
  mult[{z_, y_}] :=
   PolynomialRemainder[(y)^PowerMod[z, b, n], 1 + x^2 + x^7, x, Modulus → 2];
  w = Map[mult, ciperInteger]; w]
```

```
{t, {PuKey, PriKey}} = Timing[CreateKeyPairsElGamal[7]];
Print[t];

{t, ciperText} = Timing[EncryptElGamal[{x, x + x^2}, PuKey]]

Timing[DecryptElGamal[ciperText, PriKey]]
```

Appendix III : Both are Cyclic

Appendix III – a : n = 2.3^a

```
GeneratorOfU[α_] :=
 Module[{ϕ, k, j, ok, t},
  p = 2 * 3^α;
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n = RandomInteger[{2, p - 1}];
   If[GCD[n, p] == 1, k = n, Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]
GeneratorOfU2[α_] :=
 Module[{p, phi, l, f, l1, k, c},
  p = 2 * 3^α;
  r = GeneratorOfU[α];
  c = 0;
  p1 = {};
  phi = EulerPhi[p];
  ϕ = EulerPhi[EulerPhi[p]];
```

```
 l = FactorInteger[ϕ];
 f = Length[l];
 For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
 l1 = Length[p1];
 For[k = 1, k ≤ ϕ, k++, Label[b];
  n = RandomInteger[{1, phi - 1}];
  If[GCD[n, p - 1] == 1, s = n, Goto[b]];
  For[j = 1, j ≤ Length[l1], j++,
   If[PowerMod[r, s^(ϕ/p1[[j]]), p] == r, c = 1]];
  If[c == 0, t = PowerMod[r, s, p]; Break[]]; c = 0]; {s, t, r}]

CreateKeyPairsElGamal[α_] :=
Module[{w, s, f},
  p = 2 * 3^α;
  Print["Picking n = ", p];
  b = GeneratorOfU2[α];
  θ = b[[2]];
  s = b[[1]];
  θ1 = b[[3]];
  a = RandomInteger[{1, EulerPhi[EulerPhi[p]] - 1}];
  Print["Picking a = ", a];
  f = PowerMod[s, a, EulerPhi[p]];
  publicKey = {p, θ1, s, f};
  privateKey = a;
  Print["Public Key is ", publicKey];
  Print["Private Key is ", privateKey]; {publicKey, privateKey}]

MessageToIntegersElGamal[m_String] := Module[ {l},
 l = {};
  l = ToCharacterCode[m]]
```

```
EncryptElGamal[message_, publicKey_] :=
 Module[{s, p, f},
  p = publicKey[[1]];
  θ1 = publicKey[[2]];
  s = publicKey[[3]];
  f = publicKey[[4]];
  messageIntegers = MessageToIntegersElGamal[message];
  Print[messageIntegers];
  Map[EncryptIntegerElGamal[#, p, s, f, θ1] &, messageIntegers]]


EncryptIntegerElGamal[messageIntegers_, p_, s_, f_, θ1_] :=
 Module[{k},
  k = RandomInteger[{1, EulerPhi[EulerPhi[p]] - 1}];
  c = PowerMod[s, k, EulerPhi[p]];
  γ = PowerMod[θ1, c, p];
  c1 = PowerMod[f, k, EulerPhi[p]];
  δ = PowerMod[messageIntegers, c1, p];
  cipherlist = {c, δ}]

DecryptElGamal[cipherIntegersList_List, privateKey_] :=
 Module[{a},
  a = privateKey;
  messageIntegers = DecryptIntegerElGamal[cipherIntegersList, a, θ1];
  Print[messageIntegers];
  IntegersToMessageElGamal[messageIntegers]]

DecryptIntegerElGamal[cipherInteger_List, a_, θ1_] :=
 Module[{w, e, x, y},
  e = EulerPhi[EulerPhi[p]];
  b = e - a;
  mult[{x_, y_}] := PowerMod[y, PowerMod[x, b, EulerPhi[p]], p];
  w = Map[mult, cipherInteger]; w]

{t, {PuKey, PriKey}} = Timing[CreateKeyPairsElGamal[10^001]];
Print[t];
```

```
{t, cipherText} = Timing[EncryptElGamal["  d", PuKey]]

Timing[DecryptElGamal[cipherText, PriKey]]
```

Appendix III – b : n = 2 p^a + 1

```
Compositeint[a1_, p_] :=
 Module[{n},
  n = 2 * p^a1 + 1;
  n]
GeneratorOfU[a1_, q_] :=
 Module[{ϕ, k, j, ok, t},
  p = Compositeint[a1, q];
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
  l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n = RandomInteger[{2, p – 1}];
   If[GCD[n, p] == 1, k = n, Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]
```

```
GeneratorOfU2[a1_, q_] :=
 Module[{p, phi, l, f, l1, k, c},
  p = Compositeint[a1, q];
  r = GeneratorOfU[a1, q];
  c = 0;
  p1 = {};
  phi = EulerPhi[p];
  ϕ = EulerPhi[EulerPhi[p]];
  l = FactorInteger[ϕ];
  f = Length[l];
  For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
  l1 = Length[p1];
  For[k = 1, k ≤ ϕ, k++, Label[b];
   n = RandomInteger[{1, phi - 1}];
   If[GCD[n, p - 1] == 1, s = n, Goto[b]];
   For[j = 1, j ≤ Length[l1], j++,
    If[PowerMod[r, s^(1/p1[[j]]), p] == r, c = 1]];
   If[c == 0, t = PowerMod[r, s, p]; Break[]]; c = 0]; {s, t, r}]
```

```
CreateKeyPairsElGamal[a1_, q_] :=
Module[{w, f},
  p = Compositeint[a1, q];
  Print["Picking p = ", p];
  b = GeneratorOfU2[a1, q];
  θ = b[[2]];
  s = b[[1]];
  θ1 = b[[3]];
  a = RandomInteger[{2, EulerPhi[EulerPhi[p]] - 1}];
  Print["Picking a = ", a];
  f = PowerMod[s, a, EulerPhi[p]];
  publicKey = {p, θ1, s, f};
  privateKey = a;
  Print["Public Key is ", {p, θ1, s, f}];
  Print["Private Key is  ", privateKey]; {publicKey, privateKey}]

MessageToIntegersElGamal[m_String] := Module[ {l},
 l = {};
  l = ToCharacterCode[m]]

IntegersToMessageElGamal[integers_List] :=
 Module[{m},
  m = FromCharacterCode[integers]]

EncryptElGamal[message_, publicKey_List] :=
 Module[{s, p, f},
  p = publicKey[[1]];
  θ1 = publicKey[[2]];
  s = publicKey[[3]];
  f = publicKey[[4]];
  messageIntegers = MessageToIntegersElGamal[message];
  Print[messageIntegers];
  Map[EncryptIntegerElGamal[#, p, s, f, θ1] &, messageIntegers]]
```

```
EncryptIntegerElGamal[messageIntegers_, p_, s_, f_, θ1_] :=
 Module[{k},
  k = RandomInteger[{1, EulerPhi[EulerPhi[p]] - 1}];
  c = PowerMod[s, k, EulerPhi[p]];
  γ = PowerMod[θ1, c, p];
  c1 = PowerMod[f, k, EulerPhi[p]];
  δ = PowerMod[messageIntegers, c1, p];
  cipherlist = {c, δ}]

DecryptElGamal[cipherIntegersList_List, privateKey_] :=
 Module[{a},
  a = privateKey;
  messageIntegers = DecryptIntegerElGamal[cipherIntegersList, a, θ1];
  Print[messageIntegers];
  IntegersToMessageElGamal[messageIntegers]]

DecryptIntegerElGamal[cipherInteger_List, a_, θ1_] :=
 Module[{w, e, x, y},
  e = EulerPhi[EulerPhi[p]];
  b = e - a;
  mult[{x_, y_}] := PowerMod[y, PowerMod[x, b, EulerPhi[p]], p];
  w = Map[mult, cipherInteger]; w]

{t, {PuKey, PriKey}} = Timing[CreateKeyPairsElGamal[1, 233]];
Print[t];

{t, cipherText} = Timing[EncryptElGamal["■", PuKey]]

Timing[DecryptElGamal[cipherText, PriKey]]
```

Appendix III – c : n = 3^a

```
GeneratorOfU[α_] :=
 Module[{ϕ, k, j, ok, t},
  p = 3^α;
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n = RandomInteger[{2, p – 1}];
   If[GCD[n, p] == 1, k = n, Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]
GeneratorOfU2[α_] :=
 Module[{p, phi, l, f, l1, k, c},
  p = 3^α;
  r = GeneratorOfU[α];
  c = 0;
  p1 = {};
  phi = EulerPhi[p];
  ϕ = EulerPhi[EulerPhi[p]];
  l = FactorInteger[ϕ];
  f = Length[l];
  For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
  l1 = Length[p1];
```

```
  For[k = 1, k ≤ ϕ, k++, Label[b];
   n = RandomInteger[{1, phi - 1}];
   If[GCD[n, p - 1] == 1, s = n, Goto[b]];
   For[j = 1, j ≤ Length[l1], j++,
    If[PowerMod[r, s^(ϕ/p1[[j]]), p] == r, c = 1]];
   If[c == 0, t = PowerMod[r, s, p]; Break[]; c = 0]; {s, t, r}]
CreateKeyPairsElGamal[α_] :=
Module[{w, s, f},
  p = 3^α;
  Print["Picking p = ", p];
  b = GeneratorOfU2[α];
  θ = b[[2]];
  s = b[[1]];
  θ1 = b[[3]];
  a = RandomInteger[{1, EulerPhi[EulerPhi[p]] - 1}];
  Print["Picking a = ", a];
  f = PowerMod[s, a, EulerPhi[p]];
  publicKey = {p, θ1, s, f};
  privateKey = a;
  Print["Public Key is ", {p, θ1, s, f}];
  Print["Private Key is  ", privateKey]; {publicKey, privateKey}]
MessageToIntegersElGamal[m_String] := Module[ {l},
 l = {};
  l = ToCharacterCode[m]]
IntegersToMessageElGamal[integers_List] :=
 Module[{m},
  m = FromCharacterCode[integers]]
```

```
EncryptElGamal[message_, publicKey_List] :=
 Module[{s, p, f},
  p = publicKey[[1]];
  θ1 = publicKey[[2]];
  s = publicKey[[3]];
  f = publicKey[[4]];
  messageIntegers = MessageToIntegersElGamal[message];
  Print[messageIntegers];
  Map[EncryptIntegerElGamal[#, p, s, f, θ1] &,
   messageIntegers]

EncryptIntegerElGamal[messageIntegers_, p_, s_, f_, θ1_] :=
 Module[{k},
  k = RandomInteger[{1, EulerPhi[EulerPhi[p]] - 1}];
  c = PowerMod[s, k, EulerPhi[p]];
  γ = PowerMod[θ1, c, p];
  c1 = PowerMod[f, k, EulerPhi[p]];
  δ = PowerMod[messageIntegers, c1, p];
  cipherlist = {c, δ}]

DecryptElGamal[cipherIntegersList_List, privateKey_] :=
 Module[{a},
  a = privateKey;
  messageIntegers = DecryptIntegerElGamal[cipherIntegersList, a, θ1];
  Print[messageIntegers];
  IntegersToMessageElGamal[messageIntegers]]


DecryptIntegerElGamal[cipherInteger_List, a_, θ1_] :=
 Module[{w, e, x, y},
  e = EulerPhi[EulerPhi[p]];
  b = e - a;
  mult[{x_, y_}] := PowerMod[y, PowerMod[x, b, EulerPhi[p]], p];
  w = Map[mult, cipherInteger]; w]
```

```
{t, {PuKey, PriKey}} = Timing[CreateKeyPairsElGamal[5000]];
Print[t];
{t, cipherText} = Timing[EncryptElGamal["  d", PuKey]]
Timing[DecryptElGamal[cipherText, PriKey]]
```

Appendix IV : Baby Giant Attack Scheme

Appendix IV – a : n = 3. p .2^a

```
α = 9671;
β = 215;
q = 1009;
a = 2;
p = 3 * q * 2^a;
n = EulerPhi[EulerPhi[q]];
m = Ceiling[Sqrt[n]];
t1 = Array[t11, {2, m}];
MatrixForm[t1];
Do[t1[[1, j]] = j – 1; t1[[2, j]] = PowerMod[α, j – 1, p], {j, 1, m}];
MatrixForm[t1];
ai = PowerMod[α, –1, p];
am = PowerMod[ai, m, p];
h = 1; i = 0;
e = 1;
l = 0;
While[e ≠ 0 && l ≤ m,
ami = PowerMod[am, i, p];
ɣ = Mod[β * ami, p];
Do[If[ɣ == t1[[2, k]], e = 0; j = t1[[1, k]];
   Print["j = ", j]; t = Timing[ɣ]; h = 2; Break[]], {k, 1, m}]; i = i + 1;]
Print["i = ", i – 1];
a = (i – 1) * m + j; Print["The Discrete Logarithm is ", a];
Print["Time is ", t];
```

Appendix IV – b : Elements of n = 3. p .2^a

```
GeneratorOfU[p_] :=
 Module[{ϕ, k, j, ok, t},
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n1 = RandomInteger[{2, p – 1}];
   If[GCD[n1, p] == 1, k = n1
    , Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]

ElmOfU2[p_] :=
 Module[{un, s},
  n = 3 * p * 4;
  m = {};
  un = {};
  s = {};
  r = GeneratorOfU[p]; Print[r];
  For[i = 1, i ≤ n, i++, If[GCD[i, n] == 1,  un = Append[un, i]]];
```

```
 For[i = 1, i ≤ Length[un], i++,
   s = Append[s, {Mod[un[[i]], 3], Mod[un[[i]], p], Mod[un[[i]], 4]}]];
 For[k = 2, k ≤ 200, k++,
  For[j = 0, j < EulerPhi[p], j++,
   If[Mod[r^j, p] == s[[k]][[2]],
    If[GCD[j, p - 1] == 1,
     If[s[[k]][[1]] == 2, If[s[[k]][[3]] == 3, m = Append[m, un[[k]]]]]]]]; m]

GeneratorOfU2[p_] :=
 Module[{phi, l, f, l1, k, c},
  r = 563;
  c = 0;
  p1 = {};
  phi = EulerPhi[p];
  ϕ = EulerPhi[EulerPhi[p]];
  l = FactorInteger[ϕ];
  f = Length[l];
  For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
  l1 = Length[p1];
  For[k = 1, k ≤ ϕ, k++, Label[b];
   n1 = RandomInteger[{1, phi - 1}];
   If[GCD[n1, p - 1] == 1, s = n1, Goto[b]];
   For[j = 1, j ≤ Length[l1], j++,
    If[PowerMod[r, s^(ϕ/p1[[j]]), p] == r, c = 1]];
   If[c == 0, t = ChineseRemainder[{2, PowerMod[r, s, p], 3}, {3, p, 4}];
    Break[]]; c = 0]; t]
```

Appendix IV – c : n = 3. p

```
α = 1550;
β = 107;
q = 1009;
p = 3 * q;
n = EulerPhi[EulerPhi[q]];
m = Ceiling[Sqrt[n]];
t1 = Array[t11, {2, m}];
MatrixForm[t1];
Do[t1[[1, j]] = j – 1; t1[[2, j]] = PowerMod[α, j – 1, p], {j, 1, m}];
MatrixForm[t1];
ai = PowerMod[α, –1, p];
am = PowerMod[ai, m, p];
h = 1; i = 0;
e = 1;
l = 0;
While[e ≠ 0 && l ≤ m,
ami = PowerMod[am, i, p];
γ = Mod[β * ami, p];
Do[If[γ == t1[[2, k]], e = 0; j = t1[[1, k]];
   Print["j = ", j]; t = Timing[γ]; h = 2; Break[]], {k, 1, m}]; i = i + 1;]
Print["i = ", i – 1];
a = (i – 1) * m + j; Print["The Discrete Logarithm is ", a];
Print["Time is ", t];
```

Appendix IV – d : Elements of n = 3. p

```
GeneratorOfU[p_] :=
 Module[{ϕ, k, j, ok, t},
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n1 = RandomInteger[{2, p – 1}];
   If[GCD[n1, p] == 1, k = n1
    , Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]
ElmOfU2[p_] :=
 Module[{un, s},
  n = 3 * p;
  m = {};
  un = {};
  s = {};
  r = GeneratorOfU[p]; Print[r];
  For[i = 1, i ≤ 200, i++, If[GCD[i, n] == 1,  un = Append[un, i]]];
  For[i = 1, i ≤ Length[un], i++,
   s = Append[s, {Mod[un[[i]], 3], Mod[un[[i]], p]}]];
  For[k = 2, k ≤ 50, k++,
   For[j = 0, j < EulerPhi[p], j++,
    If[Mod[r^j, p] == s[[k]][[2]],
     If[GCD[j, p – 1] == 1, If[s[[k]][[1]] == 2, m = Append[m, un[[k]]]]]]]; m]
```

```
GeneratorOfU2[p_] :=
 Module[{phi, l, f, l1, k, c},
  r = 3512;
  n = 3 * p;
  c = 0;
  u = {};
  p1 = {};
  m = {};
  phi = EulerPhi[p];
  ϕ = EulerPhi[EulerPhi[p]];
  l = FactorInteger[ϕ];
  f = Length[l];
  For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
  l1 = Length[p1];
  For[k = 1, k ≤ ϕ, k++, Label[b];
   n1 = RandomInteger[{1, phi - 1}];
   If[GCD[n1, p - 1] == 1, s = n1, Goto[b]];
   For[j = 1, j ≤ Length[l1], j++,
    If[PowerMod[r, s^(ϕ/p1[[j]]), p] == r, c = 1]];
   If[c == 0,
    t = ChineseRemainder[{2, PowerMod[r, s, p]}, {3, p}]; Break[]]; c = 0]; t]
n1 = GeneratorOfU2[1481]; Print[n1];
```

Appendix IV – e : n = 4.3^a

```
α = 119;
β = 143;
a = 5;
q = 3^a;
p = 4 * q;
n = EulerPhi[EulerPhi[q]];
m = Ceiling[Sqrt[n]];
t1 = Array[t11, {2, m}];
MatrixForm[t1];
Do[t1[[1, j]] = j – 1; t1[[2, j]] = PowerMod[α, j – 1, p], {j, 1, m}];
MatrixForm[t1];
ai = PowerMod[α, –1, p];
am = PowerMod[ai, m, p];
h = 1; i = 0;
e = 1;
l = 0;
While[e ≠ 0 && l ≤ m,
ami = PowerMod[am, i, p];
γ = Mod[β * ami, p];
Do[If[γ == t1[[2, k]], e = 0; j = t1[[1, k]];
   Print["j = ", j]; t = Timing[γ]; h = 2; Break[]], {k, 1, m}]; i = i + 1;]
Print["i = ", i – 1];
a = (i – 1) * m + j; Print["The Discrete Logarithm is ", a];
Print["Time is ", t];
```

Appendix IV – f : Elements of n = 4.3^a

```
GeneratorOfU[a_] :=
 Module[{ϕ, k, j, ok, t},
  p = 3^a;
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n1 = RandomInteger[{2, p – 1}];
   If[GCD[n1, p] == 1, k = n1
    , Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]
ElmOfU2[a_] :=
 Module[{un, s},
  p = 3^a;
  n = 4 * p;
  m = {};
  un = {};
  s = {};
  r = GeneratorOfU[a]; Print[r];
  For[i = 1, i ≤ n, i++, If[GCD[i, n] == 1, un = Append[un, i]]];
  For[i = 1, i ≤ 200, i++, s = Append[s, {Mod[un[[i]], 4], Mod[un[[i]], p]}]];
  For[k = 2, k ≤ 50, k++,
   For[j = 0, j < EulerPhi[p], j++,
    If[Mod[r^j, p] == s[[k]][[2]],
     If[GCD[j, p – 1] == 1, If[s[[k]][[1]] == 3, m = Append[m, un[[k]]]]]]]; m]
```

```
GeneratorOfU2[a_] :=
 Module[{phi, l, f, l1, k, c},
  r = 95;
  p = 3^a;
  n = 4 * p;
  c = 0;
  u = {};
  p1 = {};
  m = {};
  phi = EulerPhi[p];
  ϕ = EulerPhi[EulerPhi[p]];
  l = FactorInteger[ϕ];
  f = Length[l];
  For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
  l1 = Length[p1];
  For[k = 1, k ≤ ϕ, k++, Label[b];
   n1 = RandomInteger[{1, phi - 1}];
   If[GCD[n1, p - 1] == 1, s = n1, Goto[b]];
   For[j = 1, j ≤ Length[l1], j++,
    If[PowerMod[r, s^(ϕ/p1[[j]]), p] == r, c = 1]];
   If[c == 0,
    t = ChineseRemainder[{3, PowerMod[r, s, p]}, {4, p}]; Break[]]; c = 0]; t]

n1 = GeneratorOfU2[5]; Print[n1];
```

Appendix IV – g : n = 2.3^a

```
α = 1037;
β = 923;
a = 6;
q = 2 * 3^a;
n = EulerPhi[EulerPhi[q]];
m = Ceiling[Sqrt[n]];
t1 = Array[t11, {2, m}];
MatrixForm[t1];
Do[t1[[1, j]] = j – 1; t1[[2, j]] = PowerMod[α, j – 1, q], {j, 1, m}];
MatrixForm[t1];
ai = PowerMod[α, –1, q];
am = PowerMod[ai, m, q];
h = 1; i = 0;
e = 1;
l = 0;
While[e ≠ 0 && l ≤ m,
ami = PowerMod[am, i, q];
γ = Mod[β * ami, q];
Do[If[γ == t1[[2, k]], e = 0; j = t1[[1, k]];
   Print["j = ", j]; t = Timing[γ]; h = 2; Break[]], {k, 1, m}]; i = i + 1;]
Print["i = ", i – 1];
a = (i – 1) * m + j; Print["The Discrete Logarithm is ", a];
Print["Time is ", t];
```

Appendix IV – h : Elements of n = 2.3^a

```
GeneratorOfU[α_] :=
 Module[{ϕ, k, j, ok, t},
  p = 2 * 3^α;
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n = RandomInteger[{2, p – 1}];
   If[GCD[n, p] == 1, k = n, Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]

ElmOfU2[a_] :=
 Module[{un, s},
  m = {};
  un = {};
  r = GeneratorOfU[a]; Print[r];
  For[i = 1, i < 2 * 3^a, i++, If[GCD[i, 2 * 3^a] == 1, un = Append[un, i]]];
  For[k = 2, k ≤ EulerPhi[2 * 3^a], k++,
   For[j = 0, j < EulerPhi[2 * 3^a], j++,
    If[Mod[r^j, 2 * 3^a] == un[[k]],
     If[GCD[j, EulerPhi[2 * 3^a]] == 1, m = Append[m, un[[k]]]]]]]; m]
```

```
GeneratorOfU2[α_] :=
 Module[{p, phi, l, f, l1, k, c},
  p = 2 * 3^α;
  r = 299;
  c = 0;
  p1 = {};
  phi = EulerPhi[p];
  ϕ = EulerPhi[EulerPhi[p]];
  l = FactorInteger[ϕ];
  f = Length[l];
  For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
  l1 = Length[p1];
  For[k = 1, k ≤ ϕ, k++, Label[b];
   n = RandomInteger[{1, phi - 1}];
   If[GCD[n, p - 1] == 1, s = n, Goto[b]];
   For[j = 1, j ≤ Length[l1], j++,
    If[PowerMod[r, s^(ϕ/p1[[j]]), p] == r, c = 1]];
   If[c == 0, t = PowerMod[r, s, p]; Break[]; c = 0]; t]

n1 = GeneratorOfU2[6]; Print[n1];
```

Appendix IV – i : n = 2 p^a + 1

```
α = 304;
β = 408;
a = 1;
q = 233;
p = 2 * q^a + 1;
n = EulerPhi[EulerPhi[p]];
m = Ceiling[Sqrt[n]];
t1 = Array[t11, {2, m}];
MatrixForm[t1];
Do[t1[[1, j]] = j – 1; t1[[2, j]] = PowerMod[α, j – 1, p], {j, 1, m}];
MatrixForm[t1];
ai = PowerMod[α, –1, p];
am = PowerMod[ai, m, p];
h = 1; i = 0;
e = 1;
l = 0;
While[e ≠ 0 && l ≤ m,
ami = PowerMod[am, i, p];
γ = Mod[β * ami, p];
Do[If[γ == t1[[2, k]], e = 0; j = t1[[1, k]];
    Print["j = ", j]; t = Timing[γ]; h = 2; Break[]], {k, 1, m}]; i = i + 1];
Print["i = ", i – 1];
a = (i – 1) * m + j; Print["The Discrete Logarithm is ", a];
Print["Time is ", t];
```

Appendix IV – j : Elements of n = 2 p^a + 1

```
GeneratorOfU[α_, q_] :=
 Module[{ϕ, k, j, ok, t},
  p = 2 * q^α + 1;
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n = RandomInteger[{2, p – 1}];
   If[GCD[n, p] == 1, k = n, Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]

ElmOfU2[a_, q_] :=
 Module[{un, s},
  m = {};
  un = {};
  r = GeneratorOfU[a, q]; Print[r];
  For[i = 1, i < 2 * q^a + 1, i++, If[GCD[i, 2 * q^a + 1] == 1, un = Append[un, i]]];
  For[k = 2, k ≤ EulerPhi[2 * q^a + 1], k++,
   For[j = 0, j < EulerPhi[2 * q^a + 1], j++,
    If[Mod[r^j, 2 * q^a + 1] == un[[k]],
     If[GCD[j, EulerPhi[2 * q^a + 1]] == 1, m = Append[m, un[[k]]]]]]]; m]
```

```
GeneratorOfU2[α_, q_] :=
 Module[{p, phi, l, f, l1, k, c},
  p = 2 * q^α + 1;
  r = 428;
  c = 0;
  p1 = {};
  phi = EulerPhi[p];
  ϕ = EulerPhi[EulerPhi[p]];
  l = FactorInteger[ϕ];
  f = Length[l];
  For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
  l1 = Length[p1];
  For[k = 1, k ≤ ϕ, k++, Label[b];
   n = RandomInteger[{1, phi - 1}];
   If[GCD[n, p - 1] == 1, s = n, Goto[b]];
   For[j = 1, j ≤ Length[l1], j++,
    If[PowerMod[r, s^(ϕ/p1[[j]]), p] == r, c = 1]];
   If[c == 0, t = PowerMod[r, s, p]; Break[]]; c = 0]; t]

n1 = GeneratorOfU2[1, 233]; Print[n1];
```

Appendix IV – k : n = 3^a

```
α = 2;
β = 29;
a = 4;
q = 3^a;
n = EulerPhi[EulerPhi[q]];
m = Ceiling[Sqrt[n]];
t1 = Array[t11, {2, m}];
MatrixForm[t1];
Do[t1[[1, j]] = j – 1; t1[[2, j]] = PowerMod[α, j – 1, q], {j, 1, m}];
d = MatrixForm[t1];
ai = PowerMod[α, –1, q];
am = PowerMod[ai, m, q];
h = 1; i = 0;
e = 1;
l = 0;
While[e ≠ 0 && l ≤ m,
ami = PowerMod[am, i, q];
γ = Mod[β * ami, q];
Do[If[γ == t1[[2, k]], e = 0; j = t1[[1, k]];
    Print["j = ", j]; t = Timing[γ]; h = 2; Break[]], {k, 1, m}]; i = i + 1];
Print["i = ", i – 1];
a = (i – 1) * m + j; Print["The Discrete Logarithm is ", a];
Print["Time is ", t];
```

Appendix IV - l : Elements of n = 3^a

```
GeneratorOfU[α_] :=
 Module[{ϕ, k, j, ok, t},
  p = 3^α;
  p1 = {};
  ok = 0;
  ϕ = EulerPhi[p];
  l = FactorInteger[ϕ];
  For[i = 1, i ≤ Length[l], i++, p1 = Append[p1, l[[i]][[1]]^l[[i]][[2]]]];
l1 = Length[p1];
  For[i1 = 1, i1 ≤ ϕ, i1++,
   Label[b];
   n = RandomInteger[{2, p - 1}];
   If[GCD[n, p] == 1, k = n, Goto[b]]; For[j = 1, j ≤ l1, j++,
    If[PowerMod[k, ϕ / p1[[j]], p] == 1, ok = 1]];
   If[ok == 0, t = k; Break[]]; ok = 0]; t]

ElmOfU2[a_] :=
 Module[{un, s},
  m = {};
  un = {};
  r = GeneratorOfU[a]; Print[r];
  For[i = 1, i < 3^a, i++, If[GCD[i, 3^a] == 1, un = Append[un, i]]]; Print[un];
  For[k = 2, k ≤ EulerPhi[3^a], k++,
   For[j = 0, j < EulerPhi[3^a], j++,
    If[Mod[r^j, 3^a] == un[[k]],
     If[GCD[j, EulerPhi[3^a]] == 1, m = Append[m, un[[k]]]; Print[m]]]]]; m]
```

```
GeneratorOfU2[α_] :=
 Module[{p, phi, l, f, l1, k, c},
  p = 3^α;
  r = GeneratorOfU[α];
  c = 0;
  p1 = {};
  phi = EulerPhi[p];
  ϕ = EulerPhi[EulerPhi[p]];
  l = FactorInteger[ϕ];
  f = Length[l];
  For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
  l1 = Length[p1];
  For[k = 1, k ≤ ϕ, k++, Label[b];
   n = RandomInteger[{1, phi - 1}];
   If[GCD[n, p - 1] == 1, s = n, Goto[b]];
   For[j = 1, j ≤ Length[l1], j++,
    If[PowerMod[r, s^(ϕ/p1[[j]]), p] == r, c = 1]];
   If[c == 0, t = PowerMod[r, s, p]; Break[]]; c = 0]; t]
```

Appendix IV – m : Polynomial case

```
p = 2;
α = x^5 + x^4;
β = x^6;
g = 1 + x^2 + x^7;
n = p^7 - 1;
m = Ceiling[Sqrt[n]];
t1 = Array[t11, {2, m}];
MatrixForm[t1];
Do[t1[[1, j]] = j - 1; t1[[2, j]] = PolynomialMod[α^(j-1), {g, p}]
, {j, 1, m}];
MatrixForm[t1];
polygcd[f_, g_, p_] := (
  r1 = f; r2 = g;
  d1 = 0; s1 = 1;
  d2 = 1; s2 = 0;
  While[Exponent[r2, x] != 0,
   q = PolynomialMod[PolynomialQuotient[r1, r2, x], p];
   r = PolynomialMod[PolynomialRemainder[r1, r2, x], p];
   d = PolynomialMod[-q * d2 + d1, p];
   s = PolynomialMod[-q * s2 + s1, p];
   r1 = r2; r2 = r;
   d1 = d2; d2 = d;
   s1 = s2; s2 = s;];
  t := PolynomialMod[PolynomialGCD[r1, r2, x], p]; s / r2 )
ai = polygcd[α, g, p];
am = PolynomialMod[ai^m, {g, p}];
h = 1; i = 0;
e = 1;
l = 0;
```

```
While[e ≠ 0 && l ≤ m,
γ = PolynomialMod[β * am^i, {g, p}];
Do[If[γ == t1[[2, k]], e = 0; j = t1[[1, k]]; Print["j = ", j];
   t = Timing[γ]; h = 2; Break[]], {k, 1, m}]; i = i + 1; l++]
Print["i = ", i - 1]; a = (i - 1) * m + j; Print["The Discrete Log is ", a];
Print["Time is ", t];
```

Appendix IV - n : Elements of Polynomial Case

```
Generate[H_] := Module[{A, i, j, k},
  A = {};
  For[i = 0, i ≤ H - 1, i++,
   For[j = 0, j ≤ H - 1, j++, For[k = 0, k ≤ H - 1, k++, For[l = 0, l ≤ H - 1, l++,
       For[m = 0, m ≤ H - 1, m++, For[n = 0, n ≤ H - 1, n++, For[o = 0, o ≤ H - 1, o++,
          A = Append[A, i + j x + k x^2 + l x^3 + m x^4 + n x^5 + o x^6]]]]]]]]; A]

GeneratorAlpha[n1_] := Module[{v, v1, α, l, q, flag, k, n, Low, ig},
  n = 2^n1 - 1;
  v = FactorInteger[n];
  α = {};
  α1 = {};
  v1 = {};
  z = {};
  u2 = {};
  k = 0;
  ig = 0;
  Label[b];
  α = Generate[2];
```

```
  For[i = 2, i ≤ Length[α],
    flag = 1; l = 0;
    While[flag == 1 && l <= Length[v] - 1,
     l = l + 1;
     q = v[[l]][[1]];
     If[PolynomialRemainder[α[[i]]^(n / q), 1 + x^7 + x^2, x, Modulus → 2] == 1,
      flag = 0]]; If[flag == 1, k = α[[i]]; Break[]]; i++];
  If[flag == 0, Goto[b]]; k]

GeneratorOfU2[nm_] :=
  Module[{n, p, phi, l, f, l1, k, c},
   n = 2^nm - 1;
   r = GeneratorAlpha[nm];
   Print["r = ", r];
   c = 0;
   t = {};
   p1 = {};
   ϕ = EulerPhi[n];
   l = FactorInteger[ϕ];
   f = Length[l];
   For[j = 1, j ≤ f, j++, p1 = Append[p1, l[[j]][[1]]^l[[j]][[2]]]];
   l1 = Length[p1];
   For[k = 1, k ≤ 10, k++, Label[b];
    n2 = RandomInteger[{1, n - 1}];
    If[GCD[n2, n] == 1, s = n2, Goto[b]];
    t = Append[t, PolynomialMod[r^s, {1 + x^2 + x^7, 2}]]];
```

```
For[k = 1, k ≤ n, k++, Label[b];
  n2 = RandomInteger[{1, n - 1}];
  If[GCD[n2, n] == 1, s = n2, Goto[b]];
  For[j = 1, j ≤ l1, j++,
   If[PowerMod[r, s^(n/p1[[j]]), {1 + x^7 + x^2, 2}] == r, c = 1]];
  If[c == 0,
   t1 = PolynomialMod[r^s, {1 + x^2 + x^7, 2}]; Print[s]; Break[]]; c = 0];
{t, t1}]
```