GROUP KEY ESTABLISHMENT PROTOCOLS: PAIRING CRYPTOGRAPHY AND VERIFIABLE SECRET SHARING SCHEME

A Thesis Submitted to
the Graduate School of Engineering and Science of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Engineering

by
Rabia ASLANOĞLU

July 2013
İZMİR

We approve the thesis of **Rabia ASLANOĞLU**

Examining Committee Members:

_____

Assist. Prof. Dr. Serap ŞAHİN
Department of Computer Engineering, İzmir Institute of Technology

_____

Assist. Prof. Dr. Kaan KURTEL
Department of Software Engineering, İzmir University of Economics

_____

Assist. Prof. Dr. Selma TEKİR
Department of Computer Engineering, İzmir Institute of Technology

1 July 2013

_____

Assist. Prof. Dr. Serap ŞAHİN
Supervisor, Department of Computer Engineering,
İzmir Institute of Technology

_____

Prof. Dr. İ. Sıtkı AYTAÇ
Head of the Department of
Computer Engineering

_____

Prof. Dr. R. Tuğrul SENGER
Dean of the Graduate School of
Engineering and Sciences

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to my thesis advisor Assist. Prof. Dr. Serap Şahin for her motivation, patience, enthusiasm and immense knowledge. She has provided invaluable suggestions and encouragement from the beginning of my research. Her wisdom always will guide me in the rest of my life. It is impossible to achieve my work without her support.

I would also like to state my gratitude to my lecturer in İzmir Institute of Technology, Assist. Prof. Dr. Selma Tekir for her deepest kindness and support. It has been a pleasure to know and work with her.

Finally, I would like to dedicate my thesis to my epic hero; my mum, Hatice Nurcan Aslanoğlu. She is the main character of my life journey and beyond the all human titles that I carry, I am honored to have the title of *being of her daughter*.

ABSTRACT

KEY ESTABLISHMENT PROTOCOLS: PAIRING CRYPTOGRAPHY AND VERIFIABLE SECRET SHARING

The aim of this study is to establish a common secret key over an open network for a group of user to be used then symmetrical secure communication between them. There are two methods of GKE protocol which are key agreement and key distribution. Key agreement is a mechanism whereby the parties jointly establish a common secret. As to key distribution, it is a mechanism whereby one of the parties creates or obtains a secret value and then securely distributes it to other parties. In this study, both methods is applied and analyzed in two different GKE protocols.

Desirable properties of a GKE are *security* and *efficiency*. Security is attributed in terms of preventing attacks against passive and active adversary. Efficiency is quantified in terms of computation, communication and round complexity. When constructing a GKE, the *challenge* is to provide security and efficiency according to attributed and quantified terms. Two main cryptographic tools are selected in order to handle the defined *challenge*. One of them is bilinear pairing which is based on elliptic curve cryptography and another is verifiable secret sharing which is based on multiparty computation.

In this thesis, constructions of these two GKE protocols are studied along with their communication models, security and efficiency analysis. Also, an implementation of four-user group size is developed utilizing PBC, GMP and OpenSSL Libraries for both two protocols.

ÖZET

GRUP ANAHTAR DEĞİŞİM PROTOKOLLERİ: PAIRING KRIPTOGRAFİ VE DOĞRULANABİLİR SIR PAYLAŞIM ŞEMASI

Çalışmamızın amacı; açık bir kanal üzerinde bir grup kullanıcının, çeşitli kriptografik hesaplamalarla *ortak gizli anahtarı* oluşturmasını sağlamaktır. Daha sonra bu gruptaki kullanıcılar, oluşturdukları *ortak gizli anahtarı* simetrik kriptografi araçlarında kullanarak güvenli bir haberleşme ortamı sağlayabilirler. İki tür *grup anahtar sağlama protokol*'ü mevcuttur. Bunlar; (1) Anahtar anlaşması (key agreement), (2)Anahtar dağıtım (key distribution) protokolleridir.

Anahtar anlaşması yönteminde; protokoldeki bütün kullanıcılar ortak şekilde, çeşitli kriptografik araçlarla *gizli anahtarı* oluştururlar.

Anahtar dağıtım yönteminde ise; protokoldeki katılımcılardan birisi *ortak gizli anahtarı* belirler ve çeşitli kriptografik araçlarla, diğer kullanıcıların da bu anahtarı elde etmesini sağlar.

Grup anahtar değişim protokolleri'nin en önemli sorunu pasif ve aktif saldırılara karşı güvenliği; ve hesaplama, iletişim ve tur karmaşaları bakımından etkinliği sağlamadır. Buna çözüm olarak iki farklı kriptografik araç seçilmiştir.

Bu çalışmada, bu iki farklı kriptografik aracın kullanıldığı, iki grup anahtar sağlama protokolü örneklenmekte, genel karakteristikleri, avantaj ve dezavantajları değerlendirilmektedir. Kullanılan kriptografik araçlardan biri "bilinear pairing", diğeri ise "doğrulanabilir sır paylaşım şeması"dır. Çalışmada, her iki protokolün, tasarımları incelenmiş, hesaplama, iletişim ve tur karmaşaları bakımından etkinlik analizleri ve güvenlik analizleri yapılmıştır. Ayrıca her iki protokolün dört kullanıcılı bir grup için PBC, GMP ve OpenSSL kütüphaneleriyle geliştirilen uygulama sonuçları da bulunmaktadır.

Bu araştırma; güvenli olmayan haberleşme ortamlarında, güvenli iletişimin sağlanması gereken farklı özellik ve kısıtlardaki uygulamalarda, kapalı ve güvenli grupları oluşturabilmek için, hangi metod ve protokolün seçilmesinin daha uygun olabileceği konusunda somut kriterler ortaya koyar.

TABLE OF CONTENTS

LIST OF FIGURES viii

LIST OF TABLES ix

CHAPTER 1. INTRODUCTION 1

1.1. Challenge 1

1.2. Motivation and Goals 3

1.3. Organization of Thesis and Contributions 4

CHAPTER 2. CRYPTOGRAPHIC BACKGROUND 6

2.1. Cryptographic Preliminaries 6

2.1.1. Number Theory and Abstract Algebra 6

2.1.2. Bilinear Pairings 10

2.1.3. Security Assumptions 12

2.1.4. Homomorphic Commitments 13

2.2. Key Establishment 13

2.2.1. Key Agreement 14

2.2.2. Key Distribution by Secret Sharing 15

2.2.3. Verifiable Secret Sharing 16

2.3. Authentication 17

CHAPTER 3. LITERATURE SURVEY 19

3.1. Secure Communication 19

3.1.1. PKI-Based Cryptography 19

3.1.2. Boneh-Lynn-Shacham and ElGamal Signature Schemes 22

3.1.3. ElGamal Encryption Scheme 26

3.1.4. Boneh-Franklin ID-Based Scheme 28

3.2. Key Agreement using Bilinear Pairings 30

3.2.1. Diffie-Hellman Key Exchange 30

3.2.2. Joux's One Round Protocol for Tripartite Diffie-Hellman 32

3.3. Key Distribution using Verifiable Secret Sharing 35

3.3.1. Shamir's Secret Sharing 35
3.3.2. Feldman's Verifiable Secret Sharing 38

CHAPTER 4. GROUP KEY ESTABLISHMENT PROTOCOLS 42
4.1. Specifications and Assumptions 43
4.2. Group Key Agreement Protocol using Bilinear Pairing 44
4.2.1. Preliminaries and Communication Model of Key Agreement Protocol 44
4.2.2. Construction of Key Agreement Protocol 45
4.2.3. Security Analysis of Key Agreement Protocol 49
4.2.4. Implementation Results of Key Agreement Protocol 53
4.3. Group Key Distribution Protocol using Verifiable Secret Sharing 60
4.3.1. Preliminaries and Communication Model of Key Distribution Protocol 60
4.3.2. Construction of Key Distribution Protocol 61
4.3.3. Security Analysis of Key Distribution Protocol 68
4.3.4. Implementation Results of Key Distribution Protocol 69

CHAPTER 5. EFFICIENCY ANALYSIS 80
5.1. Efficiency Analysis of Key Agreement Protocol 80
5.2. Efficiency Analysis of Key Distribution Protocol 83
5.3. Key Agreement Protocol vs. Key Distribution Protocol 87

CHAPTER 6. CONCLUSION 89

REFERENCES 97

APPENDICES
APPENDIX A. IMPLEMENTATION OF KEY AGREEMENT PROTOCOL 104
APPENDIX B. IMPLEMENTATION OF KEY DISTRIBUTION PROTOCOL 125

LIST OF FIGURES

Figure **Page**

Figure 2.1. Elliptic Curve Operations 11
Figure 3.1. Certificate Creation and Verification 21
Figure 3.2. Certificate Verification using CA's Public Key 21
Figure 3.3. Certificate Verification using Bilinear Pairing 25
Figure 3.4. Diffie-Hellman Key Agreement 31
Figure 3.5. Three-Party Two-Round Key Agreement Protocol based on Diffie-Hellman... 33
Figure 3.6. Joux's One Round, Tripartite Protocol 34

LIST OF TABLES

Table **Page**

Table 3.1. Diffie-Hellman Key Agreement Example. 32
Table 3.2. Lagrange's Interpolation Formula. 36
Table 3.3. Example of Lagrange's Interpolation Formula. 37
Table 3.4. Example of Lagrange's Interpolation Formula. 41
Table 4.1. Group Key Agreement Protocol. 47
Table 4.2. Input and Setup Parameters.. 54
Table 4.3. Public Message Computation on Round-1 54
Table 4.4. SHA-512 Algorithm and BLS Signature on Round-1 115
Table 4.5. Verification of BLS Signature on Round-2 58
Table 4.6. Public Message Computation on Round-2 64
Table 4.7. SHA-512 Algorithm and BLS Signature on Round-2. 57
Table 4.8. Verification of BLS Signature on Key Agreement 57
Table 4.9. Common Key Agreement. 58
Table 4.10. Summary of Results of the Implemented Protocol. 59
Table 4.11. Group Key Distribution Protocol 87
Table 4.12. Input and Setup Parameters. 95
Table 4.13. Polynomial Generation, Commitment and Subsecret Calculation 71
Table 4.14. SHA-512 Algorithm and ElGamal Signature 72
Table 4.15. ElGamal Encryption of Subsecrets 72
Table 4.16. ElGamal Signature Verification 73
Table 4.17. ElGamal Decryption. 74
Table 4.18. Verifying Correctness of Subsecrets by Using Commitment Vector.. 74
Table 4.19. Encryption of own subsecret by U_i. 75
Table 4.20. Decryption of Received Subsecrets of Senders 75
Table 4.21. Subsecrets Verification using Commitment Vector 76
Table 4.22. Key Establishment by Lagrange Interpolation 64
Table 4.23. Summary of Results of the Implemented Protocol 78
Table 5.1. Efficiency Analysis of Key Agreement Protocol 81
Table 5.2. Total Efficiency Cost of Group Key Agreement 82
Table 5.3. Efficiency Analysis of Key Distribution Protocol 83

Table 5.4. Total Efficiency Cost of Group Key Distribution Protocol.....85
Table 5.5. Efficiency Comparision.....87
Table 6.1. Key Distribution vs. Key Agreement 95

CHAPTER 1

INTRODUCTION

Collaborative applications such as teleconferencing, distributed simulations, replicated servers, multi-user games and near field applications (NFA) have become very popular. All these group oriented applications are applied over open network. Security aspect is vital when it comes to apply on areas like homes, schools and universities to inaccessible terrains and critical infrastructures (healthcare, transportations, telecommunications, etc.). All users or servers that participate in a particular application should be able to communicate securely and exchange information that is inaccessible to any external party. Hence, there is a need to find new protocols that provide such confidential communication, named *Secure Group Key Establishment Protocol*. The aim of these protocols is to *establish a common secret key* among the users (or servers), called group key, which can be used for data encryption between them over *an open network* (Makri and Konstantinou (2011)).

There are two methods to *establish a common secret key* by communicating parties over an open network (Stinson (1995)):

- Key agreement,
- Key distribution.

Key agreement is a mechanism whereby the parties jointly establish a common secret. As to key distribution, it is a mechanism whereby one of the parties creates or obtains a secret value and then securely distributes it to other parties. In this study, both methods is applied and analyzed in two different Group Key Establishment (GKE) protocols.

1.1. Challenge

Desirable properties of a GKE are *security* and *efficiency*.

Security is attributed in terms of preventing attacks against passive and active adversary.

- A *passive adversary* is a hidden listener who tries to gain information and

compute parties' common secret key by listening to the broadcast messages between the legitimate parties.

- An *active adversary* is a dishonest participant who tries to disrupt the establishment of common key among all of the participants.

Efficiency is quantified in terms of computation, communication and round complexity.

- Computation complexity denotes the amount of computations of each participant to obtain common secret key.
- Communication complexity is the amount of broadcast messages of each participant to other ones.
- Round complexity is simply the number of rounds needed to complete the protocol. A round means one broadcasting session in which every party can cast messages to others but all at once. Minimizing round complexity is very important *challenge* in designing key agreement protocols as well as reducing other complexities (Jho et al. (2007)).

When constructing a GKE, the *main challenge* is to provide security and efficiency according to attributed and quantified terms. Two main cryptographic tools are selected in order to handle the defined *challenge*.

The typical approach of the *main challenge* requires some data to go through the complete set of parties, which by sequentially adding some private contribution, *build* the common secret key in a linear number of rounds of communication (Bellare and Rogaway (1993), Ateniese et al. (2000), Bresson et. al (2001)). The problem with this approach is that it leads very slow protocols including many rounds depending of participant numbers. The solution is to try to devise a protocol that allows for simultaneous sending of contributors to improve on communication and round complexity (Bresson and Catalano (2004)).

Two main cryptographic tools are selected in order to handle the defined *challenge*. One of them is bilinear pairing and another is verifiable secret sharing.

- The first tool and also the trending one is *bilinear pairing*s which is based on elliptic curve cryptography. *Pairing* is a map from two cyclic groups into another cyclic group. Since 2000 (Joux (2000)), it has provided important developments in key agreement protocols by reducing communication round, independent from participant number.

- The second tool is *verifiable secret sharing scheme* (Chor et al. (1985), Feldman (1987)) which is based on multiparty computation (Yao 1982), where a group of players wants to compute the output of a public function when the input is shared among the participants. *Secret sharing* (Shamir (1979), Blakely (1979)) is a way of distributing a secret among a group of participants by a dealer, each of whom is shared a piece of the secret. Then, the secret can be reconstructed only when a sufficient number of participants' secret pieces are combined together. *Verifiable secret sharing* ensures to detect malicious parties or dealer via commitments in addition to secret sharing's methodology.

1.2. Motivation and Goals

The motivation of the study is to establish a common secret key over an open network for a group of user to be used then symmetrical secure communication between them. Two different protocol models were studied in order to achieve the motivation using both two key establishment methods, namely key agreement and key distribution.

- In key agreement, bilinear pairing is applied. It is based on Lin et al.'s work (Lin et al. (2006)) and requires only constant two rounds of message transmission with equal contribution between communicating parties, which is independent of numbers of participants. In the beginning of the protocol, there is no one who has possession of the common secret. It will be generated by each participant in the protocol jointly. Authentication is provided via certificates and communication between each parties proceeds broadcasting.
- In group key distribution protocol, verifiable secret sharing is applied. It is based on Feldman's work (Feldman (1987)), and requires two rounds for message transmission, but there is a leader who has the first possession of the common secret in the beginning of the protocol, and then he distributes this secret to other parties in the protocol using verifiable secret sharing technique. Authentication is provided via certificates and communication between leader and each party and also between the parties proceeds point to point sending.

The goal of this thesis is twofold; (1) Study the construction of these protocol models and analysis in terms of security and efficiency aspects; and (2) Implement both

works to present time measure for computation cost and bit lengths for communication cost.

1.3. Organization of Thesis and Contributions

The thesis is organized in the following way;

In Chapter 2, the cryptographic preliminaries on which the rest of the thesis is based are reviewed. Firstly, some preliminaries which are number theory and abstract algebra, bilinear pairings, security assumptions and homomorphic commitments are introduced. Then, key establishment concept including key agreement, key distribution, secret sharing and verifiable secret sharing is reviewed. Finally, authentication notion is explained briefly.

In Chapter 3, a literature review of secure communication, ElGamal Encryption, key agreement, secret sharing and verifiable secret sharing are examined.

In Chapter 4, construction of group key establishment protocols for each of two methods are studied which of them is based on *key agreement* using *bilinear pairings* and another is based on *key distribution* using *verifiable secret sharing*; along with their *communication models*, *security analysis* and *implementation results*.

In Chapter 5, efficiency analysis of the two protocols is presented.

In Chapter 6, the results of this study is evaluated and open problems are discussed.

Contributions:

Group Key agreement Protocol: Its construction and security analysis depends on Lin et al.'s work (Lin et al. (2006)) and this thesis's contributions are:

- A signature algorithm is added in order to provide message integrity on each public message which is broadcasted during the two rounds. In the original work doesn't include message authentication mechanism, instead of this mechanism prefers to use dedicated secure channel.
- An open security problem is explored for this protocol and will be presented in section 4.2.2 and analyze it in section 4.2.3.
- An implementation of this protocol is realized for four users in order to present how many milliseconds calculations of each user on per round take.

- Detailed efficiency analysis is performed in terms of computation, communication and round complexity.

Group Key Distribution Protocol: Its construction depends on Feldman's work (Feldman (1987)) and this thesis's contributions are:

- Original protocol is adapted by adding certificates for the participants to meet the requirements of a secure, closed group communication.
- The protocol is rendered to be applied over open unsecure channel by adding encryption algorithm for sending messages of each user.
- A signature algorithm on publicly known commitments is added in order to prevent modifications by malicious parties. Signature algorithm satisfies the message integrity.
- Un-formal security analysis in terms of security attributes defined in section 2.2.3 is presented.
- An implementation is proposed for four user with a leader in order to present how many milliseconds calculations of each user on per round take.
- Detailed efficiency analysis is performed in terms of computation, communication and round complexity and its comparison with the previous protocol is studied.

CHAPTER 2

CRYPTOGRAPHIC BACKGROUND

The goal of this chapter is to provide the necessary background and foundations of cryptography that will be used in the subsequent chapters. Firstly, some preliminaries including abstract algebra, number theory bilinear pairings, and homomorphic commitments are introduced. Furthermore, security assumptions based on these preliminaries are studied. Then, key agreement, verifiable secret sharing and authentication concepts are reviewed.

It is a self-contained background for the readers with little knowledge in this field to understand the ideas and arguments presented in the thesis.

2.1. Cryptographic Preliminaries

Cryptographic preliminaries are studied in this part as abstract algebra, number theory, bilinear pairings, secret sharing and homomorphic commitments and security assumptions.

2.1.1. Number Theory and Abstract Algebra

Number Theory and Abstract Algebra are the mathematical foundation of modern cryptography and the cornerstone of provable security of cryptographic schemes. Numerous cryptographic algorithms are designed around results from them.

Number Theory:

Some basic definitions (Menezes et al. (1996)) of Number Theory as the following;

The set of integers $\{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$ is denoted by symbol Z.

Definition 1 (Greatest Common Divisor): There is an integer c which is a common divisor of integers a and b such as $c|a$ and $c|b$. A non-negative integer d is the *greatest common divisor* of the integers a and b, denoted $d = gcd\ (a, b)$, if

(1) d is a common divisor of a and b; and

(2) Whenever $c|a$ and $c|b$ and then $c|d$.

Definition 2 (Relatively Prime): Two integers a and b are said to be *relatively prime* or *coprime* if $gcd(a,b) = 1$.

Definition 3 (Euler Phi Function): For $n \geq 1$, let $\phi(n)$ denote the number of integers in the interval $[1,n]$ which are relatively prime to n. The function ϕ is called the *Euler phi function* (or the *Euler totient function*).

Definition 4 (Congruent and Modulus): If a and b are integers, then a is said to be *congruent to* $modulo\ n$, written $a \equiv b\ (mod\ n)$, if n divides $(a-b)$. The integer n is called the *modulus* of the congruence.

Definition 5 (Equivalence Class): The *integers* $modulo\ n$, denoted Z_n, is the set of (equivalence classes of) integers $\{0, 1, 2, \ldots, n-1\}$. Addition, subtraction, and multiplication in Z_n are performed $modulo\ n$.

Definition 6 (Multiplicative Inverse): Let $a \in Z_n$. The *multiplicative inverse* of $a\ modulo\ n$ is an integer $x \epsilon Z_n$ such that $ax \equiv 1(mod\ n)$. If such an x exists, then it is unique, and a is said to be *invertible*, or *unit*; the inverse of a is denoted by a^{-1}.

Definition 7 (Division on Z_n) : Let $a, b \in Z_n$. *Division* of a by $b\ modulo\ n$ is the product of a and $b^{-1}\ modulo\ n$, and is only defined if b is invertible $modulo\ n$.

Definition 8 (Multiplicative Group): The *multiplicative group* of Z_n is $Z_n^* = \{a \in Z_n \mid gcd(a,n) = 1\}$. In particular, if n is prime, then $Z_n^* = \{a \mid 1 \leq a \leq n-1\}$.

Definition 9 (Order of Z_n): The *order* of Z_n^* is defined to be the number of elements in Z_n^*, namely $|Z_n^*|$.

Definition 10 (Order of Element in Z_n^*): Let $a \in Z_n^*$. The order of a, denoted $ord(a)$, is the least positive integer t such that $a^t \equiv 1(mod\ n)$.

Definition 11 (Generator): Let $\alpha \in Z_n^*$. If the order of α is $\phi(n)$, then α is said to be a *generator* or *a primitive element* of Z_n^*. If Z_n^* has a generator, then Z_n^* is said to be *cyclic*.

Abstract Algebra:

Basic algebraic objects and their properties (Menezes et al. (1996)) as the following;

Definition 12 (Binary Operation): A *binary operation* $\circ$ on a set S is a mapping

from $S \times S$ to S notated as $S \times S \rightarrow S$. That is, $\circ$ is a rule which assigns to each ordered pair of elements from S an element of S.

Definition 13 (Group): A *group* $(G,\circ)$ consists of a set G with a binary operation $\circ$ on G satisfying the following three axioms.

(1) The group operation is *associative*. That is, $a \circ (b \circ c) = (a \circ b) \circ c$ for all $a, b, c \in G$.

(2) There is an element $e \in G$, called the *identity element*, such that $a \circ e = e \circ a = a$ for all $a \in G$.

(3) For each $a \in G$ there exists an element $a^{-1} \in G$, called the *inverse* of a, such that $a \circ a^{-1} = a^{-1} \circ a = e$.

A group G is abelian (or commutative) if, furthermore,

(4) $a \circ b = b \circ a$ for all $a, b \in G$.

Note that multiplicative group notation has been used for the group operation. If the group operation is addition, then the group is said to be an *additive* group, the identity element is denoted by 0, and the inverse of a is denoted $-a$.

Definition 14 (Finite Group and Group Order): A group G is *finite* if $|G|$ is finite. The number of elements in a finite group is called its *order*.

Definition 15 (Subgroup): A non-empty subset H of a group G is a *subgroup* of G if H is itself a group with respect to the operation of G. If H is a subgroup of G and $H \neq G$, then H is called a *proper* subgroup of G.

Definition 16 (Cyclic Group): A group G is *cyclic* if there is an element $\alpha \in G$ such that for each $b \in G$ there is an integer i with $b = \alpha^i$. Such an element α is called a *generator* of G.

Definition 17 (Element Order in Group G): Let G be a group and $a \in G$. The *order* of a is defined to be the least positive integer t such that $a^t = 1$, provided that such an integer exists. If such a t does not exist, then the order of a is defined to be ∞ *(infinitive)*.

Definition 18 (Group Homomorphism): Let G and G' be groups and let $\phi: G \rightarrow G'$ is a mapping preserves the group operation; $\phi(ab) = \phi(a)\phi(b)$ for all $a, b \in G$, then ϕ is called a *group homomorphism*.

Definition 19 (Group Isomorphism): Let G and G' be groups and let $\phi: G \rightarrow G'$ is a one to one mapping from G onto G' preserves the group operation; $\phi(ab) = \phi(a)\phi(b)$ for all $a, b \in G$, then ϕ is called a *group isomorphism*.

Definition 20 (Ring): A ring $(R, +, \times)$ consists of a set R with two binary operations arbitrary denoted $+$ (addition) and $\times$ (multiplication) on R, satisfying the following axioms.

(1) $(R, +)$ is an abelian group with identity denoted 0.

(2) The operation $\times$ is associative. That is, $a \times (b \times c) = (a \times b) \times c$ for all $a, b, c \in R$.

(3) There is a multiplicative identity denoted 1, with $1 \neq 0$, such that; $1 \times a = a \times 1 = a$ *for all* $a \in R$.

(4) The operation $\times$ is *distributive* over $+$. That is, $a \times (b + c) = (a \times b) + (a \times c)$ and $(b + c) \times a = (b \times a) + (c \times a)$ for all $a, b, c \in R$.

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

Definition 21 (Invertible Element in Ring R): An element a of a ring R is called a *unit* or an *invertible element* if there is an element $b \in R$ such that $a \times b = 1$.

Definition 22 (Ring Homomorphism): Let R and R' be rings and let $\phi: R \to R'$ is a mapping preserves the ring operation; $\phi(ab) = \phi(a)\phi(b)$ and $\phi(a + b) = \phi(a) + \phi(b)$ for all $a, b \in R$, then ϕ is called a *ring homomorphism*.

Definition 23 (Ring Isomorphism): Let R and R' be rings and let $\phi: R \to R'$ is a one to one mapping from R onto R' preserves the ring operation $\phi(ab) = \phi(a)\phi(b)$ and $\phi(a + b) = \phi(a) + \phi(b)$ for all $a, b \in R$, then ϕ is called a *ring isomorphism.*

Definition 24 (Field): A *field* is a commutative ring in which all non-zero elements have multiplicative inverses.

Definition 25 (Characteristic of a Field): The *characteristic* of a field is 0 if $1 + 1 + \ldots + 1\ (m\ times)$ is never equal to 0 for any $m \geq 1$. Otherwise, the characteristic of the field is the least positive integer m such that $\sum_{i=1}^{m} 1$ equals 0.

Definition 26 (Subset of Field): A subset F of a field E is a *subfield* of E if F is itself a field with respect to the operations of E. If this is the case, E is said to be *extension field* of F.

Definition 27 (Polynomial Coefficients and Degree, Constant Polynomial, Zero Polynomial): If R is a commutative ring, then a *polynomial* in the indeterminate x over ring R is an expression of the form $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$ where each $a_i \in R$ and $n \geq 0$. The element a_i is called the *coefficient* of x^i in $f(x)$. The largest integer m for which $a_m \neq 0$ is called the *degree* of $f(x)$, denoted

$deg\, f(x)$; a_m is called the *leading coefficient* of $f(x)$. If $f(x) = a_0$ (*a constant polynomial*) and $a_0 \neq 0$, then $f(x)$ has degree 0. If all the coefficients of $f(x)$ are 0, then $f(x)$ is called the *zero polynomial* and its degree, for mathematical convenience, is defined to be $-\infty$. The polynomial $f(x)$ is said to be *monic* if its leading coefficient is equal to 1.

Definition 28 (Polynomial Ring): If R is a commutative ring, the *polynomial ring* $R[x]$ is the ring formed by the set of all polynomials in the indeterminate x having coefficients from R. The two operations are the standard polynomial addition and multiplication, with coefficient arithmetic performed in the ring R.

Definition 29 (Irreducible Polynomial): Let $f(x) \in F[x]$ be a polynomial of degree at least 1. Then $f(x)$ is said to be *irreducible over* F if it cannot be written as the product of two polynomials in $F[x]$, each of positive degree.

Definition 30 (Finite Field): A *finite field* is a field F which contains a finite number of elements. The order of F is the number of elements in F.

Definition 31 (Multiplicative Group of Field F_q^*): The non-zero elements of F_q form a group under multiplication called the *multiplicative group* of F_q, denoted by F_q^*.

Definition 32 (Generator of Field): A generator of the cyclic group F_q^* is called a *primitive element* or generator of F_q.

Definition 33 (Lagrange Interpolation): (Gasca and Sauer (2000)) Let there be a polynomial $f(x) = a_0 + a_1x + a_2x^2 + ... + a_{k-1}x^{k-1}$ over a field F_q of degree $k-1$, for set of pairs (x_i, y_i) where $i = 1, 2, \ldots, k \in F_q \times F_q$, which is $f(x_i) = y_i$. All coefficients of $f(x)$ can be efficiently computed from these k pairs using lagrange interpolation formula; such that $f(x) = \sum_{i=1}^{k} L_i(x) y_i$.

- Lagrange coefficient polynomial is $L_i(x) = \prod_{j=1, j\neq i}^{k} \frac{x - x_j}{x_i - x_j}$.
- Lagrange coefficient is $L_i(0) = \lambda_i = \prod_{j=1, j\neq i}^{k} \frac{x_i}{x_j - x_i}$.

2.1.2. Bilinear Pairings

Elliptic curve E is a curve given by the equation of the form defined over finite field F_p with $p > 3$, such that $E: y^2 = x^3 + ax + b\ (mod\ p)$, where a and b constants satisfying $4a^3 + 27b^2 \neq 0\ (mod\ p)$. Then there are points $P(x, y)$ on the curve, together with $\mathcal{O} = (x, \infty)$ the point at infinity (Yuen (2010)).

Let E be an abelian group under the operation addition $+$ defined as follows;

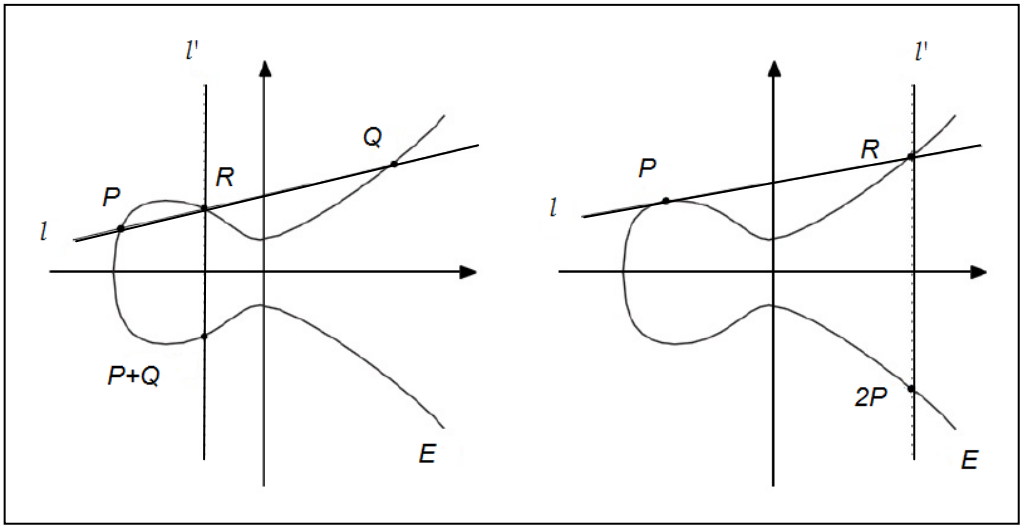

Figure 2.1. Elliptic Curve Operations.

Elliptic Curve Group Operation: Let $P, Q \in E$, l be the line containing P and Q (tangent line to E if $P = Q$), and R, the third point of intersection of l with E. Let l' be the line connecting R and $\mathcal{O}$ (see Figure 2.1). Then $P + Q$ is the point such that l' intersects E at R, $\mathcal{O}$ and $P + Q$. Let $P + Q = -R$. Then the point multiplication for $k \in Z$ defined as (Yuen (2010));

$$[k]P = \begin{cases} P + P + \ldots + P \ (k \ times) & for \ k > 0, \\ \mathcal{O} & for \ k = 0, \\ [-k](P) & for \ k < 0. \end{cases}$$

Modified Weil Pairing (Boneh and Franklin (2001)): Let p be a prime such that $p = 2 \ (mod \ 3)$ and $p = 6q - 1$ for some prime $q > 3$. Let E be a super-singular elliptic curve defined by $y^2 = x^3 + 1$ over F_p. The set of rational points $E[F_p] = \{ (x, y) \in F_p \times F_p : (x, y) \in E \}$ forms a cyclic group of order $p + 1$. Furthermore, because $p + 1 = 6q$ for some prime q, the set of points of order q in $E[F_p]$ form a cyclic subgroup, denoted as G_q. Let $P \in E/F_p$ be a generator of the group of points with order $q = (p + 1)/6$. Let G_T be the subgroup of $F_{p^2}^*$ that contains all elements of order q. The Weil pairing on the curve E/F_{p^2} is a mapping : $G_q \times G_q \rightarrow G_T$. The modified weil

pairing is defined as $G_q \times G_q \rightarrow G_T$, $e(P,Q) = (P, \phi(Q))$, where $\phi(x,y) = (\xi x, y)$, $1 \neq \xi \in F_{p^2}^*$ is a solution of $x^3 - 1 = 0 \ (mod\ p)$ and G_q is the group of points with order q. The *modified Weil pairing* then satisfies the following properties:

1. Bilinear:
   $e(P_1 + P_2, Q) = e(P_1, Q)\, e(P_2, Q)$,
   $e(P, Q_1 + Q_2) = e(P, Q_1)\, e(P, Q_2)$,
   $e(aP, bQ) = e(P, Q)^{ab}$,
   where for all $P, P_1, P_2, Q, Q_1, Q_2 \in G_q$ and $a, b \in Z_q^*$.
2. Alternative: $e(P,Q) = e(Q,P)^{-1}$.
3. Nondegenerate: If P is generator of G_q, then $e(P,P) \neq 1$.
4. Computable: There is an efficient algorithm to compute $e(P,Q)$ for all $P, Q \in G_q$.

2.1.3. Security Assumptions

There are some security assumptions. Public key cryptography is based on the intractability of solving these assumptions.

- *Discrete Logarithm (DLog) Assumption* (Menezes et al. (1996)): The DLog assumption is that, given elements $g, y \in G$, to output element $x = log_g y \in G$, where G is a cyclic group.
- *Computational Diffie-Hellman (CDH) Assumption* (Menezes et al. (1996)): The CDH is that, given elements $g, g^a, g^b \in G$ for unknown elements $a, b \in Z_q^*$, to output g^{ab} where G is a cyclic group.
- *Decisional Diffie-Hellman (DDH) Assumption* (Boneh (1998)): The DDH assumption is that, given elements $g, g^a, g^b, g^c \in G$ for unknown $a, b, c \in Z_q^*$, to decide if $ab = c$ where G is a cyclic group.

Other security assumptions are in the cyclic groups: G_q, G_T of prime order q, equipped with a pairing e: $G_q \times G_q \rightarrow G_T$.

- *Bilinear Diffie-Hellman (BDH) Assumption* (Boneh and Franklin (2001)): Given $g, g^a, g^b, g^c \in G$ for unknown $a, b, c \in Z_q^*$, to output $e(g,g)^{abc}$.

- *Decisional Bilinear Diffie-Hellman (DBDH) Assumption* (Boneh and Franklin (2001)): Given $g, g^a, g^b, g^c \in G_q$ and $T \in G_T$ for unknown $a, b, c \in Z_q^*$, to output 1 if $T = e(g,g)^{abc}$, and to output 0, otherwise.

DLog assumption implies CDH assumption, CDH assumption implies DDH assumption and BDH assumption. DDH and BDH assumptions imply DBDH assumption.

2.1.4. Homomorphic Commitments

Commitment schemes are fundamental components of many cryptographic protocols. A commitment scheme allows a *committer* to publish a value, called the *commitment* (say C), which binds her to a message m *(binding)* without revealing it *(hiding).* Later, she may open the *commitment* C and reveal the *committed message* m to a verifier, who can check that the message is consistent with the commitment. Damgard surveys the basics of commitment schemes in (Damgard (1999)).

Let $C(\alpha, [r])$ be a commitment to α, where r is an optional randomness parameter. For the homomorphic commitments to be used, given $C_1 = C(\alpha_1, [r_1])$ and $C_2 = C(\alpha_2, [r_2])$, it is $C_1 \cdot C_2 = C(\alpha_1 + \alpha_2, [r_1 + r_2])$.

Let $g, h \in G$. The *discrete logarithm (DLog) commitment scheme* is the most commonly used homomorphic commitment. It is of the form $C_{\langle g \rangle}(\alpha) = g^\alpha$ with computational *hiding* (secrecy) under the *DLog assumption* and unconditional *binding* (correctness). Pedersen (Pedersen (1991)) presented another homomorphic commitment of the form $C_{\langle g,h \rangle}(\alpha, r) = g^\alpha h^r$ with unconditional hiding but computational binding under the *DLog assumptio*n in section 2.1.3.

In this thesis, DLog commitment scheme is studied in the protocol of Section 4.3.

2.2. Key Establishment

According to Menezes et al. (Menezes et al. (1996)); a *protocol* is a multiparty algorithm, defined by a sequence of steps precisely specifying the actions required of two or more parties in order to achieve a specified objective. *Key establishment* is a process or protocol whereby a shared secret becomes available to two or more parties,

for subsequent cryptographic use. Key establishment may be broadly subdivided into *key agreement* and *key distribution*.

2.2.1. Key Agreement

Key agreement protocol or mechanism is a key establishment technique in which a shared secret is derived by two (or more) parties as a function of information contributed by, or associated with, each of these, (ideally) such that no party can predetermine the resulting value.

The first key agreement protocol is proposed in 1976 by Diffie and Hellman (Diffie and Hellman (1976)) which two participants respectively hold a secret exponent and send its corresponding public value to the other participant, then a common session key can be established using the secret exponent and the opposite participant's public value. However, two party Diffie-Hellman protocol suffers man in the middle attack since it does not provide entity authentication (see Section 3.1 for authentication notion and Section 3.2.1 for details of Diffie-Helman protocol). There are unauthorized third parties named intruder, adversary, attacker, eavesdropper or impersonator in addition to legitimate parties. In a key agreement protocol, there are two kinds of adversaries: *passive adversary* and *active adversary*.

- A *passive adversary* is a hidden listener who tries to compute participants' common secret key by listening to the broadcast mesages between the legitimate participants.
- An *active adversary* is a dishonest participant who tries to disrupt the establishment of a common key among all of the participants. An active adversary can try to be looks like an honest participant into believe that he has computed the same common key as the other honest participants do.

In unauthenticated key agreement, impersonation is possible by active adversary. So, firstly entity authentication must be provided to handle this issue (See Section 2.3 and 3.1). When examining security of key agreement protocol, there are some attributes. They are regarding of security against active adversary for authenticated key agreement protocols (Blake-Wilson et al. (1997)). Their inspections are as the following; which are generally believed to be necessary for an *authenticated key agreement protocol* (Blake-Wilson et al. (1997));

- *Known session key security*. Each execution of the protocol should result in a unique secret session key. The compromise of one session key should not compromise the keys of other sessions (e.g., the parallel sessions, previous sessions and future sessions).
- *Forward secrecy* If the long-term private keys of one or more entities are compromised, the secrecy of previously established session keys should not be affected. We say that a protocol has *partial forward secrecy* if one or more but not all the entities' long-term keys can be corrupted without compromising previously established session keys, and we say that a protocol has *perfect forward secrecy* (PFS) if the long-term keys of all the entities involved may be corrupted without compromising any session key previously established by these entities.
- *Key-compromise impersonation resilience* The compromise of entity *A*'s long-term private key will allow an adversary to impersonate *A*, but it should not enable the adversary to impersonate other entities to *A*.
- *Unknown key-share resilience.* Entity *A* should not be able to be coerced into sharing a key with entity *C* when in fact *A* thinks that he is sharing the key with some entity *B*.
- *Key Control* Neither entity should be able to force the session key to be a preselected value.

These security attributes' inspections are available in Section 4.2.3 for the *Key Agreement Protocol using Bilinear Pairing*.

2.2.2. Key Distribution by Secret Sharing

A *key distribution protocol* or mechanism is a key establishment technique where one party creates or otherwise obtains a secret value, and securely transfers it to the other(s).

One of the methods of *key distribution* is secret sharing. The notion of secret sharing was introduced independently by Shamir (Shamir (1979)) and Blakey (Blakely (1979)) in 1979. Since then, it has remained an important topic in cryptographic research. The idea is to start with a secret chosen by one party, and divide it into pieces called *shares (subsecrets or shadows in some technical documents)* by that party, which

are distributed among other users such that the pooled shares of specific subsets of users allow reconstruction of the original secret.

For integers n, t and δ such that $n \geq t + \delta > t \geq 0$, an $(n, t + \delta, t)$ *secret sharing scheme* is a protocol used by a dealer (who has a sole possession of secret before the protocol) to share a secret s among a set of n parties in such a way that any subset of $t + \delta$ or more parties can compute the *secret* s, but subsets of size t(threshold) or fewer cannot (Kate and Goldberg (2009)).

In the thesis, Shamir's Secret Sharing Scheme (Shamir (1979)) is used. It is a form of secret sharing, where a secret s is divided into parts using a t degree polynomial $f(x)$, giving to each participant its own unique part (subsecret) as evaluation of polynomial, where some of the parts $t + \delta$ or all of them n are needed in order to reconstruct the secret s using lagrange interpolation in section 2.1.1. All elements n, t, coefficients of P are in a *Finite Field* F.

Details of Shamir's Secret Sharing Scheme are available in Section 3.3.1.

2.2.3. Verifiable Secret Sharing

Multiparty computation is typically accomplished by making secret shares of the inputs, and manipulating the shares to compute some function. To handle active adversaries (that is, adversaries that corrupt parties and then make them deviate from the protocol), the secret sharing scheme needs to be *verifiable* to prevent the deviating parties from throwing off the protocol. To solve this problem, Chor et al. (Chor et al. (1985)) introduced verifiability in secret sharing, which led to the concept of *verifiable secret sharing (VSS)*.

An (n, t)-*Verifiable Secret Sharing (VSS) scheme* consists of two phases: the *sharing phase* and the *reconstruction phase*.

Sharing phase: A dealer P_d distributes a *secret* $s \in K$ among n parties, where K is a sufficiently large key space. At the end of the *sharing phase*, each honest party P_i holds a *share* s_i of the distributed *secret* s.

Reconstruction phase: In this phase, each party P_i broadcasts its *secret share* s'_i and a reconstruction function is applied in order to compute the secret $s = Rec(s'_1, s'_2, \ldots, s'_n)$ or output $\perp$ indicating that P_d is malicious. For honest parties $s'_i = s_i$, while for malicious parties s'_i may be different from s_i or even absent.

It has two security requirements: *Secrecy* and *Correctness*.

- *Secrecy (VSS-S)* A t-limited adversary who can compromise t parties cannot compute s during the *Sharing phase*.
- *Correctness (VSS-C)* The reconstructed value z should be equal to the shared secret s or every honest party concludes that the dealer is malicious by outputting $\perp$.

In the thesis, Feldman's VSS scheme (Feldman (1987)) is applied which is based on *DLog homomorphic commitments* in Section 2.1.4. and details of Feldman's VSS are available in Section 3.3.1.

2.3. Authentication

Authentication is the cornerstone of secure communication (details are in Section 3.1). Without some form of authentication (Public Key Infrastructure in Section 3.1.1 or Identity (ID) based Infrastructure in Section 3.1.4), all the other common security properties such as integrity or confidentiality do not make much sense.

Authentication or *identification* is a process to provide the assurance to one party participates a protocol that identity of the second party involved in the same protocol is who he/she/it claims to be (Menezes et al. (1996)).

It is generally based on long-term keys which can be associated with identities. There are two main approaches to provide authentication of public keys: Public Key Infrastructure (PKI, see Section 3.1.1) and Identity Based Infrastructure (ID-Based Infrastructure, see Section 3.1.4).

Large-scale deployments of public-key cryptography generally employ the services of Certificate Authority (CA) as a part of PKI, which generates a digital certificate to bind an entity with its public key.

Although the notion of a certificate is very simple, there are many practical difficulties with managing certificates, such as key revocation.

In 1984, Shamir (Shamir (1984)) introduced the notion of ID-Based cryptography to alleviate many of the problems inherent with managing certificates. In identity-based cryptography, a public key can be derived from a widely known identity, such as an email address or phone number. Entity's private key can be generated by a trusted authority called private key generator (PKG) with the help of master secret key

of PKG and then transfer the entity's private key via a secure channel. Construction of ID-Based encryption remains as an open problem until the seminal work proposed by Boneh-Franklin in 2001 using bilinear pairings (Boneh and Franklin (2001), see Section3.1.4 for details of the work). However ID-Based Infrastructure suffers the key escrow problem since the PKG generates private keys of entities using his master secret.

CHAPTER 3

LITERATURE SURVEY

In this chapter, some existing works are reviewed which are related to the group key establishment protocols as analyzed in the thesis.

3.1. Secure Communication

Secure communication can be defined as the transmission of data from a sender to a receiver with one or more of the properties of authentication, confidentiality and integrity (Karagodin (2005)).

- *Authentication* is the assurance to one entity that another entity is who he/she/it claims to be.
- *Integrity* is the assurance to an entity that data has not been intentionally or unintentionally altered while in transmitting.
- *Confidentiality* is the assurance to an entity that communicated information is not disclosed to unauthorized eavesdroppers.

Furthermore, transmitted data has been sent and received by the parties needs to be binding to the related parties (they cannot deny). *Non-repudiation* is a way to guarantee that the sender of a message cannot later deny having sent the message and that the recipient cannot deny having received the message.

PKI and ID-based Cryptography ensure these properties. Cryptographic tools such as public key encryption and digital signatures are primary ingredients of PKI and ID-Based Cryptography in achieving these properties.

3.1.1. PKI-Based Cryptography

A PKI is a set of hardware, software, people, policies and procedures needed to create, manage, distribute, use, store and revoke public IDs and related certificates of entities (Stallings (2006)). It is closely linked to the asymmetric key encryption, digital signatures and encryption services, but to enable these services are used digital

certificates. So, it facilitates storage and exchanges electronic data in a secure way; safety is ensured by using public key cryptography and the types of security services offered are *authenticity, confidentiality, integrity* and *non-repudiation* (Vatră (2009)).

In asymmetrical communication, each entity (such as device, person, or connection end-point) has a cryptographic private-public key pairs. Private key is kept secret by the related entity and public key is distributed to other entities who the related party wants to communicate securely. Other entities can encrypt message using the related party's public key and then send to him/her. The related party can decrypt the received encrypted message using his/her private key. Furthermore, the private key can be used to generate a digital signature on a message, and anyone knowing the public key can confirm that the signature is authentic.

The most basic function of PKI is to support the distribution of public keys of entities. Public keys are generally distributed in the form of certificates in PKI. In this way, anyone using a public key can be certain that it is the correct public key of the entity who he intended to communicate. Otherwise, an intruder could substitute his own public key to the related entity and the related entity encrypts his message using intruder's public key thinking the public key belongs to other legitimate entity who he intended to communicate. However, the intruder can decrypt the encrypted message instead the legitimate entity.

A certificate is a data item comprising a public key value, together with information identifying the holder of the corresponding private key, all digitally signed by a trusted party called a certification authority (CA). Other parties who have certificate of the related entity, can verify correctness of the certificate in two ways:

1. In Figure 3.1, User A generates her public-private key pair and applies a legitimate CA. CA signs her ID and public key using his secret key and sends to her. When User B wants to verify the validity of A's certificate, he can directly applies the CA if her certificate is in subscription list. This way needs interaction with CA, however it assures User A if B's certificate private key is revoked or not (or User B's private key may be compromised).

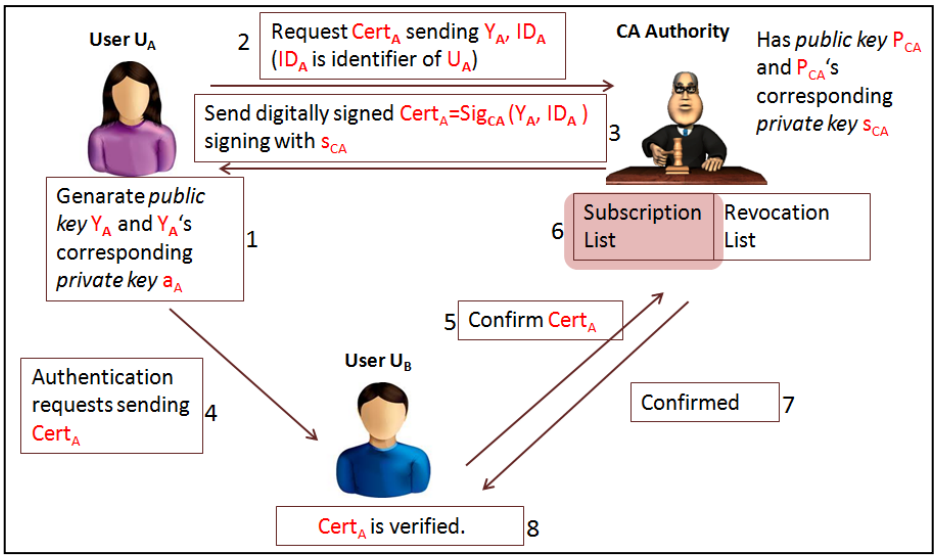

Figure 3.1. Certificate Creation and Verification.

2. In Figure 3.2, User B verifies the validity of A's certificate in another way without any interaction with CA. He can use CA's public key and applies a verification algorithm depending on the cryptosystem CA uses. However User A cannot be sure if B's certificate is revoked or not. There is an example proposed using Pairing Cryptography in Section 3.1.2.

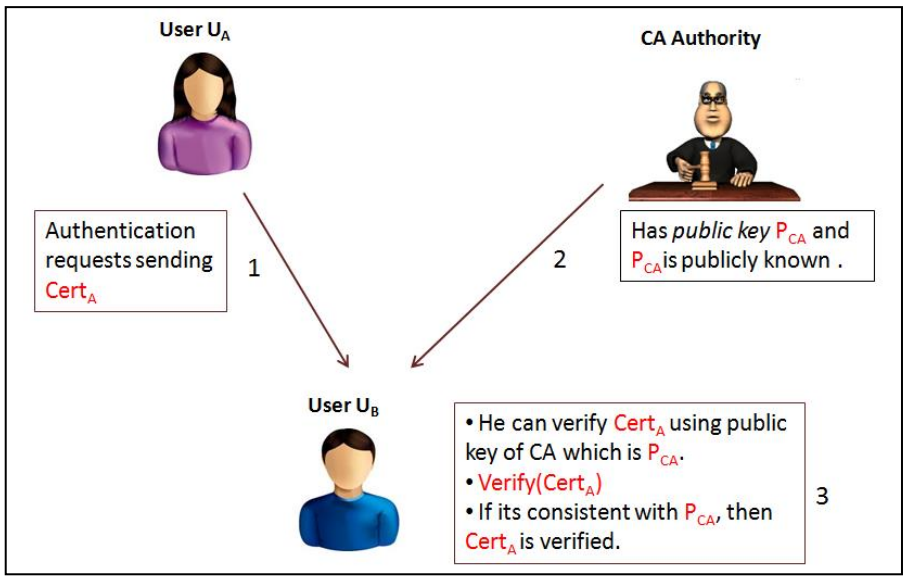

Figure 3.2. Certificate Verification using CA's Public Key.

The main components of PKI are (Menezes et al. (1996)):

- *Certifying Authorities* are basic components of a PKI to issue and revoke digital certificates.
- *Registration Authorities* validates requests for issuing certificates and identity of end users.
- *Repository* stores and distributes certificates and certificate revocation lists (CRL), they are issued periodically by the CA and are lists of certificates that are no longer valid.
- *Archives*; an archive is responsible for long-term storage of information in the name of the CA, certifying that the information archived it was good when that was received and was not changed while it was archived.
- *End Entity* represents the end users for digital certificates that were issued.

In this study, all entities in the both two proposed protocols (Section 4.2 and Section 4.3) apply a legitimate CA in order to provide secure communication during the protocols.

3.1.2. Boneh-Lynn-Shacham (BLS) and ElGamal Signature Schemes

Digital Signatures are equivalent to traditional handwritten signatures in many respects. According to Menezes et al. (Menezes et al. (1996));

- A *digital signature* is a data string which associates a message (in digital form) with some originating entity.
- A *digital signature generation algorithm* is a method for producing a *digital signature*.
- A *digital signature verification algorithm* is a method for verifying that a digital signature is authentic.
- A *digital signature scheme* consists of a *signature generation algorithm* and an associated *verification algorithm*.
- A *digital signature signing* process consists of a (mathematical) *digital signature generation algorithm*, along with a method for formatting data into messages which can be signed.
- A *digital signature verification process* consists of a *verification algorithm*, along with a method for recovering data from the message.

Digital signatures are one of the tools for secure communication, providing *authentication*, *data integrity*, and *non-repudiation*. A valid digital signature gives a recipient reason to believe that the message was created by a known sender, such that the sender cannot deny having sent the message (*authentication* and *non-repudiation*) and that the message was not altered in transit (*integrity*). Significant usages of digital signature are in certification of public key to bind the identity of a user to a public key, so that at some later time, other entities can authenticate a public key without assistance from a CA. Another usage is in private key generation by trusted third party (TTP) in ID-based cryptography to bind identity of user to a private key using TTP's signature; so that other entities can encrypt messages using the identity as public key.

Diffie-Hellman first described the notion of a digital signature scheme in 1976 (Diffie and Hellman (1976)), although they only conjectured that such schemes existed. Soon afterwards, Ronald Rivest, Adi Shamir, and Len Adleman invented the RSA algorithm, which could be used to produce primitive digital signatures (Rivest et al. (1978)). In 1984 Tahel ElGamal described a digital signature scheme which is based on the difficulty of computing discrete logarithms (ElGamal (1985)). Then variants of ElGamal was proposed such as Schnorr (Schnorr (1989)) and DSA (FIPS 186 (1994)). In such schemes, signatures are generally comprised of a pair of integers modulo p, where p is the order of the underlying group with generator P; $G_p = \langle P \rangle$. Boneh, Lynn and Shacham (BLS) (Boneh et al. (2001)) proposed the first signature scheme in which signatures are comprised of a single group element.

I. BLS Signature Scheme (Boneh et al. (2001)):

Setup

The BLS signature scheme utilizes a bilinear pairing e on (G_p, G_T) of Z_p^*, which is $e: G_p \times G_p \rightarrow G_T$ for which the CDH assumption (see Section 2.1.3) in G_p is intractable and P is generator of G_p. It also uses a cryptographic hash function $H: \{0,1\}^* \rightarrow G_p/\{\infty\}$.

Signature Generation

A user named Alice;

- Selects randomly a private key a in the interval $[1, p-1]$.

- Calculates the public key $P_a = aP$, which is the group element of G_p and publishes it.

Signing

Given the private key a and some message m, Alice computes the signature by hashing $m \in \{0,1\}^*$, as $h_m = H(m)$ and then, by multiplying it with the private key $\sigma = a \cdot h_m$.

Verification

Any party possessing Alice's public key P_a can verify the signature by computing $h_m = H(m)$ and checking that (P, P_a, h_m, σ) is a valid Diffie-Hellman quadruple. This is precisely an instance of the DHP assumption (see Section 2.1.3) in G_p which the verifier can solve by checking that;

$$e(P, \sigma) = e(P_a, h_m)$$
$$e(P, a \cdot h_m) = e(a \cdot P, h_m)$$
$$e(P, h_m)^a = e(P, h_m)^a$$

BLS short signature scheme can be aggregated and also has been used to design protocols for threshold, multi-signature and blind signatures (Boldyreva (2003)).

In this study, first protocol which is *Key Agreement using Bilinear Pairing* applies this signature scheme in the implementation (see Section 4.2.4) since the protocol is based on pairing cryptography and signature scheme is consistent with all parameters generated for the protocol (see Appendix A).

There is a sample for certificate creation and verification using BLS signature who uses the cryptosystem based on pairing cryptography.

A Sample; Certificate Creation and Verification with BLS Short Signature:

A party U_A who is willing to have a certificate generates her static public key $Y_A = a_A P$ where a_A is random number used as the long term private key selected by U_A and applies the CA to obtain her certificate $Cert_A$ which contains static public key Y_A and an unique identifier string U_A (such as U_A's name). CA signs this information using his private key s such as; $Cert_A = sH(Y_A, ID_A)$ where H is map-to-point hash function, $H: \{0,1\}^* \rightarrow G_p/\{\infty\}$ and then delivers to U_A (see Figure 3.3).

When the user U_A send to her certificate to other parties such as U_B into the protocol, U_B can verify the validity of the certificate by either applying to CA to question if it is in subscription list or revocation list (or never issued) (see Figure 3.3); or verifying by themselves using the public key of CA and pairings:

$$e(P_{cert}, H(Y_A, ID_A)) = e(P, Cert_A)$$
$$e(sP, H(Y_A, ID_A)) = e(P, sH(Y_A, ID_A))$$
$$e(P, H(Y_A, ID_A))^s = e(P, H(Y_A, ID_A))^s$$

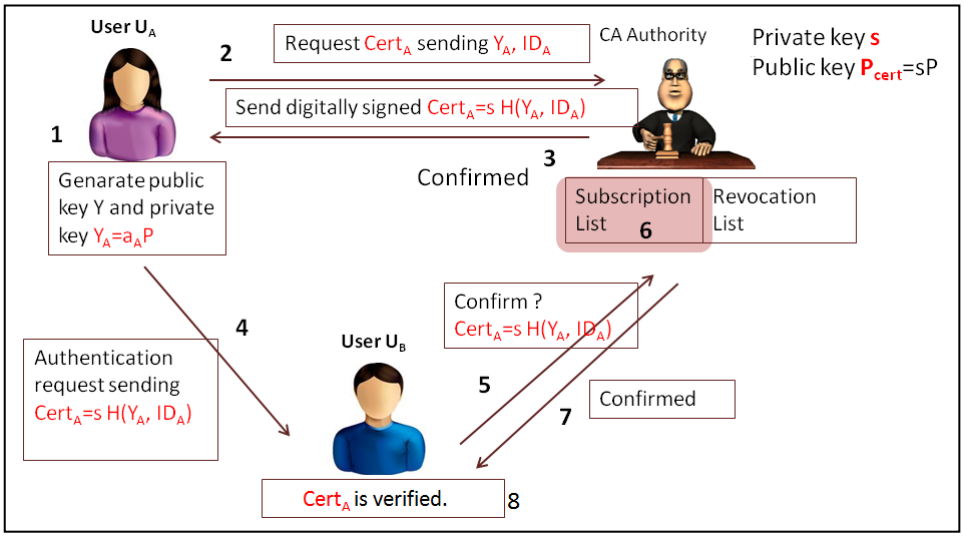

Figure 3.3. Certificate Verification using Bilinear Pairings.

II. ElGamal Signature Scheme (ElGamal (1985))

Setup

The following system parameters are shared between the users;

- H is a collision-resistant hash function.
- p is a large prime such that computing discrete logarithms modulo p is difficult.
- $g < p$ be a randomly chosen generator of the multiplicative group of integers modulo p.

Signature Generation

A user named Alice;

- Selects randomly a private key a in the interval $[1, p-1]$.
- Calculates the public key $P_a = g^a \ mod \ p$.

Signing

Given the private key a and some message m, Alice performs following steps;

- Chooses a random k such that $0 < k < (p-1)$ and $gcd\ (k,(p-1)) = 1$.
- Computes $r \equiv g^k\ (mod\ p)$.
- Computes $s \equiv (H(m) - ar)k^{-1} (mod\ p-1)$.
- If s=0, then starts over again.

The pair (r, s) is the digital signature of m. Alice repeats these steps for every signature.

Verification

Any party possessing Alice's public key P_a can verify the signature (r, s) of message m by computing $g^{H(m)} \equiv {P_a}^r r^s\ (mod\ p)$.

The verifier accepts a signature if all conditions are satisfied and rejects it otherwise.

In this study, second protocol which is *Key Distribution using Verifiable Secret Sharing* applies this signature scheme in the implementation (see Section 4.3.4) since signature scheme is consistent with all parameters generated for the protocol (see Appendix B).

A brief evaluation of implementation of BLS and ElGamal signature scheme is available in Chapter 6.

3.1.3. ElGamal Encryption Scheme

Encryption algorithm is another cryptographic tool for secure communication, providing *authentication*, *confidentiality*, *data integrity*, and *non-repudiation*. An encrypted message gives a recipient reason to believe that the message was created by a known sender, such that the sender cannot deny having sent the message (*authentication* and *non-repudiation*), ensures the privacy of the message (confidentiality) and that the message was not altered in transit (*integrity*).

ElGamal encryption scheme is an asymmetric key encryption algorithm for public-key cryptography which is based on the Diffie-Hellman key exchange (Diffie and Hellman (1976)) It can be defined over any cyclic group G and its security depends on the difficulty of computing discrete logarithm in G.

It consists of two components: encryption algorithm and decryption algorithm.

Setup

The following system parameters are shared between the users named Alice (performs decryption algorithm) and Bob (performs encryption algorithm);

- p is a large prime such that computing discrete logarithms modulo p is difficult.
- $g < p$ be a randomly chosen generator of the multiplicative group G of integers modulo p.

Alice ;

- Selects randomly a private key a in the interval $[1, p-1]$.
- Calculates the public key $P_a = g^a \ mod \ p$.
- Publishes P_a.

Encryption Algorithm

Bob will encrypt a message m using Alice's public key P_a and then send to Alice. He performs the encryption algorithm as follows:

- Chooses a random r from the interval $[1, p-1]$ (r is an ephemeral secret key and is generated for every message).
- Computes $c_1 = g^r \ (mod \ p)$.
- Computes $s = {P_a}^r \ (mod \ p)$.
- Converts the secret message m into an element m' of G.
- Computes $c_2 = m' \cdot s \ (mod \ p)$.
- Sends the ciphertext $(c_1, \ c_2)$ to Alice.

If m' is compromised, then $s = {P_a}^r$ can easily be found. This is the reason of generating a new r for every message.

Decryption Algorithm

Alice will decrypt the cipher text $(c_1, \ c_2)$ using her private key a. She performs the decryption algorithm as follows:

- Computes the shared secret $s = {C_1}^a \ (mod \ p)$ (Typical Diffie-Hellman key exchange (Diffie and Hellman (1976)).
- Computes $m' = c_2 \cdot s^{-1}$ which she then converts back to the plain text m.

In this study, ElGamal Encryption Scheme is performed on implementation of the second protocol; *Key Distribution Protocol using Verifiable Secret Sharing* in order to make the protocol applied over open channel by encrypting the communication

messages between protocol entities (see Section 4.3.4 for implementation). (Actually, the communication messages are common secret pieces and they are sent to recipient entities via secure channel in the original protocol, see Section 4.3.2).

3.1.4. Boneh-Franklin ID-Based Scheme

This part is a literature review of ID-based cryptography. In this study, PKI is applied for the secure communication during the protocol.

In Section 3.1.1, to achieve assurance of *public-key authenticity* on a large scale, *public keys* are generally distributed in the form of certificates. Although the notion of a certificate is very simple, there are many practical difficulties with managing certificates. For example, a party named Alice may not know how to obtain another party named Bob's certificate. Also, Alice should have the assurance that Bob's public key is still valid (could be expired or compromised).

In 1984, Shamir (Shamir (1984)) introduced the notion of ID-based cryptography to handle problems related to certificate management. According to Shamir's scheme; Alice's public key consists of her identifying information ID_A (such as Alice's e-mail address). A trusted third party (TTP) would use its private key to generate Alice's private key from ID_A and transmit it via a secure channel to Alice. Bob could encrypt messages for Alice using only ID_A and the TTP's public key. Unlike the case with traditional certificate-based encryption schemes, Bob can encrypt a message for Alice even before Alice has generated a key pair. Bob could include in ID_A any set of conditions that should be met before the TTP issues the private key.

Shamir's notion was an open problem till 2001. In 2001, Boneh and Franklin (Boneh and Franklin (2001)) proposed the first practical ID-Based encryption scheme using a bilinear pairing e on (G_p, G_T) for which the BDH assumption in Section 2.1.3 is intractable.

Setup

- Entities named Alice and Bob.
- A prime p.

- A bilinear map (assume Weil pairing) $e: G_p \times G_p \rightarrow G_T$ between two groups G_p (additive group) and G_T (multiplicative group) as long as a variant of the computational Diffie-Hellman assumption is hard on G_p,
- A generator P of this additive group G_p.
- Two hash functions $H_1: \{0,1\}^* \rightarrow G_p/\{\infty\}$ } and $H_3 : G_T \rightarrow \{0,1\}^l$, where l is the bitlength of the plaintext.

Private Key Creation

The TTP selects its master private key $s \in [1, p-1]$, and generates its public key is $P_{TTP} = sP$.

All parties are able to obtain an authentic copy of P_{TTP}.

When Alice requests her private key d_A, the TTP creates Alice's identity string ID_A, computes $d_A = sH_1(ID_A)$, and securely delivers d_A to Alice. d_A is TTP's *BLS signature* on the message ID_A.

Encryption

Bob computes $Q_A = H_1(ID_A)$, selects private key $b \in [1, p-1]$, and computes public key $P_{Bob} = bP$. Then $c = m \oplus H_2(e(Q_A, P_{TTP})^b)$. Finally he transmits (P_{Bob}, c) to , Alice.

Decryption

Alice uses her private key d_A to compute $m = c \oplus H_2(e(d_A, P_{Bob}))$. Decryption works because $e(d_A, P_{Bob}) = e(sQ_A, bP) = e(Q_A, sP)^b = e(Q_A, P_{TTP})^b$.

An eavesdropper who wishes to recover m from (P_{Bob}, c) must compute $e(Q_A, P_{TTP})^b$ given $(P, Q_A, P_{TTP}, P_{Bob})$; this is precisely an instance of the BDH assumption in Section 2.1.3. But, the scheme is not resistant to chosen-ciphertext attacks (Bleichenbacher (1998)). Given a target ciphertext (P_{Bob}, c), the attacker can simply flip the first bit of c to get c', and thereafter obtain the decryption m' of the modified ciphertext (P_{Bob}, c'). She then flips the first bit of m' to recover m. In addition to H_1 and H_2, two hash function $H_3 : \{0,1\}^* \rightarrow [1, p-1]$ and $H_4 : \{0,1\}^l \rightarrow \{0,1\}^l$ are employed in order to troubleshoot this attack. Then, to encrypt m, Bob randomly selects a bitstring $\sigma \in \{0,1\}^l$ and computes $g = e(Q_A, P_{TTP}), b = H_3(\sigma, m), P_{Bob} =$

$bP, c_1 = \sigma \oplus H_2(g^b)$, and $c_2 = m \oplus H_4(\sigma)$. The ciphertext is (P_{Bob}, c_1, c_2). To decrypt, Alice computes $g^b = e(d_A, P_{Bob}), \sigma = c_1 \oplus H_2(g^b), m = c_2 \oplus H_4(\sigma)$, and $b = H_3(\sigma, m)$. Alice accepts the plaintext m provided that $P_{Bob} = bP$.

As public keys are derived from identifiers, ID-Based encryption (IBE) eliminates the need for a public key distribution infrastructure. The authenticity of the public keys is guaranteed implicitly as long as the transport of the private keys to the corresponding entity is kept secure (*Authenticity, Confidentiality, Integrity*).

On the other hand, TTP generates private keys for entities, it may decrypt and/or sign any message without authorization. This implies that IBE systems cannot be used for *non-repudiation*. The issue of *key escrow* does not exist with the current PKI system wherein private keys are usually generated on the entity's computer; but in IBE, TTP can decrypt all of the entity's messages passively since it computes private keys of entities from its master secret.

ID-based infrastructure fits two party key establishment protocols as secure communication methodology. Especially, key establishment on Email protocols, Voice over Internet Protocol (VoIP) and Session Initiation Protocol (SIP) use ID-based cryptography. However, in a group oriented case like this study, it is the best choice to apply a PKI.

3.2. Key Agreement using Bilinear Pairings

In this part a literature review of Diffie-Hellman Key Exchange (Diffie and Hellman (1976)) and Joux's one round, tripartite protocol (Joux (2000)) are examined in order to gain insight on *Key Agreement Protocol using Bilinear Pairing*.

3.2.1. Diffie-Hellman Key Exchange

This seminal works developed in 1976 by Whitfield Diffie and Martin Hellman and published in "New Directions in Cryptography" (Diffie and Hellman (1976)). The protocol allows two parties to jointly establish a secret key over an insecure channel without any prior knowledge of each other.

The Diffie-Hellman protocol relies on the difficulty of solving discrete logarithms (*DLog*) in finite fields and the related intractability of the *Computational*

Diffie-Hellman Assumption (see Section 2.1.3). Due to the difficulty of solving these mathematical problems, an eavesdropper (passive adversary defined in Section 2.2) is unable to compute efficiently the secret key with any or all of the information intercepted in the open communication channel. Once the secret key has been exchanged successfully between the two parties, the key can be used to encrypt confidential communications between them using a symmetric key cipher.

Main drawback of this seminal work is lack of authentication. This results that an active adversary defined in Section 2.2.1 can compromise the communication of legitimate parties using method of man in the middle attack (Menezes et al. (1996)).

Diffie-Hellman Key Agreement Protocol:

Setup

Two parties wish to communicate securely named Alice and Bob. They agree upon public parameters a prime p and a generator g of Z_p^* $(1 < g < p-1)$.

Exchange (Round-1)

- Alice chooses a random a from $(1 < a < p-1)$, then computes $A \equiv g^a \ (mod\ p)$ and sends A to Bob.
- Bob chooses a random b from $(1 < b < p-1)$, then computes $B \equiv g^b \ (mod\ p)$ and sends B to Alice.

Key Agreement

- Bob computes a key $K \equiv A^b \equiv (g^a)^b \ (mod\ p)$.
- Alice computes a key $K \equiv B^a \equiv (g^b)^a \ (mod\ p)$.
- Now, both Alice and Bob have the same secret key $\equiv g^{ab} \ (mod\ p)$.

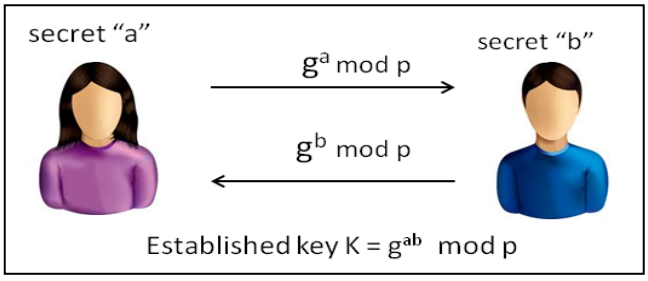

Figure 3.4. Diffie-Hellman Key Agreement.

Figure 3.4 presents the *exchange* and *key agreement phases*.

Table 3.1. Diffie-Hellman Key Agreement Example.

Setup	***Public Parameter Creation***	
	A large prime $p = 37$ and an integer $g = 2$ having large prime order in Z_p^*	
Exchange (Round 1)	***Private Computations***	
	Alice	***Bob***
	Choose a secret $a = 14$ Compute $30 \equiv 2^{14}\ (mod\ 37)$.	Choose a secret $b = 23$ Compute $5 \equiv 2^{23}\ (mod\ 37)$.
	Public Exchange of Values	
	Alice sends 30 to Bob $\rightarrow$ 30 5 $\leftarrow$ *Bob* sends 5 to Alice	
Key Agreement	***Further Private Computations***	
	Alice	***Bob***
	Compute the number $5^{14}\ (mod\ 37)$.	Compute the number $30^{23}\ (mod\ 37)$.
	Established common secret value $5^{14} \equiv (2^{23})^{14} \equiv 2^{14 \cdot 23} \equiv (2^{14})^{23} \equiv 30^{23}\ (mod\ 37)$.	

A computational example is available in Table 3.1.

3.2.2. Joux's One Round Protocol for Tripartite Diffie-Hellman

After Diffie-Hellman seminal work, research interests evolved to multiparty (group) concepts. The Diffie-Hellman protocol can be viewed as a one-round protocol because the two exchanged messages are independent of each other. The protocol can easily be extended to three parties, but an extra round is needed depending on participant number as seen in Figure 3.5. Therefore, round minimization concerns occurred with these interests. A natural question to ask is whether there exists a three-party one-round key agreement protocol that is secure against eavesdroppers. This question remained open until 2000 when Joux (Joux (2000)) devised a surprisingly simple protocol that used bilinear pairings. Joux's paper named "A One Round Protocol

for Tripartite Diffie-Hellman" was of great interest to cryptographers, who started investigating further applications of pairings and become a basis for construction of multiparty key agreement protocols which are based on *bilinear pairing approach*.

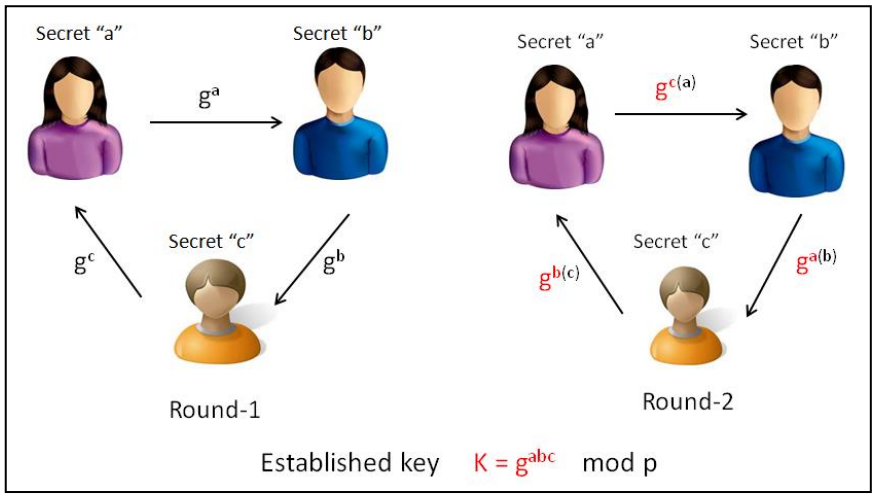

Figure 3.5. Three-Party Two-Round Key Agreement Protocol based on Diffie-Hellman.

Joux's three participants' variation of Diffie-Hellman protocol is based on the Weil and Tate pairings on elliptic curves, which were first used in cryptography as cryptanalytic tools for reducing the discrete logarithm problem on some elliptic curves to the discrete logarithm problem in a finite field. Its security depends on bilinear Diffie-Hellman assumption (see Section 2.1.3). However, same as Diffie-Hellman Key Exchange; although it is secure against passive adversary defined in Section 2.2.1, it suffers man in the middle attack since lack of authentication of parties.

One Round Protocol for Tripartite Diffie-Hellman:

Setup

Three parties wish to communicate securely named Alice, Bob and Chris. They agree upon public parameters which are;

- Prime p,
- Bilinear map (assume Weil pairing) $e: G_p \times G_p \rightarrow G_T$ between two groups G_p (additive group) and G_T. (multiplicative group) as long as a variant of the computational Diffie-Hellman assumption is hard on G_p,

- Generator P of this additive group G_p.

Exchange (Round-1)

- Alice chooses a random a from $(1 < a < p-1)$, then computes $A \equiv aP \pmod{p}$ and sends A to Bob and Chris.
- Bob chooses a random b from $(1 < b < p-1)$, then computes $B \equiv bP \pmod{p}$ and sends B to Alice and Chris.
- Chris chooses a random c from $(1 < c < p-1)$, then computes $C \equiv cP \pmod{p}$ and sends C to Alice and Bob.

Key Agreement

- Alice computes a key $K \equiv e(bP, cP)^a \pmod{p}$.
- Bob computes a key $K \equiv e(aP, cP)^b \pmod{p}$.
- Chris computes a key $K \equiv e(aP, bP)^c \pmod{p}$.
- Now, all of them have the same secret key $K \equiv e(P, P)^{abc} \pmod{p}$ (see Figure 3.6).

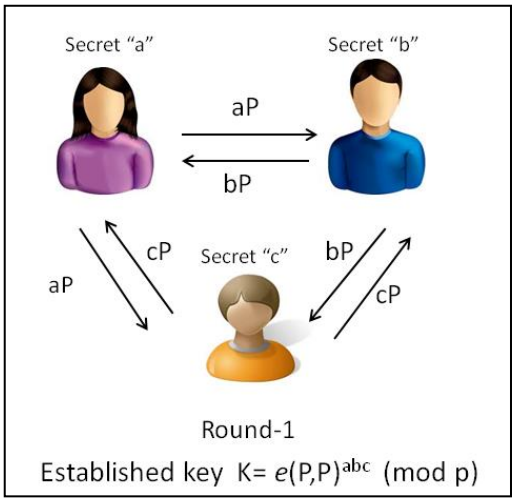

Figure 3.6. Joux's One Round, Tripartite Protocol.

3.3. Key Distribution using Verifiable Secret Sharing

In this part a literature review of Shamir's secret sharing (Shamir (1979)) and Feldman's verifiable secret sharing (Feldman (1987)) are examined in order to gain insight on *Key Distribution Protocol using Verifiable Secret Sharing*.

3.3.1. Shamir's Secret Sharing

Shamir's (k, n) *secret sharing scheme* (Shamir (1979)) divides a *secret* s into n *shares* by a dealer P_d and distributes them among n shareholders $P = \{P_1, P_2, \ldots, P_n\}$ in such a way that at least k *shares* are required to reconstruct the *secret* s and less than k *shares* gain no information about the secret.

It tackles the single point of failure. For example, the most secure key management schemes keep the key in a single, well-guarded location. But storing multiple copies of the key at different locations prevents the danger of compromising of that single location. By using a (k, n) threshold scheme (Shamir (1979)) with $n = 2k - 1$, very robust key management schemes can be constructed: the original key can be recovered even when $\lceil n/2 \rceil = k - 1$ of the n pieces are destroyed, but opponents cannot reconstruct the key even when security breaches expose $\lceil n/2 \rceil = k - 1$ of the remaining k pieces.

Shamir's (k, n) Secret Sharing Scheme

It consists of *sharing phase* which is based on $(k-1)^{th}$ degree polynomial and *reconstruction phase* which is based on Lagrange interpolating polynomial of at least k private shares (see Section 2.1.1).

- Let secret $s \in F$ for some finite field F that is Z_p for some suitable prime p (which is known to all the shareholders (parties) and the dealer as well).
- Shamir's (k, n) scheme is for n parties $(P_1, P_2, \ldots, P_n)$ with dealer P_d, where k is threshold and $n < |F|$.

Sharing Phase

- Dealer randomly chooses a $f(x)$ polynomial of degree $k-1$ such that; $f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{k-1}x^{k-1}$, in which the secret $s = a_0$ and all coefficients are $a_1, \ldots, a_{k-1} \in F$.
- Then, sends the shares $s_i = f(i)\ (mod\ p)$ to each P_i where $i = 1, \ldots, n$ (i value is a known value and the secret $s = f(0)$) by using of security channel.

Reconstruction Phase

- Any subset of k or more shares can be used to reconstruct the secret s. Without loss of generosity, the subset is : $f(1), f(2), \ldots, f(k)$.
- Lagrange interpolating formula is used to find the polynomial $f(x)$, such that degree of $f(x)$ is $k-1$ and any k or more $(i, f(i))$ points for $i = 1, 2, \ldots, n$ will be sufficient to reconstruction (The reconstructed secret must be $f(0)$).

Given any k pairs of $(i, f\ (i))$, with distinct i values, there is a unique polynomial $f(x)$ of degree $k-1$, passing through all these points. This polynomial can be effectively computed from the pairs $(i, f\ (i))$ (see Table 3.2).

Table 3.2. Lagrange's Interpolation Formula.

	$L_i\ (x)$	$f_i\ (x)$
The polynomial $f(x)$ can be recovered as;	$L_i(x) = \prod_{\substack{i=1 \\ i \neq j}}^{k} \frac{x - x_j}{x_i - x_j}$	$f(x) = \sum_{i=1}^{t} f(i) \cdot L_i(x)$ where $L_i\ (x)$ is Lagrange interpolation polynomial.
The secret s can be recovered as;	$L_i(0) = \prod_{\substack{i=1 \\ i \neq j}}^{k} \frac{x_j}{x_j - x_i}$	$f(0) = \sum_{i=1}^{t} f(i) \cdot L_i(x)$ where $L_i\ (0)$ is Lagrange coefficient.

Example: Let $(k, n) = (3, 5)\ mod\ 7$; for $P_i = \{P_1, P_2, \ldots, P_5\}$.

Sharing Phase

Dealer;

- Chooses $f(x) = 3x^2 + 2x + 5\ (mod\ 7)$ where secret is $f(0) = 5$.

- Calculates shares *of each shareholder*

  $f(1) = 10 \equiv 3 \ mod \ 7$ sends to P_1 as point (1,3)

  $f(2) = 21 \equiv 0 \ mod \ 7$ sends to P_2 as point (2,0)

  $f(3) = 38 \equiv 3 \ mod \ 7$ sends to P_3 as point (3,3)

  $f(4) = 61 \equiv 5 \ mod \ 7$ sends to P_4 as point (4,5)

  $f(5) = 90 \equiv 6 \ mod \ 7$ sends to P_5 as point (5,6)
- Sends each share $(i, f(i))$ to corresponding node P_i.

Reconstruction Phase

In order to reconstruct the secret, interpolation of any 3 points will be enough. Assume that $P_1 = (1,3)$, $P_2 = (2,0)$, $P_4 = (4,5)$ get together and reveal their shares to each other in order to reconstruct the secret $s = \mathbf{5}$ (see Table 3.3).

Table 3.3. Example of Lagrange's Interpolation Formula.

	$L_i\ (x)\ (mod\ 7)$	$f_i\ (x)\ (mod\ 7)$
The polynomial $f(x)$ can be recovered as;	$L_1(x) = \frac{x-2}{1-2} \cdot \frac{x-4}{1-4}$ $L_2(x) = \frac{x-1}{2-1} \cdot \frac{x-4}{2-4}$ $L_3(x) = \frac{x-1}{4-1} \cdot \frac{x-2}{4-2}$	$f(x) =$ $3 \cdot L_1(x) + 0 \cdot L_2(x) + 5 \cdot L_3(x)$ $f(x) =$ $3x^2 + 2x + 5$
The secret s can be recovered as;	$L_1(0) = \frac{2}{2-1} \cdot \frac{4}{4-1} = \frac{8}{3}$ $L_2(0) = \frac{1}{1-2} \cdot \frac{4}{4-2} = -2$ $L_3(0) = \frac{1}{1-4} \cdot \frac{2}{2-4} = \frac{1}{3}$	$f(0) =$ $3 \cdot \frac{8}{3} + 0 \cdot (-2) + 5 \cdot \frac{1}{3} =$ $\mathbf{5}$

Drawback of Shamir's protocol is if some party P_i is malicious, then it can input a fake share to the reconstruction and thus the other honest parties get nothing but a faked secret. More importantly, the dealer may misbehave and sends deviated shares to the participants. For the fair reconstruction of the secret, cheater detection (dealer or participants) and identification (for secure communication) are very essential. However,

Shamir's secret sharing scheme doesn't prevent malicious behavior of dishonest shareholders (Chor et al. (1985)). Verifiable Secret Sharing Scheme provides detection of malicious entities in the protocol of Secret Sharing.

3.3.2. Feldman's Verifiable Secret Sharing

Shamir's (k, n) Secret Sharing Scheme (Shamir (1984)) assumed that the dealer is reliable. But in reality the dealer may misbehave and can deal inconsistent shares to the entities. Also, the malicious entities can input fake shares during the reconstruction phase. Thus, any k *(threshold)* participants or more which are pooled to reconstruct the secret will be unable to establish the secret value. Verifiable secret sharing scheme (VSS) addresses this issue.

The notion of VSS was first introduced by Chor et al. (Chor et al. (1985)) to the original secret sharing scheme. Using the verifiable property, shares are verifiable via commitments without having the idea of what the secret is; and cheating parties (participants or dealer) can be detected.

Feldman (Feldman (1987)) has proposed a non-interactive scheme for achieving verifiability in Shamir's scheme (see section 3.3.1) utilizing *homomorphic commitments* (see Section 2.1.4). Feldman's scheme is exactly the Shamir's protocol. The novelty is that dealer commits the coefficients of polynomial $f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{k-1}x^{k-1}$ which is $C = (C_0 = \alpha^{a_0}, C_1 = \alpha^{a_1}, .., C_{k-1} = \alpha^{a_{k-1}})$, where α is generator. Feldman's commitments are unconditionally binding and computationally hiding under the assumption DLog assumption (see Section 2.1.3). In order to open the commitment, dealer reveals evaluations of the polynomial $f(i)$ for each corresponding party P_i. Each party can easily confirm that an opening $f(i)$ for index i is consistent with C.

Feldman's VSS Scheme

It consists of *sharing phase* which is based on $(k-1)^{th}$ degree polynomial and *reconstruction phase* which is based on Lagrange interpolating polynomial of at least k private shares.

- There are public parameters which are odd primes such that $q|p-1$, and $\alpha \in Z_p^*$ is an element of order q.

- Feldman's (k, n) VSS scheme is for n parties $(P_1, P_2, \ldots, P_n)$ with dealer P_d, where k is threshold and secret s; n, k, and $s \in Z_q$.

Sharing Phase

Dealer;

- Chooses randomly a $f(x)$ polynomial of degree $k-1$ such that; $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1}$, in which the secret $s = a_0$ and all coefficients a_j are $a_1, \ldots, a_{k-1} \in Z_q$.
- Constructs the commitment vector based on (DLog commitments) such that; $C = (C_0 = \alpha^{a_0}, C_1 = \alpha^{a_1}, .., C_{k-1} = \alpha^{a_{k-1}}) \ (mod\ p)$.
- Sends in a *secure channel* the shares $s_i = f(i) \ (mod\ q)$ to each P_i where $i = 1, \ldots, n$ (the secret $s = f(0)$) and the commitment vector $(C_0 = \alpha^{a_0}, C_1 = \alpha^{a_1}, .., C_{k-1} = \alpha^{a_{k-1}}) \ (mod\ p)$.

Reconstruction Phase

- Each user verify validity of received share $f(i)$ by testing;

$$\alpha^{f(i)} (mod\ p) = \prod_{j=0}^{k-1} \mathrm{C}_j^{i^j} \ (mod\ p) \text{ where } Cj = \alpha^{a_j}$$

and; i $(i = 1, .., n)$ is indices of participants and j $(j = 0, \ldots, k-1)$ is indices of polynomial coefficients.

- If verification is consistent, any subset of k or more shares can be used to reconstruct the secret s.
- Lagrange interpolating formula is used to find the polynomial $f(x)$, such that degree of $f(x)$ is $k-1$ and any k or more $(i, f(i))$ points for $i = 1,2, \ldots, n$ will be sufficient to reconstruction (The reconstructed secret must be $f(0)$).

***Example*:**

Let $(k, n) = (3, 5)$; for $P_i = \{P_1, P_2, \ldots, P_5\}$ and let $q|(p-1)$, $p = 23$ and $q = 11$ where $\alpha = 3$ is a generator of 11.

Sharing Phase

Dealer;

- Chooses $f(x) = 3x^2 + 2x + 5\ (mod\ 11)$ where secret is $f(0) = 5$.
- Construct commitment vector such that;

  $C\ =\ [C_0 = 3^{a_0}, C_1 = 3^{a_1},\ C_2 = 3^{a_2}]$

  $=\ [C_0 = 3^5, C_1 = 3^2,\ C_2 = 3^3]\quad (mod\ 23)$.
- Calculates shares *of each shareholder (or party)* such that;

  $f(1) \equiv 10\ mod\ 11$ sends to P_1 as point (1,10)

  $f(2) \equiv 10\ mod\ 11$ sends to P_2 as point (2,10)

  $f(3) \equiv 5\ mod\ 11$ sends to P_3 as point (3,5)

  $f(4) \equiv 6\ mod\ 11$ sends to P_4 as point (4,6)

  $f(5) \equiv 2\ mod\ 11$ sends to P_5 as point (5,2)
- Sends in a secure channel the shares $P_1 = (1,10), P_2 = (2,10), P_3 = (3,5), P_4 = (4,6), P_5 = (5,2)\ (mod\ 11)$ to every corresponding party P_i accompanied with the commitment vector $C\ =\ [C_0 = 3^5, C_1 = 3^2,\ C_2 = 3^3]\ (mod\ 23)$.

Reconstruction Phase

Each party verifies the correctness of receives share such as;

For $P_1 = (1,10)$

- Since his share is 10, he needs to have equality with $3^{10}\ =\ \mathbf{8}\ (mod\ 23)$.
- From the commitment vector $C\ =\ [C_0 = 3^5, C_1 = 3^2,\ C_2 = 3^3]$, his calculation for verification; $(3^5)^{1^0} \cdot (3^2)^{1^1} \cdot (3^3)^{1^2} =\ \mathbf{8}\ (mod\ 23)$.

For $P_2 = (2,10)$

- Since his share is 10, he needs to have equality with $3^{10}\ =\ \mathbf{8}\ (mod\ 23)$.
- From the commitment vector $C\ =\ [C_0 = 3^5, C_1 = 3^2,\ C_2 = 3^3]$, his calculation for verification; $(3^5)^{2^0} \cdot (3^2)^{2^1} \cdot (3^3)^{2^2} =\ \mathbf{8}\ (mod\ 23)$.

For $P_3 = (3,5)$

- Since his share is 5, he needs to have equality with $3^5\ =\ \mathbf{13}\ (mod\ 23)$.
- From the commitment vector $C\ =\ [C_0 = 3^5, C_1 = 3^2,\ C_2 = 3^3]$, his calculation for verification; $(3^5)^{3^0} \cdot (3^2)^{3^1} \cdot (3^3)^{3^2} =\ \mathbf{13}\ (mod\ 23)$.

For $P_4 = (4,6)$

- Since his share is 6, he needs to have equality with $3^6\ =\ \mathbf{16}\ (mod\ 23)$.
- From the commitment vector $C\ =\ [C_0 = 3^5, C_1 = 3^2,\ C_2 = 3^3]$, his calculation for verification; $(3^5)^{4^0} \cdot (3^2)^{4^1} \cdot (3^3)^{4^2} =\ \mathbf{16}\ (mod\ 23)$.

For $P_5 = (5,2)$

- Since his share is 2, he needs to have equality with $3^2 = \mathbf{9}\ (mod\ 23)$.
- From the commitment vector $C = [C_0 = 3^5, C_1 = 3^2,\ C_2 = 3^3]$, his calculation for verification; $(3^5)^{5^0} \cdot (3^2)^{5^1} \cdot (3^3)^{5^2} = \mathbf{9}\ (mod\ 23)$.

If verification is consistent, any 3 points is enough to reconstruct the secret s.

Let consider $P_1 = (1,10)$, $P_3 = (3,5)$ and $P_5 = (5,2)$ get together and reveals their shares to each other in order to reconstruct the secret $s = \mathbf{5}$ using Lagrange interpolation for $mod\ 11$ (see Table 3.4).

Table 3.4. Example of Lagrange's Interpolation Formula.

	$L_i\ (x)\ (mod\ 11)$	$f_i(x)\ (mod\ 11)$
The polynomial $f(x)$ can be recovered as;	$L_1(x) = \frac{x-3}{1-3} \cdot \frac{x-3}{1-3}$ $L_2(x) = \frac{x-1}{3-1} \cdot \frac{x-5}{3-5}$ $L_3(x) = \frac{x-1}{5-1} \cdot \frac{x-3}{5-3}$	$f(x) =$ $10 \cdot L_1(x) + 5 \cdot L_2(x) + 2 \cdot L_3(x)$ $f(x) =$ $3x^2 + 2x + 5$
The secret s can be recovered as;	$L_1(0) = \frac{3}{3-1} \cdot \frac{5}{5-1} = \frac{15}{8}$ $L_2(0) = \frac{1}{1-3} \cdot \frac{5}{5-3} = -\frac{5}{4}$ $L_3(0) = \frac{1}{1-5} \cdot \frac{3}{3-5} = \frac{3}{8}$	$f(0) =$ $10 \cdot \frac{15}{8} + 5 \cdot \left(-\frac{5}{4}\right) + 2 \cdot \frac{3}{8} =$ $\mathbf{5}$

Feldman's work provides to detect the malicious party. But the protocol needs identification for secure communication between communicating parties.

Feldman's VSS is basis for the second protocol of the thesis.

CHAPTER 4

GROUP KEY ESTABLISHMENT PROTOCOLS

There are two methods to *establish a common secret key* by communicating parties over an open network: key agreement and key distribution (Stinson (1995)).

- A key agreement protocol; is a mechanism whereby the parties jointly establish a common secret key.
- A key distribution protocol; is a mechanism whereby one party creates or obtains a secret value and then securely transfers it to other parties.

This chapter describes the *construction* of group key establishment protocols for each of two methods which of them is based on *key agreement using bilinear pairings* and another is based on *key distribution using verifiable secret sharing*; along with their *communication models*, *security analysis* and *implementation results* for four users.

Contribution:

Group Key Agreement Protocol: Its construction and security analysis depends on Lin et al.'s work (Lin et al. (2006)) and this thesis's contributions are:

- A signature algorithm is added in order to provide message integrity on each public message which is broadcasted during the two rounds. The original work doesn't include message authentication mechanism, instead of this mechanism prefers to use dedicated secure channel.
- An open security problem is explored for this protocol and will be presented in the Section 4.2.2 and analyze it in the Section 4.2.3.
- An implementation of this protocol is realized for four users in order to present how many milliseconds computation and how many bit-lengths communication of each user on per round are performed.
- Detailed efficiency analysis is studied in terms of computation, communication and round complexity with and without thesis's contribution.

Group Key Distribution Protocol: Its construction depends on Feldman's work (Feldman (1987)) and this thesis's contributions are:

- Original protocol is adapted a secure and closed group communication by adding certificates for the participants.
- The protocol is rendered to be applied over open-unsecure channel by adding ElGamal encryption algorithm for sending messages of each user.
- ElGamal signature algorithm on publicly known commitments is added in order to prevent modifications by malicious parties. Signature algorithm satisfies the message integrity and identification of sender.
- Un-formal security analysis in terms of security attributes defined in section 2.2.3 is presented.
- An implementation is proposed for four users one of them is a leader in order to present how many milliseconds computation and how many bit-lengths communication of each user on per round are performed.
- Detailed efficiency analysis is studied with and without thesis's contribution in terms of computation, communication and round complexity and its comparison with the previous protocol is evaluated.

4.1. Specifications and Assumptions

There are some assumptions and specifications defined by *this study* to explain and evaluate both key establishment methods and selected instance protocols. These are:

- Both protocols are working over open channel forming a secure, closed communication group for small number of participants (such as in range $3 \leq n \leq 100$ where n is participant number) so that the members of these group can be controlled on peer by peer.

- A participant is called *reliable*, if they satisfy the following properties:
    - They do not leak secret information to outside the group.
    - They send the correct messages defined by the protocol.
    - At the start of the protocol, assume that all participants are registered by a legitimate Certification Authority (CA) via Public Key Infrastructure (PKI) as explained in Section 3.1.1, and any time one of the certificates of participants can be easily checked from the list of CA. Before the

protocol start, certificate of users have already exchanged. Therefore; the basic assumptions of this study about the communication between reliable participants U_A and U_B are that:

- When U_A sends a message to U_B, nobody else can learn anything about its content, only receiver U_B can open it; this should be satisfied by cryptographic structures such as discrete logarithm problem.
- When U_B receives a message from U_A, U_B can be certain that nobody but U_A could have sent the message; this should be satisfied by signature schemes.

- Communication messages sent will be received in a timely manner.

- All participants agree on the protocols to be followed.

4.2. Group Key Agreement Protocol using Bilinear Pairing

Bilinear pairing is a hot topic in the last decade since it provides construction of ingenious protocols such tasks as key agreement (Joux (2000)), ID-based encryption (Boneh and Franklin (2001)), signature scheme (Boneh et al. (2004)), etc.

In multiparty key agreement protocols, *bilinear pairing* serves round minimizing solution. The first protocol model of the thesis which is based on bilinear pairing approach is constructed on the basis of Lin et al.'s work (Lin et al. (2006)).

4.2.1. Preliminaries and Communication Model of Key Agreement Protocol

This protocol is defined under the specifications and assumptions which are mentioned in Section 4.1. The communicating participants need only two constant rounds of public message transmissions with equal contributory of each participant. Also, number of communicating parties is independent of number of communication rounds. Besides, the message size, the total number of scalar multiplications, and number of Weil pairing is reduced as explained by the study of Lin et al. (2006).

For each run of the protocol one among n parties works as a leader L. The leader L starts the protocol specifying the group number. Group number is static and allocation of each party is fixed forming a ring structure during all rounds. Addition or removal of a party or parties is possible only when the new protocol run. In each run of the protocol new common key is established. Therefore, new joining participants are not able to become involved the old established key.

In order to provide secure communication (Section 3.1) between the participants, a PKI (Section 3.1.1) is applied. Before the protocol run, each legitimate party obtains his/her certificate from CA. Security of the protocol depends on the assumption of Bilinear Diffie-Hellman problem (Section 2.1.3) is hard and meets the security attributes of key agreement protocol defined in Section 2.2.1.

4.2.2. Construction of Key Agreement Protocol

Notations used in the Group Key Agreement Protocol as the following;

- n parties wish to agree on a common secret key.
- U_i is a participant in a communication round where the participant set $U = \{U_1, U_2, \dots, U_n\}$.
- p is an integer prime number.
- A bilinear map (Weil pairing) $e: G_p \times G_p \rightarrow G_T$ between two groups G_p (additive group) and G_T (multiplicative group) as long as a variant of the Computational Diffie-Hellman Assumption is hard on G_p. (Section 2.1.3).
- P is the generator of additive group G_p.
- a_i is the long term private key randomly chosen by U_i.
- Y_i is the long term public key computed $Y_i = a_i P$ by the related party U_i.
- $Cert_i$ is U_i's long term public key certificate which is signed by CA.
- x_i is the short term (ephemeral) secret key randomly chosen by U_i.
- T_i and X_i are U_i's public messages in each communication round.
- The public parameters are $param = \langle G_p, G_T, e, p, P \rangle$.

Setup

Before the protocol is began ; each party U_i generates his/her static public key $Y_i = a_i P$ where a_i is random number used as the long term private key selected by U_i

and applies a certificate authority (CA) in order to obtain his/her certificate $Cert_i$ which contains static public key Y_i and an unique identifier string U_i (Section 3.1.1).

Every party U_i exchange $Cert_i$ between themselves and verifies received $Cert_i$ with CA. Hence all group users have been identified.

Exchange (Round 1)

The leader L from set U announces the public parameters $param = \langle G_p, G_T, e, p, P, H \rangle$ and the group number n. Then the first round starts.

- Each user $U_i, i = 1, \ldots, n$, chooses a random secret number x_i,
- Computes $T_i = x_i Y_i = x_i(a_i P)$.
- Signs T_i in order to protect message integrity and assure authenticity. Such that; $Sig_i(T_i)$, where Sig_i is a cryptographic signature algorithm (BLS Short Signature is used in the implementation, see Section 4.2.4 and Appendix A).
- Broadcasts T_i with $Sig_i(T_i)$.

Exchange (Round 2)

- Each $U_i, i = 1, \ldots, n$, verifies $Sig_i(T_i)$ using sender's public key.
- Computes
  - $X_i = e((Y_{i+1} + T_{i+1}), (Y_{i+2} + T_{i+2}) - (Y_{i-1} + T_{i-1}))^{(a_i + a_i x_i)}$.
  - Signs X_i, such that $Sig_i(X_i)$.
- Broadcasts X_i with $Sig_i(X_i)$.

Key Agreement (Round 2)

- Each $U_i, i = 1, \ldots, n$, verifies $Sig_i(X_i)$ using sender's public key.
- Computes
  - $K_i = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i + a_i x_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \ldots X_{i-2} =$
    $e(P, P)^{\left[\begin{matrix}(a_n + a_n x_n)(a_1 + a_1 x_1)(a_2 + a_2 x_2) + (a_1 + a_1 x_1)(a_2 + a_2 x_2)(a_3 + a_3 x_3) + \cdots \\ + (a_{n-1} + a_{n-1} x_{n-1})(a_n + a_n x_n)(a_{n+1} + a_{n+1} x_{n+1})\end{matrix}\right]}$

Furthermore, the common shared secret key is then obtained as; $K = kdf(K_1 \parallel U_1 \parallel U_2 \parallel \ldots \parallel U_n) = kdf(K_2 \parallel U_1 \parallel U_2 \parallel \ldots \parallel U_n) = \ldots = kdf(K_n \parallel U_1 \parallel U_2 \parallel \ldots \parallel U_n)$, where kdf is a *key derivation function algorithm* (i.e. KDF1, KDF2

(ISO/IEC 18033-2 (2006)), PBKDF1 (RSA Laboratories (2006)), etc.) and U_i is an unique identifier of participant U_i.

Simple summary of each phase for 10 group number is available in the Table 4.1. Implementation of the protocol for four users with 512-bit and 160-bit primes is performed in the Section 4.2.4 in order to present time measures of computation and bit-lengths of communication of each user on per round. Also, numeric values of the variable in the implementation can be followed under Appendix A.

Table 4.1. Group Key Agreement Protocol.

Setup	***Creation of Public Parameter***
	A large prime p, having large prime order in F_p^*, bilinear map $e: G_p \times G_p \rightarrow G_T$, a generator P of G_p.
	- There are $n = 10$ parties and representation of each U_i. - U_i applies a CA and exchanges $Cert_i$. - $Cert_i$contains static public key $Y_i = a_i P$ where a_i is the long term private key, an unique identifier string U_i and a signature of CA on this information.
Exchange (Round 1)	***Private Computations*** Each U_i $(i = 1, 2, \ldots, 10)$
	Step#1 Choose a short term secret x_i computes $T_i = x_i Y_i = x_i a_i P$. For $U_1: T_1 = x_1 Y_1$, $U_2: T_2 = x_2 Y_2, \ldots, U_{10}: T_{10} = x_{10} Y_{10}$ **Step#2 Contribution** to satisfy the identification of the users and integrity of the public messages; $Sig_i\ (T_i)$.
	Broadcasts T_i and $Sig_i\ (T_i)$.

(cont. on next page)

Table 4.1. (cont.)

	Private Computations Each U_i $(i = 1, 2, \ldots, 10)$
Exchange (Round-2)	**Step#3 Contribution** to satisfy the identification of the users and integrity of the public messages; Verifies $Sig_i\ (T_i)$. **Step#4** Computes $X_i = e((Y_{i+1} + T_{i+1}), (Y_{i+2}+T_{i+2}) - (Y_{i-1}+T_{i-1}))^{(a_i+a_i x_i)}$ $X_1 = e\big((Y_2 + T_2), (Y_3+T_3) - (Y_{10}+T_{10})\big)^{(a_1+a_1 x_1)}$ For U_1: $X_1 = e\big((Y_2 + T_2), (Y_3+T_3) - (Y_{10}+T_{10})\big)^{(a_1+a_1 x_1)}$. For U_2: $X_2 = e\big((Y_3 + T_3), (Y_4+T_4) - (Y_1+T_1)\big)^{(a_2+a_2 x_2)}$. . . For U_{10}: $X_{10} = e\big((Y_1 + T_1), (Y_2+T_2) - (Y_9+T_9)\big)^{(a_{10}+a_{10} x_{10})}$. **Step#5 Contribution** to satisfy the identification of the users and integrity of the public messages; signs X_i, such that $Sig_i\ (X_i)$.
	Broadcasts X_i and $Sig_i\ (X_i)$.
	Further Private Computations Each U_i $(i = 1, 2, \ldots, 10)$
Key Agreement (Round 2)	**Step#6 Contribution** to satisfy the identification of the users and integrity of the public messages; verifies $Sig_i\ (X_i)$. **Step #7** Computation of common key. $K_i = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i+a_i x_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \ldots X_{i-2} =$ $e(P,P)^{\left[\begin{matrix}(a_n+a_n x_n)(a_1+a_1 x_1)(a_2+a_2 x_2)+(a_1+a_1 x_1)(a_2+a_2 x_2)(a_3+a_3 x_3)+\cdots \\ +(a_{n-1}+a_{n-1} x_{n-1})(a_n+a_n x_n)(a_{n+1}+a_{n+1} x_{n+1})\end{matrix}\right]}$. For U_1:$K_1 = e\big((Y_2 + T_2), 10(Y_{10} + T_{10})\big)^{a_1+a_1 x_1} \cdot X_1^9 \cdot X_2^8 \ldots X_9 =$ $e(P,P)^{\left[\begin{matrix}(a_{10}+a_{10} x_{10})(a_1+a_1 x_1)(a_2+a_2 x_2)+(a_1+a_1 x_1)(a_2+a_2 x_2)(a_3+a_3 x_3)+\cdots \\ +(a_9+a_9 x_9)(a_{10}+a_{10} x_{10})(a_1+a_1 x_1)\end{matrix}\right]}$. For U_2:$K_2 = e\big((Y_3 + T_3), 10(Y_1 + T_1)\big)^{a_2+a_2 x_2} \cdot X_2^9 \cdot X_3^8 \ldots X_{10} =$ $e(P,P)^{\left[\begin{matrix}(a_{10}+a_{10} x_{10})(a_1+a_1 x_1)(a_2+a_2 x_2)+(a_1+a_1 x_1)(a_2+a_2 x_2)(a_3+a_3 x_3)+\cdots \\ +(a_9+a_9 x_9)(a_{10}+a_{10} x_{10})(a_1+a_1 x_1)\end{matrix}\right]}$. . . For U_{10}:$K_{10} = e\big((Y_1 + T_1), 10(Y_9 + T_9)\big)^{a_{10}+a_{10} x_{10}} \cdot X_{10}^9 \cdot X_1^8 \ldots X_8 =$ $e(P,P)^{\left[\begin{matrix}(a_{10}+a_{10} x_{10})(a_1+a_1 x_1)(a_2+a_2 x_2)+(a_1+a_1 x_1)(a_2+a_2 x_2)(a_3+a_3 x_3)+\cdots \\ +(a_9+a_9 x_9)(a_{10}+a_{10} x_{10})(a_1+a_1 x_1)\end{matrix}\right]}$.
	Established common secret value ; $K = kdf(K_1 \parallel U_1 \parallel U_2 \parallel \ldots \parallel U_{10}) = kdf(K_2 \parallel U_1 \parallel U_2 \parallel \ldots \parallel U_{10}) = \ldots = kdf(K_n \parallel U_1 \parallel U_2 \parallel \ldots \parallel U_{10})$.

Open Problem:

If any party establishes the wrong secret, he/she will realize the failure only when the communication with the common secret starts via symmetrical cryptography. He/She will encrypt messages with wrong common secret and the recipient cannot decrypt it. If any failure occurs with decryption, the recipient sends error message and the protocol restarts by picking new ephemeral secret key.

It is an *open problem* and the protocol needs a cryptographic tool which provides the verifiability. Therefore, users can be sure that the established secret is consistent before the communication with common secret starts.

4.2.3. Security Analysis of Key Agreement Protocol

Security is attributed in terms of preventing attacks of passive adversary and active adversary in Section 1.1. All parties of the protocol is performing same calculations during all rounds due to no party can predetermine the resulting value.

There are some inspections in order to prove security of the protocol in the presence of *passive* and *active adversary* (Lin et al. (2006)).

Inspections against passive adversary:

A *passive adversary* is a hidden listener who tries to compute the common secret key by listening to the channel in order to obtain communication messages between legitimate parties of the protocol. If the key agreement protocol gives no chance to deduce the secret values which generate public communication messages of each round, then the passive adversary cannot have any deduction from eavesdropped public messages about the established common secret key.

In order to prove this, the well known security assumption which is Bilinear Diffie-Hellman (BDH) Assumption (see Section 2.1.3) will be utilized.

A *passive adversary* cannot work under the assumption that solving BDH problem will be infeasible (Lin et al. (2006)). That is, given public values P, $T_1 = x_1P$, $T_2 = x_2P$, $T_3 = x_3P \in G_q$ ($x_1, x_2, x_3 \in Z_q$ are ephemeral secrets), the two tuples of random variables, $(T_1, T_2, T_3, e(P,P)^{x_1x_2x_3})$ and (T_1, T_2, T_3, T), where T is a random value in G_T, are computationally indistinguishable.

In other words there is no efficient algorithm A satisfying $|\Pr[A(x_1P, x_2P, x_3P, e(P,P)^{x_1x_2x_3}) = true]| - |\Pr[A(x_1P, x_2P, x_3P, T) = true]| > \frac{1}{Q(|q|)}$ for any polynomial Q, where the probability is over the random choice of x_1, x_2, x_3 and T.

In the light of this formal definition, assume that there is a *passive adversary E* who wants to calculate the established key K_i which is;

$K_i = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i + a_i x_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \dots X_{i-2}$.

Adversary E needs to compute the random value $T = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i + a_i x_i}$ which is equal to $e(P,P)^{n(a_{i-1} + a_{i-1}x_{i-1})(a_i + a_i x_i)(a_{i+1} + a_{i+1}x_{i+1}).}$ and $X = X_i^{n-1} \cdot X_{i+1}^{n-2} \dots X_{i-2}$, where $T, X \in G_T$ and then obtains $K_i = T \cdot X$. It can be assumed that adversary E can compute the value of X from the public messages X's. But, E cannot correctly deduce a_i and $a_i x_i$ from the public values. Therefore, adversary E cannot compute $T = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i + a_i x_i}$ since she doesn't know a_i and $a_i x_i$. She faces the hardness of BDH assumption for the pair groups G_q, G_T; means that;

There are randomly chosen secret values $a_{i-1}, a_i, a_{i+1}, x_{i-1}, x_i$ and x_{i+1}. Computing $T = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i + a_i x_i}$ by given public values $P, (a_{i-1} + a_{i-1}x_{i-1})P, (a_i + a_i x_i)P$ and $(a_{i+1} + a_{i+1}x_{i+1})P$ is computationally indistinguishable. Because of the two tuples of random variables,

$((Y_{i-1} + T_{i-1}), (Y_i + T_i), (Y_{i+1} + T_{i+1}), e(P,P)^{n(a_{i-1} + a_{i-1}x_{i-1})(a_i + a_i x_i)(a_{i+1} + a_{i+1}x_{i+1})})$

and $((Y_{i-1} + T_{i-1}), (Y_i + T_i), (Y_{i+1} + T_{i+1}), T)$ where T is a random value in G_T are computationally indistinguishable.

In other words, there is no efficient algorithm A satisfying,

$\big|\Pr\big[A\big((a_{i-1} + a_{i-1}x_{i-1})P, (a_i + a_i x_i)P, (a_{i+1} + a_{i+1}x_{i+1})P, e(P,P)^{n(a_{i-1} + a_{i-1}x_{i-1})(a_i + a_i x_i)(a_{i+1} + a_{i+1}x_{i+1})}\big) = true\big]\big| -$

$\big|\Pr\big[A\big((a_{i-1} + a_{i-1}x_{i-1})P, (a_i + a_i x_i)P, (a_{i+1} + a_{i+1}x_{i+1})P, T\big) = true\big]\big| > \frac{1}{Q(|q|)}$ for polynomial Q, where the probability is over the random chose of x_{i-1}, x_i, x_{i+1} and T. So, E cannot easily calculate the correct K_i.

Inspections against active adversary:

Active adversary tries to disrupt establishment of the common secret key among the legitimate parties. He mainly sends malicious messages into the broadcast channel to fool an honest participant into believing that this honest participant has computed the same common secret as that of other honest participants. Some security attributes are defined in Section 2.2.1. They are regarding of security against active adversary for authenticated key agreement protocols (Blake-Wilson et al. (1997)). Their inspections are as the following;

- *Known session key security*. A participant obtains a new ephemeral private key x_i to generate a unique session key (common secret) in each run of the key agreement protocol. Therefore, the knowledge of a previously established session key does not help anyone to deduce a new secret key.

- *Forward Secrecy*. Assume that an attacker has compromised one or more long term private keys a_i. However, the attacker cannot calculate the previously established session key $K_i = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i + a_i x_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \dots X_{i-2}$ without knowledge of the ephemeral private key x_i.

- *Key-compromise impersonation resilience*. Assume that, an attacker A has compromised U_1's static private key a_1, and wants to impersonate U_2 in order to deceive U_1 and establish the same secret with U_1.
  i. Firstly, attacker A pick a random u value and calculates a fake public message of round-1, which is $T_2{}' = uP$ (for honest U_2, $T_2 = a_2 x_2 P$) and broadcast $\{T_2^{'}, Cert_2\}$ by claiming that it is sent by U_2.
  ii. When U_1 computes the established secret, he will calculates

$$K_1 = e\big((Y_2 + T_2), n(Y_n + T_n)\big)^{a_1 + a_1 x_1} \cdot X_1^{n-1} \cdot X_2^{n-2} \dots X_{n-1}$$

$$= e(P, P)^{\left[\begin{matrix}(a_n + a_n x_n)(a_1 + a_1 x_1)(a_2 + \boldsymbol{u}) + (a_1 + a_1 x_1)(a_2 + a_2 x_2)(a_3 + a_3 x_3) + \cdots \\ + (a_{n-1} + a_{n-1} x_{n-1})(a_n + a_n x_n)(a_{n+1} + a_{n+1} x_{n+1})\end{matrix}\right]}$$

  iii. U_1 fails for the common secret which is established by other honest parties during the protocol and realizes the failure when he starts communication with wrong established secret and doesn't decrypt the received messages from the others (open problem of the protocol) , but

attacker A cannot compute the same key which is calculated by U_1 using the fake public message $T_2' = uP$ sent by attacker A. Such that, attacker A follows two ways:

(1) $K_2' = e\big((Y_3 + T_3), n(Y_1 + T_1)\big)^{a_2+u} \cdot X_2^{n-1} \cdot X_3^{n-2} \dots X_n$, but attacker A will fail for this way since he don't know correct value of U_2's static private key a_2.

(2) $K_2' = e\big((Y_3 + T_3), n(Y_1 + T_1)\big)^{a_2+u} \cdot X_2^{n-1} \cdot X_3^{n-2} \dots X_n$ is equal to $K_2' = \mathrm{e}((Y_3 + T_3), n((a_2 + u)Y_1 + (a_2 + u)T_1)) \cdot X_2^{n-1} \cdot X_3^{n-2} \dots X_n$ from the bilinear property of pairings in 2.1.2. And then,

$K_2' = \mathrm{e}\big((Y_3 + T_3), n(a_2Y_1 + uY_1 + a_2T_1 + uT_1)\big) \cdot X_2^{n-1} \cdot X_3^{n-2} \dots X_n$

$K_2' = \mathrm{e}\big((Y_3 + T_3), n(a_2a_1P + ua_1P + a_2a_1x_1P + ua_1x_1P)\big) \cdot X_2^{n-1} \cdot X_3^{n-2} \dots X_n$

$K_2' = \mathrm{e}\big((Y_3 + T_3), n(a_1(a_2P + uP) + a_1x_1(a_2P + uP)\big) \cdot X_2^{n-1} \cdot X_3^{n-2} \dots X_n.$

$K_2' = \mathrm{e}\big((Y_3 + T_3), n((a_1 + a_1x_1)(a_2P + uP)\big) \cdot X_2^{n-1} \cdot X_3^{n-2} \dots X_n.$

$K_2' = e\big((Y_3 + T_3), n(Y_2 + T_2)\big)^{a_1+a_1x_1} \cdot X_2^{n-1} \cdot X_3^{n-2} \dots X_n.$

However, attacker A will fail for this second way, too since he don't know correct a_1x_1 value, although he knows U_1's static private key a_1. Therefore, the protocol provides the property of key compromise impersonation resilience.

- *Unknown key-share resilience*. At the end of the common key establishment, the identity of each participant is included in the key derivation function. It provides unknown key-shared resilience in addtion to public key substitution unknown key shared attack.

- *Key Control*. Any party in the protocol doesn't control and predict the value of the common session key due to each one picks a new ephemeral private keys x_i in each run of the protocol to generate a unique session key.

4.2.4. Implementation Results of Key Agreement Protocol

A *Group Key Agreement Protocol* using *bilinear pairing* is implemented for four users. The implementation is developed under C programming and the implementation environment contains:

- Pairing Based Cryptography (PBC) Library (WEB_1 (2013)) for pairing map and elliptic curve calculations (point addition and scalar multiplication).
- GNU Multiple Precision Arithmetic (GMP) Library (WEB_2 (2013)) for multiple precision calculations,
- OpenSSL Library (WEB_3 (2013)) for secure hash algorithm, SHA-512 (FIPS PUB 180-2, (2002)).

The configuration of the machine on which the implementation run is as the following:

- CPU is Intel® Core™ 2 Duo 2.0 GHz,
- RAM is 4 GB,
- Operating System is Ubuntu 12.10.

All implementation details and the every values of each variable in the related steps can be found and followed under Appendix A.

Table 4.2 includes the input and setup parameters. There are super singular curve E, 512-bit prime q, 160-bit prime p, and a generator $P \in E/F_q$.

Each user generates his/her static public-private key pair for setup parameters. Static private keys a_i are picked from 160-bit group order p and static public keys Y_i are 512-bit length for each coordinate of point calculated from $a_i P$. Then, he/she applies a legitimate Certification Authority (CA) such as VeriSign (WEB_4, (2013)) and exchanges his/her certificate with other participants before, the first round of the protocol starts.

It is offline task of the protocol and numeric values of input and setup parameters in Table 4.2 are calculated using PBC Library (see Appendix A; Table A1 and Table A2 for numeric values).

Table 4.2. Input and Setup Parameters.

Input and Setup Parameters
User number is 4 namely; $U_i = \{U_1, U_2, U_3, U_4\}$. A super singular curve E defined by $y^2 = x^3 + x$ over F_q. The Weil pairing on the curve E/F_{q^2} is a mapping: $G_q \times G_q \rightarrow G_T$. q=512-bit prime and $q = p \cdot h + 1$ where h is multiple of 12. p= 160-bit prime P= 512-bit (each coordinate of point) generator where $P \in E/F_q$. a_i=160-bit static private key. Y_i = 512-bit (each coordinate of point) static public key where $Y_i = a_i P$.

In Table 4.3, each user picks ephemeral secret key x_i from 160-bit prime p and multiplies it with his/her public key Y_i. Approximate time for each user's $T_i = x_i Y_i$ computation is 4 milliseconds. T_i is a point on elliptic curve and its length is 1024 bits (which is defined by x-coordinate and y-coordinate, each of them is 512 bits). Specific time measures are available in the following table and numeric values are in the Table A.3 of Appendix A.

Table 4.3. Public Message Computation on Round-1.

Round-1
Step#1 Computation of public message with short term secret key of Round-1. $T_i = x_i Y_i$ Computation time of public message for U_i is approximately 3,6 milliseconds

In Table 4.4, each users applies SHA-512 hashing algorithm from OpenSSL Library for T_i. It outputs 512-bit result. Approximate time for each user's SHA-512 hashing computation is 37 μ seconds.

Then, each user applies BLS Short Signature algorithm for hashed value. Firstly they map this hashed value to a point on the curve E in order to be applied in pairing function for BLS Short Signature. After that, they sign this value with their static private keys a_i and broadcast to others. $Sig_i(T_i)$ is also a point on the curve and its

length is 1024bits. Approximate time for signing computation of each user is 10 milliseconds. Specific time measures of each user are available in the following table and numeric values are in the Table A.4 and Table A.5 of Appendix A.

Table 4.4. SHA-512 Algorithm and BLS Signature on Round-1.

Round-1
Contribution to satisfy the identification of the users and integrity of the public messages. **Step #2** Signing of the first public message T_i by U_i. $$H_i = SHA - 512(T_i)$$ $$Sig_{\mathrm{i}} = a_{\mathrm{i}} H_{\mathrm{i}}$$ T_i and sig_i are sent to all users and they are 512-bit length (for each coordinate of point). Approximate computation time for signing for each user is 10 milliseconds.

In Table 4.5, each user applies two pairing function; one for hashed value H_i and static public key Y_i; another one for signature result Sig_i and generator P. If the result of these functions are equal, then Sig_i is verified. Approximate time for BLS signature verification including hashing algorithm for each user is 14 milliseconds. Specific time measures are available in the following table and the numeric values are in Table A.6 of Appendix A.

Table 4.5. Verification of BLS Signature on Round-2.

Round-2
Contribution to satisfy the identification of the users and integrity of the public messages. **Step #3** Verification of U_i's signature and integrity checking of T_i. $$H_i = SHA - 512(T_i)$$ $$\mathrm{e}(Sig_{\mathrm{i}}, P) = e(H_{\mathrm{i}}, Y_{\mathrm{i}})$$ If this equality is satisfied; the receiver U_i can be sure the integrity of the T_i and legitimacy of sender. Approximate time for BLS signature verification including hashing algorithm for each user is 14 milliseconds.

In Table 4.6, each user performs $X_i = e((Y_{i+1} + T_{i+1}), (Y_{i+2}+T_{i+2}) - (Y_{i-1}+T_{i-1}))^{(a_i+a_i x_i)}$ calculation. X_i is a point on elliptic curve and length of X_i is 1024 bits. Approximate time for this calculation of each user is 3 milliseconds. Specific time measures are available in the following table and numeric values are in Table A.7 of Appendix A.

Table 4.6. Public Message Computation on Round-2.

Round-2
Step#4 Computation of public message of Round-2. $X_i = e((Y_{i+1} + T_{i+1}), (Y_{i+2}+T_{i+2}) - (Y_{i-1}+T_{i-1}))^{(a_i+a_i x_i)}$ Approximate time for this calculation of each user is 3 milliseconds.

In Table 4.7, each users applies SHA-512 hash algorithm from OpenSSL Library for X_i. Its hash result is 512-bits. Computation time of SHA-512 hashing algorithm for each user is approximately 20 μ seconds.

Then, each user applies BLS Short Signature algorithm for hashed value. Firstly they map this hashed value to a point on the curve E in order to be applied in pairing function for BLS Short Signature. Mapping results are available in the Table A.9 of Appendix A. Then, they sign this value with their static private keys a_i and broadcast to other users of the group. Each broadcasted message of $Sig_\mathrm{i} = a_\mathrm{i} H_\mathrm{i}$ is 1024-bit length due to be a point on the curve.. Approximate time for signing algorithm for each user is 10 milliseconds. Specific time measures are available in the following table and numeric values are in Table A.8 and A.9 of Appendix A.

Table 4.7. SHA-512 Algorithm and BLS Short Signature on Round-2.

Round-1
Contribution satisfies the identification of the users and integrity of the public messages. **Step #5** Signing of the second public message X_i by U_i. $$H_i = SHA\_512(X_i)$$ $$Sig_{\mathrm{i}} = a_{\mathrm{i}} H_{\mathrm{i}}$$ T_i and sig_i are sent to all users and they are 512-bit length (for each coordinate of point). Approximate time for signing algorithm for each user is 10 milliseconds.

In Table 4.8, each user applies two pairing function; one inputs hashed value H_i and static public key Y_i; another inputs signature result Sig_i and generator P. If result of each function is equal, then Sig_i is verified. Approximate time for BLS signature verification including hashing algorithm for each user is 13 milliseconds. Specific time measures are available in the following table and numeric values are in A.10 of Appendix A.

Table 4.8. Verification of BLS Short Signature on Key Agreement.

Key Agreement
Contribution to satisfy the identification of the users and integrity of the public messages. **Step #6** Verification of U_i's sign and integrity checking of X_i. $$H_i = SHA - 512(T_i)$$ $$\mathrm{e}(Sig_{\mathrm{i}}, P) = e(H_{\mathrm{i}}, Y_{\mathrm{i}})$$ If this equality is satisfied; the receiver U_i can be sure the integrity of the X_i and legitimacy of sender. Approximate time for BLS signature verification including hashing algorithm for each user is 13 milliseconds.

In Table 4.9, each user performs $K_i = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i + a_i x_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \dots X_{i-2}$ calculation for key agreement. Approximate time for this

calculation for each user is 3 milliseconds. Specific time measures are available in the following table and numeric values are in the Table A.11 of Appendix A.

Table 4.9. Common Key Agreement.

Key Agreement
Step #7 Computation of common key. $K_i = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i + a_i x_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \dots X_{i-2}$ $= e(P,P)^{\left[\begin{matrix}(a_n + a_n x_n)(a_1 + a_1 x_1)(a_2 + a_2 x_2) + (a_1 + a_1 x_1)(a_2 + a_2 x_2)(a_3 + a_3 x_3) + \cdots \\ + (a_{n-1} + a_{n-1} x_{n-1})(a_n + a_n x_n)(a_{n+1} + a_{n+1} x_{n+1})\end{matrix}\right]}$ Approximate time for this calculation for each user is 3 milliseconds.

Discussion:

The following Table 4.10 presents the summary of the implementation results and complexity analysis of the protocol.

Original protocol (Lin et at., (2006)) without thesis contribution needs 8,8 milliseconds computation time with n ·2048-bit length communication cost for n users in totality. Communication messages are defined by a point on the elliptic curve with 512-bit length for each coordinate; hence a point is defined by 1024-bit length.

In this work as a contribution, BLS Short Signature algorithm is added on public messages T_i and X_i of the original protocol in order to provide their message authentication and integrity. BLS Short Signature output is a point with 1024-bit length.

BLS Signature can be easily adapted to the original protocol by using same public parameters. But one of the drawback is that it needs two pairing algorithm for one verification algorithm. Hence, there will be added $4n$ (n is user number) pairing algorithm and also, $2n$ scalar multiplication to the original work as computation cost.

Total computation time for four users including these contributions is approximately 56 milliseconds with n ·4096- bit length - communication cost during the whole protocol.

Therefore, the implemented protocol fits small size group number such as less than 20 since its costly computation and communication complexity as seen in the Table 4.10. Original protocol can be applied for group number till 100; but it cannot provide

authentication and integrity of the public messages although it doesn't have any verifiability mechanism for the received public messages (its open problem).

Table 4.10. Summary of Results of the Implemented Protocol.

	Step # and Title	***Computational Cost (millisecond)***	***Communication Cost (bit length)***
Exchange (Round-1)	#1 Computation of public message T_i	3,6	$n \cdot 1024$
	Contribution #2 Signing of T_i	10,2	$n \cdot 1024$
	Sub-total cost without contribution steps.	3,6	$n \cdot 1024$
	Sub-total cost with contribution steps.	13,8	$n \cdot 2048$
Exchange (Round-2)	**Contribution** #3 Verification of U_i's signature and integrity checking of T_i.	13,7	-
	#4 Computation of public message X_i	2,6	$n \cdot 1024$
	Contribution #5 Signing of X_i	10	$n \cdot 1024$
	Sub-total cost without contribution steps.	2,6	$n \cdot 1024$
	Sub-total cost with contribution steps.	26,3	$n \cdot 2048$
Key Agreement (Round-2)	**Contribution** #6 Verification of U_i's signature and integrity checking of X_i.	13,3	-
	#7 Computation of common key.	2,6	-
	Sub-total cost without contribution steps.	2,6	-
	Sub-total cost with contribution steps.	15,9	-
	TOTAL COST WITHOUT CONTRIBUTIONS	8,8	$n \cdot 2048$
	TOTAL COST WITH CONTRIBUTIONS	56	$n \cdot 4096$

4.3. Group Key Distribution Protocol using Verifiable Secret Sharing

Verifiable Secret Sharing (VSS) allows parties of a protocol to be certain honesty of other parties in the same protocol in addition to share a common secret using *secret sharing*. It takes place in construction of many cryptographic protocols such as; multi party computation, secure storage, e-voting system in addition to key distribution. In group key distribution protocols, it provides constant round to share a secret independent from the party number. Our second protocol is based on Feldman' VSS (Feldman (1987)).

4.3.1. Preliminaries and Communication Model of Key Distribution Protocol

This protocol is defined under the specifications and assumptions which are mentioned in Section 4.1.

The second protocol consists of $n+1$ entities that form secure, closed group. There are *n* parties who participates the establishment of common secret and a leader *L* who picks and shares the *common secret s* among the n participants of the protocol.

The protocol model uses point to point communication between each $n+1$ entities and it needs only two rounds namely; sharing and reconstruction. Also, number of communicating parties is independent of number of communication rounds. There is a threshold value t means that more than t shared subsecret value can used to reconstruct the common secret generated by the leader. The relation between threshold t and n is $n \geq 2t+1$ (Shamir (1979), Shannon (1949)). There can be occurred new party by *addition* or *removal* users to/from the group during any round of the protocol under the condition that preserving parameterized (n, t) values at same.

This (n, t) values (Shamir (1979)) is information theoretically secure even when the adversary has unlimited computing power. So, the adversary simply does not have enough information to break the encryption unless he has $t+1$ values (Shannon (1949)).

In order to provide secure and authentic communication between the $n+1$ entities, a PKI (Section 3.1) is applied. Before the protocol run, each legitimate party

obtains his certificate including his *public key* and *numeric* ID_i from CA (Section 3.1.1). Especially, in each round, all sent messages are encrypted and/or digitally signed.

Received messages in each round can be verified using DLog commitments to be sure the validity of received content from leader or other parties (Section 2.1.4). These commitments are created by the leader. Since the users fully trust the leader by using the commitments that he generates, authentication is vitally important between the leader and the other parties.

4.3.2. Construction of Key Distribution Protocol

It consists of *sharing phase* which is based on distribution of common secret via t^{th} degree polynomial and *reconstruction phase* which is based on establishment of the common secret via *Lagrange interpolating polynomial* in Section 2.1.1 of at least $t + 1$ private subsecrets (this is called as *shadow* or *share* in some technical documents).

Setup

Notations used in the Group Key Distribution Protocol as the following;

- Primes q and p such that $q|p - 1$, and a generator $g \in Z_p^*$ is an element of order q.
- s is common secret will be shared.
- $s_i = \{s_1, s_2, \ldots, s_n\}$ represents *subsecret* pooled of which participants will be establish the *common secret* s with.
- L is a leader who shares the *common secret* s among the n participants.
- U_i is a participant for the establishment of common secret s.
- $U = \{U_1, U_2, \ldots, U_n\}$ and U_i represents notationally current party whereas U_m represents sender party and U_j represents receiver party ($U_m \rightarrow U_i \rightarrow U_j$).
- $Cert_i$ is certificate which includes U_i's long term public key Y_i and numeric identifier ID_i.
- x_i is U_i's long term secret key of corresponding public key Y_i.
- t is threshold value (Shamir (1979), Shannon (1949)).
- $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_t x^t$, is a t degree polynomial in which the secret $s = a_0$ and all coefficients a_j are $a_0, \ldots, a_t \in Z_q$. This is randomly built up by Leader.

- C is a commitment vector including different $t+1$ DLog commitments; $C_0 = g^{a_0}, C_1 = g^{a_1}, .., C_t = g^{a_t}$.
- The public parameters are $param = \langle q, p, g \rangle$.

Before beginning of the protocol, each $n+1$ entity obtains his/her certificate $Cert_i$ and exchanges with other participants of the protocol (Section 3.1.1) and verifies received $Cert_i$ with CA. Hence all group users have been identified.

Sharing (Round-1)

The protocol starts with the Leader L's initiator calling to the participant set $U = \{U_1, U_2, \ldots, U_n\}$ and specifying the n group number. Threshold value t will take the maximum number with respect to the $n \geq 2t+1$ relation such that $\frac{n-1}{2} \geq \lceil t \rceil$ (Shamir (1979), Shannon (1949)). Leader L performs computations of round-1 as the following;

- Determines randomly the common secret $s\,(mod\ q)$ will be shared to the participant set U.
- Then, randomly chooses a $f(x)$ polynomial of degree t such that; $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_t x^t$, in which the secret $s = a_0$ and all coefficients a_j are randomly selected as $a_0, \ldots, a_t \in Z_q$.
- Constructs the commitment vector C based on DLog commitments as explained in Section 2.1.4. $C_j = (C_0 = g^{a_0}, C_1 = g^{a_1}, .., C_t = g^{a_t})\ (mod\ p)$ and $0 \leq j \leq t,\ t \in Z$.
- Computes subsecrets $s_i = f(ID_i)\ (mod\ q)$ where ID_i is numeric identifier $(mod\ q)$ in $Cert_i$.
- Encrypts the subsecrets s_i using static public keys Y_i of corresponding recipient for secrecy such that $Enc_{Y_i}[s_i\ ]$.
- Signs the commitment vector C for integrity and authentication, such that $Sig_{Leader}[(C_0 = g^{a_0} || C_1 = g^{a_1} || \ldots || C_t = g^{a_t})]$.
    - Enc and Sig algorithms can be implemented by the usage of different cryptographic systems. On the implementation in Section 4.3.4 and Appendix A, ElGamal Encryption and Signature Scheme are preferred.
- Sends encrypted subsecrets $Enc_{Y_i}[s_i\ ]$ and signed commitment vector $Sig_{Leader}[C]$ to receivers by peer to peer on open unsecure channel.

Reconstruction (Round-2)

- Each U_i verifies signed commitment vector using Leader's public key.
- Each U_i decrypts the received message $Enc_{Y_i}[s_i]$ from Leader using corresponding private key x_i, $Dec_{x_i}\left[Enc_{Y_i}[s_i]\right] = s_i$.
- Then, verify correctness of received subsecret s_i by using commitment vector

$$g^{s_i}(mod\ p) = \prod_{j=0}^{t} C_j^{i^j}\ (mod\ p)\ where\ C_j = g^{a_j}(mod\ p)$$

and; $i\ (i = 1,..,n)$ is indice of participants and $j\ (j = 0,\ldots,t)$ is indice of polynomial coefficients.

- If verification is consistent, each U_i encrypts his subsecret s_i, using each recipient U_j's public key Y_j, such that $Enc_{Y_j}[s_i]$.
- Sends $Enc_{Y_j}[s_i]$. to related party U_j.

Key Establishment (Round-2)

- Each U_j decrypts the received encrypted messages $Enc_{Y_j}[s_i]$ using his private key x_j, such that $Dec_{x_j}\left[Enc_{Y_i}[s_i]\right] = s_i$. Hence the each participant $U_j$will have all other users subsecrets.
- Then, verify correctness of received subsecret s_i by calculation of the commitment vector;

$$g^{s_i}(mod\ p) = \prod_{j=0}^{t} C_j^{i^j}\ (mod\ p)\ where\ C_j = g^{a_j}(mod\ p)$$

and; $i\ (i = 1,..,n)$ is indices of participants who U_j received subsecret s_i from, and $j\ (j = 0,\ldots,t)$ is indices of polynomial coefficients.

- If verification of each s_i is consistent, to establish the common secret s (which is $s = a_0 = f(0)$) each party U_i have to construct polynomial $f(x)$ from any $t+1$ verified subsecrets s_i which are collected in set W_i then by using Lagrange Interpolation formula in the following steps;

  Firstly, Lagrange Coefficient is calculated by $L_i(0) = \lambda_i^w = \prod_{U_i,U_j \in W, j \neq i}^{t+1} \frac{ID_j}{ID_j - ID_i}$

  and then; $f(0) = \sum_{U_i \in W}^{t+1} L_i(0) s_i$.

Simple summary of each phase for 5 group number is available in the Table 4.11. n is selected is 5 due to $n \geq 2t + 1$ rule, here $t = 2$. Implementation of the protocol for four users and the leader L with 512-bit and 1024-bit primes is performed in the Section 4.3.4 in order to present time measures of computation and bit-lengths of communication complexities of each user on per round. Also, numeric results of the variable in the implementation can be followed under Appendix B.

Table 4.11. Group Key Distribution Protocol.

Setup	***Creation of Public Parameter***
	Primes q and p such that $q \mid p - 1$, and a generator $g \in Z_p^*$ is an element of order q.
	(n, t) group number for $\frac{n-1}{2} \geq \lceil t \rceil$ = (5,2) Each participant including the leader exchange their certificates.
Sharing (Round 1)	***Private Computations*** Leader L
	Step #1 Polynomial Generation n=5 and t=2 and satisfies $n \geq 2t + 1$ rule. Choose common secret secret $s = a_0$ and generates 2 degree $f(x) = a_0 + a_1 x + a_2 x^2 \in Z_q$.
	Step #2 Commitment Vector Generation Computes commitments in $C = (C_0 = g^{a_0}, C_1 = g^{a_1}, C_2 = g^{a_2}) \ (mod\ p)$.
	Step #3 Generation of subsecrets for users U_1, U_2, U_3 and U_4. Computes the subsecrets $s_1 = f(ID_1) \ (mod\ q)$, $s_2 = f(ID_2) \ (mod\ q), \ldots, s_5 = f(ID_5) \ (mod\ q)$ where ID_i is numeric identifier $(mod\ q)$ in $Cert_i$.
	Contribution to satisfy the requirements of identification of the leader and integrity of commitment vector. **Step #4** Signing of Commitment Vector by the Leader. Signs commitments $Sig_{Leader}[(C_0 = g^{a_0} \mid\mid C_1 = g^{a_1} \mid\mid C_2 = g^{a_2})]$.

(cont. on next page)

Table 4.11. (cont.)

Sharing (Round 1-cont.)	**Contribution** satisfies the secrecy requirement of shared subsecrets for robustnest to attacks. **Step #5** Encryption of generated subsecrets for each of the receiver users of the group separately by using ElGamal Encryption scheme. Encrypts; $Enc_{Y_1}[s_1\ ]$, $Enc_{Y_2}[s_2\ ], \ldots, Enc_{Y_5}[s_5\ ]$.
	Public Exchange of Values Leader L sends each message Enc_{Y_i} to each related U_i (point to point) and commitment vector C with $Sig_{Leader}[C]$.
Reconstruction (Round 2)	***Private Computations*** Each U_i $(i = 1, 2, \ldots, 5)$
	Contribution **Step #6** Verification of Leader's signature and integrity checking of the Commitment Vector. (sig_1, sig_2) pair with commitment vector are received by U_i. Each $U_i$verifies Leaders Signature using Leader's public key.
	Contribution **Step #7** Decryption of received subsecret. U_i decrypts the received message $Enc_{Y_i}[s_i\ ]$, using corresponding private key x_i, $Dec_{x_i}\left[Enc_{Y_i}[s_i\ ]\right] = s_i$.
	Step #8 Verification correctness of received subsecrets by using commitment vector; $g^{s_i}(mod\ p) = \prod_{j=0}^{t} C_j^{i^j}\ (mod\ p)\ where\ C_j = g^{a_j}(mod\ p)$
	Contribution: Each U_i encrypts its subsecret and send to other users of the group to satisfy secrecy requirement and robustnest to attacks. **Step #9** Encryption processes of subsecret of U_i: U_i encrypts subsecret s_i. $Enc_j[s_i, Y_j]$ and sends to every U_j point to point. ***For*** $\boldsymbol{U_1}$: $Dec_{x_1}\left[Enc_{Y_1}[s_1\ ]\right] = s_i$ $g^{s_1}(mod\ p) = \prod_{j=0}^{2} C_j^{1^j} = (g^{a_0})^{1^0} \cdot (g^{a_1})^{1^1} \cdot (g^{a_2})^{1^2}(mod\ p)$

(cont. on next page)

Table 4.11. (cont.)

Reconstruction (Round 2-cont.)	**Step #9 (cont.)** . . ***For U_5***: $Dec_{x_5}\left[Enc_{Y_5}[s_5\ ]\right] = s_i$ $$g^{s_5}(mod\ p) = \prod_{j=0}^{2} C_j^{5^j} = (g^{a_0})^{5^0} \cdot (g^{a_1})^{5^1} \cdot (g^{a_2})^{5^2}(mod\ p)$$
	Public Exchange of Values ***For U_1,*** Encrypts subsecret s_1.sends $Enc_{Y_2}[s_1\ ]$ to U_2. Encrypts subsecret s_1. sends $Enc_{Y_3}[s_1\ ]$ to U_3. Encrypts subsecret s_1. sends $Enc_{Y_4}[s_1\ ]$ to U_4. Encrypts subsecret s_1. sends $Enc_{Y_5}[s_1\ ]$ to U_5. . . ***For U_5,*** Encrypts subsecret s_5. sends $Enc_{Y_1}[s_5\ ]$ to U_1. Encrypts subsecret s_5. sends $Enc_{Y_2}[s_5\ ]$ to U_2. Encrypts subsecret s_5. sends $Enc_{Y_3}[s_5\ ]$ to U_3. Encrypts subsecret s_5. sends $Enc_{Y_4}[s_5\ ]$ to U_4. There are 20 communication messages $Enc_{Y_j}[s_i\ ]$.
Key Establishment (Round-2)	***Further Private Computations*** Each U_i $(i\ =\ 1,2,\ldots,5)$
	Contribution **Step #10** Decryption process of received encrypted subsecrets by U_i: U_i decrypts the received messages $Enc_{Y_i}[s_m\ ]$,using corresponding private key x_i, $Dec_{x_i}\left[Enc_{Y_i}[s_m\ ]\right] = s_m$.
	Step #11 Verification of correctness of subsecret s_m by using commitment vector; $$g^{s_m}(mod\ p) = \prod_{j=0}^{t} C_j^{m^j}\ (mod\ p)\ where\ C_j = g^{a_j}(mod\ p)$$ If verification is consistent, each U_i use lagrange interpolating formula $f(0) = \sum_{U_i \in W}^{t+1} L_i(0)s_i$.where $L_i(0) = \lambda_i^W = \prod_{U_i, U_m \in W, m \neq i}^{t+1} \frac{ID_m}{ID_m - ID_i}$ from any $(t+1)$ s_i. . ***For U_1***: $Dec_{x_1}\left[Enc_{Y_1}[s_2\ ]\right] = s_2$ $$g^{s_2}(mod\ p) = \prod_{j=0}^{2} C_j^{2^j} = (g^{a_0})^{2^0} \cdot (g^{a_1})^{2^1} \cdot (g^{a_2})^{2^2}(mod\ p)$$ $Dec_{x_1}\left[Enc_{Y_1}[s_3\ ]\right] = s_3$ $g^{s_3}(mod\ p) = \prod_{j=0}^{2} C_j^{3^j} = (g^{a_0})^{3^0} \cdot (g^{a_1})^{3^1} \cdot (g^{a_2})^{3^2}(mod\ p)$.

(cont. on next page)

Table 4.11. (cont.)

Key Establishment (cont.)	. $Dec_{x_1}\left[Enc_{Y_1}[s_4\ ]\right] = s_4$ $g^{s_4}(mod\ p) = \prod_{j=0}^{2} C_j^{4^j} = (g^{a_0})^{4^0} \cdot (g^{a_1})^{4^1} \cdot (g^{a_2})^{4^2}(mod\ p).$. $Dec_{x_1}\left[Enc_{Y_1}[s_5\ ]\right] = s_5$ $g^{s_5}(mod\ p) = \prod_{j=0}^{2} C_j^{5^j} = (g^{a_0})^{5^0} \cdot (g^{a_1})^{5^1} \cdot (g^{a_2})^{5^2}(mod\ p).$ After verification, any 3 subsecrets is sufficient to construct the s, set $W = (U_1, U_2, U_3)$ for U_1. $s = f(0) = s_1 \frac{ID_2}{ID_2 - ID_1} \cdot \frac{ID_3}{ID_3 - ID_1} + s_2 \frac{ID_1}{ID_1 - ID_2} \cdot \frac{ID_3}{ID_3 - ID_2} + s_3 \frac{ID_1}{ID_1 - ID_3} \cdot \frac{ID_2}{ID_2 - ID_3}$ . . . ***For U_5:*** $Dec_{x_5}\left[Enc_{Y_5}[s_1\ ]\right] = s_1$ $g^{s_1}(mod\ p) = \prod_{j=0}^{2} C_j^{1^j} = (g^{a_0})^{1^0} \cdot (g^{a_1})^{1^1} \cdot (g^{a_2})^{1^2}(mod\ p)$. $Dec_{x_5}\left[Enc_{Y_5}[s_2\ ]\right] = s_2$ $g^{s_2}(mod\ p) = \prod_{j=0}^{2} C_j^{2^j} = (g^{a_0})^{2^0} \cdot (g^{a_1})^{2^1} \cdot (g^{a_2})^{2^2}(mod\ p).$. $Dec_{x_5}\left[Enc_{Y_5}[s_3\ ]\right] = s_3$ $g^{s_3}(mod\ p) = \prod_{j=0}^{2} C_j^{3^j} = (g^{a_0})^{3^0} \cdot (g^{a_1})^{3^1} \cdot (g^{a_2})^{3^2}(mod\ p).$ $Dec_{x_5}\left[Enc_{Y_5}[s_4\ ]\right] = s_4$ $g^{s_4}(mod\ p) = \prod_{j=0}^{2} C_j^{4^j} = (g^{a_0})^{4^0} \cdot (g^{a_1})^{4^1} \cdot (g^{a_2})^{4^2}(mod\ p).$ After verification, any 3 subsecrets is sufficient to construct the s, set $W = (U_5, U_1, U_4)$ for U_5. $s = f(0) = s_1 \frac{ID_4}{ID_4 - ID_1} \cdot \frac{ID_5}{ID_5 - ID_1} + s_4 \frac{ID_1}{ID_1 - ID_4} \cdot \frac{ID_5}{ID_5 - ID_4} + s_5 \frac{ID_1}{ID_1 - ID_5} \cdot \frac{ID_4}{ID_4 - ID_5}$

Group Modifications:

Secret construction of the protocol depends on polynomial interpolation. Actually, pairs which comprises of numeric identifiers ID_i and subsecrets s_i such that (ID_i, s_i) forms points in the t degree polynomial $f(x)$ generated by the leader. If entity addition and removal occurs simultaneously in the same amount, there won't be any change in the subsecrets of old parties and the common secret s since the (n, t) parameter remains the same. This modification is included the protocol at any time

since the Leader acts the distribution of subsecrets with the same parameters and therefore, establishment of secret s happens for only the new parties with new identifiers ID_{new}.

However, if entity addition and removal isn't balanced (in the same amount), the subsecrets of old parties and the common secret s will change since the protocol starts with new (n, t) parameters.

4.3.3. Security Analysis of Key Distribution Protocol

Requirements of secure communication in Section 3.1 between the parties and leader are provided by certificates ($Cert_i$) and certification authority (CA). Consistency of all subsecrets s_i in communication messages coming from the leader and the other participants can be verified by using commitment C vector.

VSS has two security requirements defined in Section 2.2.3: *Secrecy* and *Correctness*.

- *Secrecy*: During both two rounds, each point to point sending messages (subsecrets s_i) are encrypted via static public keys Y_i which of corresponding private key x_i are certified by CA. Secrecy of communication messages are provided by this way. An adversary A needs to compromise at least $t + 1$ static private keys x_i in order to encrypt the corresponding public messages and then he acquires the subsecrets s_i and uses them in Lagrange interpolation to establish the common secret s.
- *Correctness*: Suppose an honest leader L has shared a t degree polynomial $f(x)$ with constant term to the common secret s. As the leader is honest, the message coming from the leader contains commitment C and $f(ID_i)$ for a honest party U_i in the first round. Party U_i accepts his subsecret $f(ID_i)$ if it consistent with the commitment C. In the second round, party U_i receives the subsecrets $f(ID_m) = s_m$ of each U_m. Party U_i accepts the subsecrets $f(ID_m) = s_m$ if it is consistent with the commitment C. When U_i has $t + 1$ (set W) consistent subsecrets, U_i constructs common secret s such that;
    - Let λ_i^W be Lagrange interpolation coefficients for the set W such that $\lambda_i^W = \prod_{U_i, U_j \in S, i \neq j}^{t+1} \frac{ID_j}{ID_j - ID_i}$. U_i has $s' = \sum_{U_i \in S}^{t+1} \lambda_i^W s_i = \sum_{U_i \in S}^{t+1} \lambda_i^W f(ID_i) = s$

if the leader and the other parties has sent consistent subsecrets s_i and then, $s' = s$.

- Let assume two distinct honest parties U_i and U_j reconstruct values s'_i and s'_j by interpolating two distinct sets W_i and W_j of $t+1$ subsecrets each, which are valid with respect to the unique commitment C. As the subsecrets in W_i and W_j are verified against commitment C and they are valid, it is easy to see that $g^{s'i} = C_0 = g^{s'j}$. As g is a generator for a prime order group, $s'_i = s'_j$ (Table 4.11 illustrates example of such parties U_i and U_j namely; U_1 and U_5 in the *key establishment* column).

On the other hand, this work needs formal security inspections in order to prove security of the protocol in the presence of passive and active adversary. It forms *the future tasks* of the research.

4.3.4. Implementation Results of Key Distribution Protocol

A *Group Key Distribution Protocol* using *verifiable secret sharing scheme (VSS)* is implemented for four users and a Leader. The implementation is developed under C programming and the implementation environment contains:

- GNU Multiple Precision Arithmetic (GMP) Library (WEB_2, (2013)) for multiple precision calculations,
- OpenSSL Library (WEB_3, (2013)) for secure hash algorithm, SHA-512 (FIPS PUB 180-2, (2002)).

The configuration of the machine on which the implementation run is as the following:

- CPU is Intel® Core™ 2 Duo 2.0 GHz,
- RAM is 4 GB,
- Operating System is Ubuntu 12.10.

All implementation details and the every values of each variable in the related steps can be found and followed under Appendix B.

Table 4.12 includes the input parameters which are 1024-bit prime p with primitive root g of Z_p^* and 512-bit prime q where $q \mid (p-1)$. This test data is acquired from (Allen (2008)).

Each user generates his/her static public private key pair for setup parameters. Static private keys x_i are picked from 1024-bit group order p and static public keys Y_i are 1024-bit length calculated from $Y_i = g^{x_i} \ (mod\ p)$. Then he/she applies a legitimate Certification Authority (CA) such as VeriSign (WEB_4, (2013)) and exchanges his/her certificate with other participants before the first round of the protocol starts. The certificates include static public key Y_i and an unique numeric identifier ID_i.

It is offline task of the protocol and numeric values of input and setup parameters in Table 4.12 are calculated using GMP Library (see Appendix B; Table B.1 and Table B.2 for numeric values).

Table 4.12. Input and Setup Parameters.

Input and SetupParameters
There are Leader L and 4 users, namely; U_1, U_2, U_3 and U_4. p=1024-bit prime g= 1024-bit primitive root of Z_p^* q= 512-bit prime where $q \mid (p-1)$. x_i= 1024-bit static private key $\in Z_p^*$. Y_i = 1024-bit static public key where $Y_i = g^{x_i} \ (mod\ p)$. $Cert_i$ =(Y_i, ID_i).

In Table 4.13, the Leader generates 2-degree polynomial $f(x)$ (degree size is depending on user number as explained in section 4.3.2) and picks the common secret which is $f(0)$. Polynomial generation takes approximately 21 microseconds. Then Leader calculates commitment vector which takes is approximately 6 milliseconds. Finally Leader compute subsecrets which will be distribute to the each users. Subsecret computation takes 16 μ seconds for four users (details are presented by Table B.3 of Appendix B).

Table 4.13. Polynomial Generation, Commitment and Subsecret Calculation.

	Round-1
Leader L	**Step #1** Polynomial Generation $$f(x) = a_0 + a_1 x + a_2 x^2 \; mod \; q$$ a_0 would like to be shared among to the four participants U_1, U_2, U_3 and U_4 as common SECRET KEY, and randomly selected by the Leader who is started the group communication. a_0, a_1 , $a_2 \in Z_q$ are selected randomly. Total computation time for *polynomial generation* is 21 μ seconds.
	Step #2 Commitment Vector Generation $$C_j = (C_0 = g^{a_0}, C_1 = g^{a_1}, C_2 = g^{a_2}) \; (mod \; p)$$ Total computation time for *generation of commitment vector* is 5,483 milliseconds.
	Step #3 Generation of subsecrets for users U_1, U_2, U_3 and U_4. $$s_i = f(ID_i) \; (mod \; q)$$ Subsecret generation time is approximately 4 μ seconds per user. Total *subsecrets* generation time for four users is 16 μ seconds.

In Table 4.14, Leader uses ElGamal signature algorithm in order to sign the commitment vector. Elements in commitment vector are concatenated and input SHA-512 hash algorithm from OpenSSL Library. Signature pair is sent to all users (details are presented by Table B.4 of Appendix B).

Table 4.14. SHA-512 Algorithm and ElGamal Signature.

	Round-1
Leader L	**Contribution** satisfies the identification of the leader and integrity of commitment vector requirements. **Step #4** Signing of Commitment Vector by the Leader. $H_{Commit} = SHA - 512(C_0 \| C_1 \| C_2)$ Computation time to generate hash value of commitment vector by using SHA-512 is 58 μ seconds. $sig_1 = g^r \ (mod\ p)\ \ where$ random r is gcd$(r, p-1) = 1$ $sig_2 = (H_{Commit} - x_L s_1) r^{-1} \ \ mod\ (p-1)$ (x_L is static private key of Leader). (sig_1, sig_2) pair has 2048 bits (each of them 1024-bit length) and they sent to all users. Signing of hash value of commitment vector is 1,548 milliseconds. Total time to sign commitment vector is 1,606 millisecond.

In Table 4.15, Leader encrypts calculated subsecrets using ElGamal encryption. Total calculation time for all users is 10,391 milliseconds. 1024-bit length encrypted messages are sent to related users. Specific time measures are available in the following table (see Table B.5 of Appendix B).

Table 4.15. ElGamal Encryption of Subsecrets.

	Round-1
Leader L	**Contribution** satisfies the secrecy requirement of shared subsecrets for robustnest to attacks. **Step #5** Encryption of generated subsecrets for each of the receiver users of the group separately by using ElGamal Encryption scheme. $ca_1 = g^{r_1} \ (mod\ p) where\ random\ r_1\ is\ \{1, 2, .., \mathrm{p}-1\}$ $ca_2 = s_1 (Y_1)^{r_1}\ mod\ p$ (ci_1, ci_2) pairs has 2048bit length (each of them 1024-bit length) and it is sent to each related U_i, where $1 \leq i \leq 4$. The encryption time is approximately is 2,558 milliseconds for per user. Total encryption time for four users is 10,391milliseconds.

In Table 4.16, each user verifies signature on commitment vector. Total time for verification calculation of per user is 3,402 with hashing algorithm (see Table B.6 of Appendix B).

Table 4.16. ElGamal Signature Verification.

	Round-2
Each User U_i	**Contribution** **Step #6** Verification of Leader's signature and integrity checking of the Commitment Vector. (sig_1, sig_2) pair with commitment vector are received by U_i. Each $U_i$verifies Leaders Signature using Leader's public key. $$g^{H_{commit}} = (Y_L)^{sig_1} {sig_1}^{sig_2} \ mod\ p$$ ($Y_L$is the public key of Leader). To verification first; $H_{Commit} = SHA\_512(C_0 \| C_1 \| C_2)$ is calculated. Computation time to generate hash value of commitment vector by using SHA-512 is 58 μ seconds. $$g^{H_{commit}} = (Y_L)^{sig_1} {sig_1}^{sig_2} \ mod\ p$$ If this equality is satisfied; the receiver U_i can be sure the integrity of the commitment vector and legitimacy of sender L. Total calculation time of signature verification is 3,402 milliseccond per user.

In Table 4.17 each user decrypts received encrypted message to output his/her subsecret. Approximate decryption time for each user is 1,337 milliseconds. Specific computation time is available in the following table (see Table B.7 of Appendix B).

Table 4.17. ElGamal Decryption.

	Round-2
User U_i	**Contribution** **Step #7** Decryption of received subsecret. Each U_i decrypts incoming encrypted subsecret from U_j. $$s_j = ca_2 \cdot ((ca_1)^{x_i})^{-1} \; mod \; p$$ The decryption time of each subsecret is approximately 1,3 milliseconds.

In Table 4.18, each user verifies correctness of his/her subsecret using commitment vector. Approximate verification time for each user is 748 μ seconds. Specific computation time is available in the following table (see Table B.8 of Appendix B).

Table 4.18. Verifying Correctness of Subsecrets by Using Commitment Vector.

	Round-2
User U_i	**Step #8** Verification correctness of subsecrets by using commitment vector. Each U_i verifies received subsecret. $$g^{s_i}(mod\ p) = \prod_{j=0}^{t} C_j^{i^j}\ (mod\ p)\ where\ C_j = g^{a_j}(mod\ p)$$ and; i $(i = 1,..,n)$ is indice of participants and j $(j = 0,\ldots,t)$ is indice of polynomial coefficients. The calculation time for verification of subsecret is approximately 774 μ seconds per user U_i.

In Table 4.19, each U_i encrypts his subsecret using ElGamal encryption and the receiver's public key. The computational time of each encryption of subsecret is approximately 2,580 milliseconds per U_i, then user sends this $(c_1, c_2)_i$ pair (each of them 1024-bit) to all other users of the group. The user U_i have to do this same job for the each of the members of the group hence the total execution time of this process for

U_i is approximately 7,596 milliseconds (see Table B.9, B.10, B.11 and B.12 of Appendix B).

Table 4.19. Encryption of own subsecret by U_i.

User U_i	**Step #9** Each U_i sends its subsecret to other user of the group. **Contribution:** Each U_i encrypts its subsecret and send to other users of the group to satisfy secrecy requirement and robustnest to attacks. **Step #9** Encryption processes of subsecret of U_i: $$c_1 = g^r \ (mod\ p) where\ random\ \ r \epsilon Z_p^*$$ $$c_2 = s_i (Y_j)^r \ mod\ p\ where\ s_i\ is\ subsecret\ of\ U_i$$ (Y_j is receiver's public key). The computational time of each encryption of subsecret is approximately 2,580 milliseconds per U_i. Total execution time of this process for U_i is approximately 7,596 milliseconds.

In Table 4.20 each user decrypts received encrypted subsecret messages which are sent by the other users of the group. The computation time for the decryption of each user's subsecret is approximately 1,3 milliseconds. Each user have to do decryption for messages of other three users. Therefore for four user, the decryption of subsecrets will take approximately 4 milliseconds (see Table B.13 of Appendix B).

Table 4.20. Decryption of Received Subsecrets of Senders.

	Round-2 – Key Establishment
User U_i	**Contribution:** decryption of received encrypted subsecrets. **Step #10** Decryption process of received encrypted subsecrets by U_i: $(c_1, c_2)_i$ is received from each user of the group. $s_i = c_2 \cdot ((c_1)^{x_j})^{-1} \ mod\ p$ where $i, j \in \{1,2,3,4\}$, i defines sender, j defines receiver user, here x_j is private key of receiver. The computation time for the decryption of each user's subsecret is approximately 1,3 milliseconds. The computation time of decryption of subsecrets for four users take approximately 4 milliseconds.

In Table 4.21 each user verifies received subsecrets by using commitment vector. Approximate calculation time of each user for subsecret verification is 2,3 milliseconds. Specific calculations of users are available in the following table (see Table B.14 of Appendix B).

Table 4.21. Subsecrets Verification using Commitment Vector.

	Key Establishment
User U_i	**Step #11** Verification of subsecret by using commitment vector. Every user have to have verified (t+1) subsecret to generate common share secret by the constant a_0 of the polinomial. Each U_i verifies received subsecrets from other users. $$g^{s_i}(mod\ p) = \prod_{j=0}^{t} C_j^{i^j}\ (mod\ p)\ where\ C_j = g^{a_j}(mod\ p)$$ and; i $(i = 1,..,n)$ is indice of participants and j $(j = 0,\ldots,t)$ is indice of polynomial coefficients. The calculation time for verification of subsecret is approximately 774 microseconds per user U_i. Total calculation time for all three received subsecrets is approximately 2,3 milliseconds. All users do this job simultaneously.

In Table 4.22 each user applies Lagrange Interpolation. When a user verifies all subsecrets as shown in the Table 4.21, then he/she can establish the common secret key using Lagrange Interpolation from any $t + 1$ subsecrets. Approximate calculation time of each user for Lagrange Interpolation is 35 μ seconds. Specific calculations of users are available in the following table (see Table B.15 of Appendix B).

Table 4.22. Key Establishment by Lagrange Interpolation.

	Key Establishment
User U_i	**Step #12** Lagrange Interpolation and common secret key establishment. Each party U_i have to construct polynomial $f(x)$ from any $t+1$ verified subsecrets s_i which are collected in set W_i then by using Lagrange interpolation formula in the following steps; First Lagrange coefficient is calculated by $L_i(0) = \lambda_i^W = \prod_{U_i, U_j \in W, j \neq i}^{t+1} \frac{ID_j}{ID_j - ID_i}$ and then; $f(0) = \sum_{U_i \in W}^{t+1} L_i(0) s_i$ (Note that $s = a_0 = f(0)$) For example the formula for the U_1 is: $a_0 = f(0) = s_1 \frac{ID_2}{ID_2 - ID_1} \cdot \frac{ID_3}{ID_3 - ID_1} + s_2 \frac{ID_1}{ID_1 - ID_2} \cdot \frac{ID_3}{ID_3 - ID_2} + s_3 \frac{ID_1}{ID_1 - ID_3} \cdot \frac{ID_2}{ID_2 - ID_3}$ Total computation time for Lagrange Interpolation of U_i is 35 μ seconds.

Discussion:

Table 4.23 presents the summary of the implementation results for one user. Original protocol (Feldman, (1987)) without thesis contributions needs 9 milliseconds computation time with 7168-bit communication cost for four users in total. There are three commitment elements, each of which is 1024-bit; four subsecrets (will be sent in secure channel) each of which is 512-bit in the first round and four subsecrets each of which is 512-bit in the second round. So, total communication cost is 3·1024 + 4·512 + 4·512 = 7168-bit length. Results are close to Lin et al.'s protocol for four users.

In this work, ElGamal Signature algorithm is added on publicly known comminment to provide its message authentication and integrity. ElGamal Signature can be easily adapted to the original protocol by using same public parameters. Due to ElGamal Signature gives the efficient results comparing the BLS Signature, it should be applied in the first protocol in order to optimize computation time.

It outputs 1024-bit signed message in addition to 3·1024-bit communication cost of the original work. Also, ElGamal Encryption algorithm is applied to subsecrets of the protocol in order to be adapted on open channel. There are four encrypted messages

with 1024-bit in the first round and 4·(4-1) encrypted messages with 1024-bit length in the second round. So, total communication cost is 3·1024 + 1024 + 4·1024 + 12·1024 = 20480 bit length. Total computation time for four users including these contributions is approximately 37,5 milliseconds with 22528-bit length communication cost during the whole protocol.

ElGamal Signature is more efficient than BLS Short Signature. Results of signing computation of BLS Signature is approximately 10 milliseconds and verification computation is approximately 13 milliseconds. On the other hand, signing computation of ElGamal is approximately 1,6 milliseconds and verification is 3,4 milliseconds.

Total computation time for four users including these contributions is approximately 37,5 milliseconds with 20480-bit length communication cost during the whole protocol.

The proposed protocol isn't proper for group key establishment protocol. Although its results of computation time look more efficient than the first protocol for small group size, it has very costly communication overhead. When group size increases, computation time will be getting inefficient, too due to dependency of (n, t) parameters.

Table 4.23. Summary of Results of the Implemented Protocol.

	Step # and Title	***Computational Cost (millisecond)***	***Communication Cost (bit length)***
Round-1 LEADER	#1 Polynomial Generation	0,021	
	#2 Commitment Vector Generation	5,5	$(t+1)\cdot 1024$
	#3 Generation of subsecrets for users U_1, U_2, U_3 and U_4.	0,016	$(n-1)\cdot 512$
	Contribution #4 Signing of Commitment Vector	1,6	1024
	Contribution #5 Encryption of generated subsecrets for each of the receiver users separately	10,5	$(n-1)\cdot 1024$
	Sub-total cost without contribution steps.	5,87	$\Omega(n)\cdot 1024$
	Sub-total cost with contribution steps.	17,97	$\Omega(n)\cdot 1024$

(cont. on next page)

Table 4.23. (cont.)

Round-2 USERS	**Contribution** #6 Verification of Leader's signature and integrity checking of the Commitment Vector.	3,4	-
	Contribution **#7** Decryption of received subsecret	1,3	-
	#8 Verification correctness of subsecrets by using commitment vector.	0,774	-
	#9 Each U_i sends its subsecret to other user of the group.		$\Omega(n^2) \cdot 512$
	Contribution #9 Encryption processes of subsecret of U_i. ((n-1) users send own subsecret to other (n-2) users).	7,6	$\Omega(n^2) \cdot 1024$
Key Establishment	**Contribution** #10 Decryption process of received encrypted subsecrets by U_i	4	-
	#11 Verification of subsecret by using commitment vector	2	-
	#12 Lagrange Interpolation and common secret key establishment	0,035	-
	Sub-total cost without contribution steps.	3,2	$\Omega(n^2) \cdot 512$
	Sub-total cost with contribution steps.	19,5	$\Omega(n^2) \cdot 1024$
	TOTAL COST WITHOUT CONTRIBUTION	9	$\Omega(n^2) \cdot 512$
	TOTAL COST WITH CONTRIBUTION	37,5	$\Omega(n^2) \cdot 1024$

CHAPTER 5

EFFICIENCY ANALYSIS

In this chapter, the protocols studied in the Chapter 4 will be examined in terms of efficiency properties quantified as computation complexity, communication complexity and round complexity as defined in Section 1.1.

The computation complexity and the communication complexity denote the amount of computations of each participant to obtain common secret key and the amount of messages of each participant to other ones, respectively. Round complexity is number of round of the protocol. A round means one messaging session in which every party can cast messages to others but all at once. Minimizing round complexity is very important *challenge* in designing key establishment protocols as well as reducing other complexities.

5.1. Efficiency Analysis of Key Agreement

The protocol uses topology of an open network and needs only two rounds independent from the participant number. All parties carry on the protocol by achieving same computations during the both two rounds whereas the parties in the second protocol which uses verifiable secret sharing (VSS) carry on the computations depending on computations of their leader within the group.

In Table 5.1, the first column represents key agreement phases with their related computations. As to the first row, it represents one user's calculations and the whole users' calculations in the protocol respectively.

Table 5.1. Efficiency Analysis of Key Agreement Protocol.

	Costs for each U_i	**Costs for all $U_1, U_2, \dots, U_n$**
Setup $Y_i = a_i P$ $Cert_i = (Y_i, ID_i)$by CA	Assume that all entities of the group have private and public key and certification which is signed by CA.	
Round-1 $T_i = x_i Y_i$	1 scalar multiplication	n scalar multiplications
$Sig_i\,(T_i)$	1 signing algorithm	n signing algorithms
Broadcast T_i and $Sig_i\,(T_i)$	2 broadcast messages	$2n$ broadcast messages
Round-2 $Verify_i\,(T_i)$	$n-1$ verifying operations	$n(n-1)$ verifying operations
$X_i = e((Y_{i+1} + T_{i+1}), (Y_{i+2} + T_{i+2}) - (Y_{i-1} + T_{i-1}))^{(a_i + a_i x_i)}$	4 point additions 1 integer multiplication 1 integer addition 1 point exponentiation 1 Weil pairing operation	$4n$ point additions n integer multiplications n integer additions n point exponentiations n Weil pairing operations
$Sig_i\,(X_i)$	1 signing algorithm	n signing algorithms
Broadcast X_i and $Sig_i\,(X_i)$	2 broadcast messages	$2n$ broadcast messages
Key Agreement		
$Verify_i\,(X_i)$	$n-1$ verifying algorithms	$n(n-1)$ verifying algorithms
$K_i = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i + a_i x_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \dots X_{i-2}$	2 point additions 1 scalar multiplication 1 integer multiplication 1 integer addition 1 point exponentiation 1 Weil pairing operation $n-1$ point multiplications	$2n$ point additions n scalar multiplications n integer multiplications n integer additions n point exponentiations n Weil pairing operations $n(n-1)$ point multiplications
$kdf(K_n \parallel U_1 \parallel U_2 \parallel \dots \parallel U_n)$.	1 key derivation function algorithm	n key derivation function algorithms

In the light of Table 5.1, the total efficiency cost in terms of computation, communication and round complexity including the whole parties during Round-1, Round-2 and Key Agreement phases are available in Table 5.2.

Table 5.2. Total Efficiency Cost of Group Key Agreement.

Computation Complexity	***Communication Complexity***	***Round Complexity***
• $2n$ scalar multiplications • $2n$ Weil pairings • $2n$ signing algorithms • $2n^2 - 2n$ signature verifying algorithms • $6n$ point additions • $n^2 - n$ point multiplications • $2n$ point exponentiations • $2n$ integer (multipresicion) additions • $2n$ integer (multipresicion) multiplications • n key derivation function algorithms	• $4n$ broadcast messages	• 2 rounds

As seen in Table 5.2, all computation costs except verifying algorithm and point multiplication are linear depending on user number of the protocol. Verifying algorithm of signature during both two rounds increases costly total computation overhead of the protocol with $(2n^2 - 2n)$ computation complexity of signature verification. Implementation results in Section 4.2.4 prove this fact with time measures. BLS Signature Algorithm is chosen to sign and verify the public messages for the implementation of the Key Agreement Protocol. ElGamal Signature Algorithm is chosen for the Key Distribution Protocol in Section 4.3, however time measures of ElGamal Signature is more efficient than BLS Signature. Hence, an efficient signature algorithm should be preferred in this protocol due to $(2n^2 - 2n)$ computatin complexity of signature verification.

Another computation cost which causes high overhead is point multiplication in Round-2 with $n^2 - n$ complexity. Therefore, the original protocol (Lin et al. (2006)) isn't applicable for user number more than 100. Since the proposed protocol in Section 4.2 adds signing and verifying costs, it works efficiently till user number is 20.

Communication and round complexity are stable with $4n$ and 2 costs respectively.

5.2. Efficiency Analysis of Key Distribution Protocol

All participants in the protocol make point to point sending between themselves with only two rounds. As so the first protocol in Section 4.2, this one needs only two rounds independent from the participant number, too. But the distinction is that computation of each party is depending on computations of his/her leader within the group. In the first round, only the leader performs the computations as he knows the secret will be established. On the other hand, parties of the protocol achieve computations in the rest of the protocol phases as seen in the Table 5.3.

Another distinction from the first protocol, computation cost depends on not only the participant number n, but also the polynomial degree t. The reason of that situation is relation between the (n, t) parameters (Shamir, (1979), Shannon (1949)).

The first column represents key distribution phases with their related calculations. As to the first row, it represents leader's calculations, one user's calculations and the whole users' calculations in the protocol respectively.

Table 5.3. Efficiency Analysis of Key Distribution Protocol.

		Leader	**Costs for each U_i**	**Costs for all $U_1, U_2, \dots, U_n$**
Setup	$Cert_i = (Y_i, ID_i)$by CA	Assume that all entities of the group have private and public key and certification which is signed by CA. Obtains and exchange $Cert_i$		
Round-1	$f(ID_i) = a_0 + a_1 ID_i + a_2 (ID_i)^2 + \dots + a_t (ID_i)^t ID_i$	n calculations for subsecrets $f(ID_i) = s_i$ **:** nt addtions $n(t-1)$ exponentitions nt multiplications		
	$C_j = [C_0 = g^{a_0}, C_1 = g^{a_1}, .., C_t = g^{a_t}]$	$t+1$ exponentiations for DLog commitments		

(cont. on next page)

Table 5.3. (cont.)

Round-1 (cont.)	$Sig_{Leader}[C_j]$ $Enc_{Y_i}[s_i\ ]$	1 signing algorithm for commitment vector n encryption algorithms for each s_i n point to point sendings messages 2 broadcast messages for C_j and$Sig_{Leader}[C_j]$		
Round-2	$Verify_{Leader}[C_j]$ $Dec_{x_i}\left[Enc_{Y_i}[s_i\ ]\right] = s_i$ $g^{s_i}(mod\ p) = \prod_{j=0}^{t} C_j^{i^j}$ $Enc_{Y_j}[s_i]$		1 verifying operation for commitment vector 1 decryption algoritms for s_i $2t+3$ exponentiations and t multiplications for subsecret verification $n-1$ encryption algorithms for s_i $n-1$ point to point sending messages to each U_j	n verifying operations for commitment vector n decryption algoritms for s_i $n(2t+3)$ exponentiations and nt multiplications for subsecret verification n^2-n encryption algorithms for each s_i n^2-n point to point sending messages

(cont on next page)

Table 5.3. (cont.)

Key Establishment (Round-2)	$Dec_{x_i}\left[Enc_{Y_i}[s_m]\right] = s_m$ $g^{s_m}(mod\ p) = \prod_{j=0}^{t} C_j^{m^j}$ $f(0) = \sum_{U_i \in W}^{t+1} L_i(0) s_i$		$n-1$ decryption algoritms for received s_m $(n-1)(2t+3)$ exponentiations and $(n-1)t$ multiplications for each subsecret verification 1 lagrange interpolation includes; t^2+2t additions and $4t(t+1)$ multiplications	n^2-n decryption algoritms for received s_m $n(n-1)(2t+3)$ exponentiations and $n(n-1)t$ multiplications for each subsecret verification n lagrange interpolation includes; $n(t^2+2t)$ additions and $4tn(t+1)$ multiplications

In the light of Table 5.3, total efficiency cost in terms of computation, communication and round complexity including the whole parties during round-1, round-2 and key establishment phases are available in Table 5.4.

Table 5.4. Total Efficiency Cost of Group Key Distribution Protocol.

	Computation Complexity	***Communication Complexity***	***Round Complexity***
Leader	• n encryption algorithms • 1 signing algorithm • nt integer (multiprecision) additions • nt integer (multiprecision) multiplications • $n(t-1)+t+1$ integer (multiprecision) exponentiations	• n point to point sending messages	• 1 round
(n,t) parties	• n verification algorithms • n^2-n encryption algorithms • n^2 decryption algorithms	• n^2-n point to point sending messages	• 1 round

(cont. on next page)

Table 5.4. (cont.)

(n, t) ***parties***	• $n(t^2 + 2t)$ integer (multiprecision) additions • $n^2(2t + 3)$ integer (multiprecision) exponentiations • $n(4t^2 + nt + 4)$ integer (multiprecision) multiplications		
Total	• n^2 encryption algorithms • 1 signing algorithm • n verification algorithms • n^2 decryption algorithms • $n(t^2 + 3t)$ integer (multiprecision) additions • $nt(4t + n + 1) + 4n$ integer (multiprecision) multiplications • $n^2(2t + 3) + n(t - 1) + t + 1$ integer (multiprecision) exponentiations	• n^2 point to point sending messages and 2 broadcast messages	• 2 rounds

As seen in Table 5.4, most of the computation costs depend on not only user number but also threshold value t and therefore when user number increases t size will increase and computation overhead will become very costly.

Also complexity of encryption and decryption are n^2 since there are point to point different encrypted messages sent unlike the first protocol (there is one public message generated by one user.) ElGamal Encryption Algorithm is chosen to encrypt and decrypt the messages in the implementation of Section 4.3.4. There are efficient time measures for ElGamal Algorithm.

As to communication cost, it is costly with n^2 complexity.

Finally, round complexity is stable with only 2 rounds independent from user number same as the first protocol.

The proposed protocol in Section 4.3 works efficiently till user number is 20.

5.3. Group Key Agreement Protocol vs. Group Key Distribution Protocol

The following Table 5.5 presents a comparison of efficiency of two protocols. They are evaluated according to results of computation, communication and round complexities presented in Table 5.2 and 5.4.

Table 5.5. Efficiency Comparision.

		Key Agreement	***Key Distribution***
Computation Complexity	Weil Pairing	$2n$	-
	Encryption Algorithm	-	n^2
	Decryption Algorithm	-	n^2
	Signing Algorithm	$2n$	1
	Verification Algorithm	$2(n^2 - n)$	n
	Key Derivation Function Algorithm	n	-
	Scalar Multiplication	$2n$	-
	Point Addition	$6n$	-
	Point Multiplication	$n^2 - n$	-
	Point Exponentitation	$2n$	-
	Integer (multiprecision) Addition	$2n$	$n(t^2 + 3t)$
	Integer (multiprecision) Multiplication	$2n$	$nt(4t + n + 1) + 4n$
	Integer (multiprecision) Exponentiation	-	$n^2(2t + 3) + n(t - 1) + t + 1$
Communication Complexity	Message number sent	$4n$	$n^2 + 2$
Round Complexity	Round number	2	2

Discussion:

- Computation cost: Key agreement protocol is based on elliptic curves. On the other hand key distribution protocol is based on multiprecision arithmetic. Computations of key agreement is looks efficient than key distribution. Main reason is that costs of key agreement is dependent on only user number whereas key distribution is dependent on not only user number, but also polynomial degree t which increases when user number increase. However, in the implementation results for four users in Section 4.2.4 and 4.3.4, although the second protocol is more efficient than the first one, when user number increases efficiency of the second protocol will decrease comparing to first one. Proving this deduction is a future task.

  Computation which causes the most overhead for the first protocol is verification algorithm with n^2 complexity. In the second protocol, encryption and decryption algorithms with n^2 complexity causes the most overhead for the total computation cost.
- Communication cost: Due to only one public message is broadcasted by one user, key agreement has advantage on this requirement as well. In in key distribution, there are n public messages created by per user. So the communication complexity is n^2.
- Round complexity: Both protocols are effective on this requirement providing only 2 rounds independent of user number.

CHAPTER 6

CONCLUSION

Motivation of the study is to establish a common secret key over an open network for a group of user to be used then symmetrical secure communication between them. A *group key establishment protocol* (GKE) is responsible for the establishment of a group key. It has numerous applications on group oriented scenarios such as secure teleconferencing, replicated servers, multi-user games, Near Field Communication (NFC), etc.

There are two methods of GKE protocol which are key agreement and key distribution. Key agreement is a mechanism whereby the parties jointly establish a common secret. As to key distribution, it is a mechanism whereby one of the parties creates or obtains a secret value and then securely distributes it to other parties. In this study, both methods is applied and analyzed in two different GKE protocols. Desirable properties of a GKE are *security* and *efficiency*. Security is attributed in terms of preventing attacks against passive and active adversary. Efficiency is quantified in terms of computation, communication and round complexity. When constructing a GKE, the *challenge* is to provide security and efficiency according to attributed and quantified terms. Two main cryptographic tools are selected in order to handle the defined *challenge*. One of them is bilinear pairing which is based on elliptic curve cryptography and another is verifiable secret sharing which is based on multiparty computation.

Before the GKE proceeds, secure communication between entities of the protocol should be ensured. Entities can apply PKI-based infrastructure or ID-based infrastructure mentioned in Section 3.1 depending on protocol requirements. ID-based infrastructure fits two party key establishment protocols as secure communication methodology. Especially, key establishment on Email protocols, Voice over Internet Protocol (VoIP) and Session Initiation Protocol (SIP) use ID-based cryptography. However, in a group oriented case like this study, it is the best choice to apply a PKI-based infrastructure.

The first protocol model (key agreement) of the study is based on Lin et al.'s work (Lin et al (2006)). Bilinear pairing is utilized as cryptographic tool in order to be

solution of the *challenge*. Bilinear pairing is a function which provides a map from two cyclic groups to another cyclic group on an elliptic curve. In the beginning in order to ensure secure communication during the protocol, each party applies a certificate authority (CA) and exchange his/her certificate with other legitimate parties. No party has the possession of the common secret when starting to the protocol unlike the second protocol. The common secret is established by similar calculations of each party jointly after *two rounds*. In each round, each party generates his/her public message, signs that message and broadcast to others. In the end, each party verifies and combines the public messages received from others with his/her static and ephemeral private keys to establish the common secret. Security of the public messages depends on the Diffie-Hellman assumption and Bilinear Diffie-Hellman assumption defined in the section 2.1.3. Integrity and authenticity of those messages is ensured via signature algorithms on them. Some security inspection is evaluated in section 4.2.3 for attributed security aspects.

Efficiency analysis is performed for n participants as follows;

- Computational complexity: $2n$ Weil pairing, $2n$ singning algorithm, $2(n^2 - n)$ verification algorithm, n key derivation function algorithm, $2n$ scalar multiplication, $6n$ point addition, $2(n^2 - n)$ point multiplication, $2n$ point exponentiation, $2n$ integer (multiprecision) addition and $2n$ integer (multiprecision) multiplication forms computational complexity of the protocol.
- Communication complexity: $2n$ message number for public messages sent and $2n$ for signed public messages forms $4n$ communication complexity of the protocol.
- Round complexity: 2 round number forms round complexity of the protocol.

An implementation for $n = 4$ user with 512-bit prime q, 160-bit prime p and curve $y^2 = x^3 + x$ over F_q parameters is developed under C programming. Implementation platform has PBC, GMP and OpenSSL libraries with Intel® Core™ 2 Duo 2.0 GHz for CPU, 4 GB for RAM and Ubuntu 12.10 machine configuration. According to implementation results, original protocol (Lin et at., (2006)) needs 8,8 milliseconds computation time with 8192-bit length communication cost performed by one user for four-user group in totality. In this work, BLS Short Signature algorithm is added on public messages T_i and X_i of the original protocol in order to provide their message authentication and integrity. BLS Signature algorithm can be replaced with

another signature algorithms such as RSA, ElGamal, Schnorr, etc. depending on application the protocol will be applied on. Result of one signing computation of BLS signature is approximately 10 milliseconds (one SHA-512 hashing algorithm, one point mapping algorithm, one scalar multiplication) and one verification computation (one SHA-512 hashing algorithm and 2 bilinear pairing) is approximately 4,6 milliseconds performed by one user.

Total computation time of the proposed protocol performed by one user in four-user group size is approximately 56 milliseconds with 16384-bit length communication cost during the whole protocol.

The second protocol model (key distribution) is based on Feldman's work (Feldman (1987)). The verifiable secret sharing is utilized as cryptographic tool in order to be solution of the *challenge*. In the beginning; for secure communication during the protocol, each party applies a certificate authority (CA) and exchange his/her certificate with other legitimate parties. In this way, it is adapted a secure, closed group communication. The leader has the possession of the common secret when starting to the protocol and then, he distributes this secret to other parties using verifiable secret sharing technique. The protocol is rendered to be applied over open channel by adding encryption algorithm to sending messages of each user instead of using secure channel as the original protocol. All point to point sending messages are encrypted (secrecy). Also, signature algorithm on publicly known commitments is added in order to prevent modifications by malicious parties (message integrity and authentication). It takes only two rounds. In the first round, only the leader performs the calculations in order to distribute pieces of the common secret (They are called as subsecrets in the thesis.). In the second round, participants carry out the computations to establish the common secret. Security of each sending messages depends on the encryption algorithm (secrecy). Consistency of the messages from the sender can be verified via commitments defined in the section 2.1.4 (correctness). The protocol is information theoretically secure even when the adversary has unlimited computing power. So, the adversary simply does not have enough information to break the encryption unless he has $t + 1$ values (Shannon (1949)).

Detailed efficiency research is performed. According to those results for n participants; there are;

- n^2 encryption algorithm, n^2 decryption algorithm, 1 signing algorithm, n verification algorithm, $n(t^2 + 3t)$ integer (multiprecision) addition, $nt(4t + n + 1) + 4n$ integer (multiprecision) multiplication and $n^2(2t + 3) + n(t - 1) + t + 1$ integer (multiprecision) exponentiation forms computational complexity.
- $n^2 + 2$ message number sent forms communication complexity.
- 2 round number forms round complexity.

An implementation for $n = 4$ user and a leader with 1024-bit prime p and 512-bit prime q parameters is developed under C programming. Implementation platform has GMP and OpenSSL libraries with Intel® Core™ 2 Duo 2.0 GHz for CPU, 4 GB for RAM and Ubuntu 12.10 machine configuration. According to implementation results, original protocol (Feldman (1987)) needs 9 milliseconds computation time with 7168-bit length communication cost in totality. 5,85 milliseconds of 9 is performed by the leader and the rest 3,5 milliseconds is performed by each user in 4-user group size.

In this work, ElGamal Signature algorithm is added on publicly known commitments of the original protocol in order to provide its message authentication and integrity. ElGamal signature algorithm can be replaced with another signature algorithms such as RSA, Schnorr, BLS Short Signature etc. depending on application the protocol will be applied on. Result of one signing computation of ElGamal Signature is approximately 1,6 milliseconds (one SHA-512 hashing algorithm, one exponentiation, one multiplication, one addition) performed by the leader and one verification computation (one SHA-512 hashing algorithm, two exponentiation, one multiplication) is approximately 3,4 milliseconds performed by one user. It is quite efficient than BLS signature.

Also, ElGamal encryption algorithm is applied to each point to point sending messages of the protocol in order to be adapted on open channel. Each encryption time is approximately 2,5 milliseconds (two exponentiation and one multiplication) and each decryption time for per user is approximately 1,3 milliseconds (one exponentiation and one multiplication). Total computation time of the proposed protocol performed by one user in four-user group size is approximately 37,5 milliseconds with 22528-bit length communication cost during the whole protocol. 17,5 milliseconds of 37,5 is performed by the leader and the rest 19,5 milliseconds is performed by one user in 4-user group size. ElGamal encryption algorithm can be replaced with another encryption algorithms

such as RSA, Elliptic Curve Cryptography, etc. depending on application the protocol will be applied on.

Some deduction of implementation results as the following;

- Computation cost: Key agreement protocol is based on elliptic curves. On the other hand key distribution protocol is based on multiprecision arithmetic. Original form of the both protocols have approximately same computation time (9 milliseconds). But in the second protocol, this cost is shared between leader (5,85 milliseconds) and other users (3,5 milliseconds for each).

    - Time measures of computations in contributed version (17,5 milliseconds by leader and 19,5 milliseconds by per user) of key distribution are more efficient than key agreement (56 milliseconds by per user) in four-user group size.

    - Main reason which causes inefficiency for key agreement's contributed version is signature algorithm. There are 2 signing and $2(n-1)$ verification algorithm performed by each user in the protocol. Due to one signing is 10 milliseconds and one verification is 4,6 milliseconds performed by one user, 47 milliseconds time cost for BLS Short Signature is performed by one user in totality. If ElGamal Signature algorithm was preferred instead of BLS Signature, there would be 23,6 milliseconds time cost for signature algorithm and 32,6 milliseconds for entire protocol computation time performed by one user.

    - Although, key distribution protocol's implementation result of computation time is more efficient than the first protocol, its computation cost depends on not only user number n but also threshold number t. Hence, increase of user number causes increase of threshold value and therefore there would be longer execution time.

- Communication cost: Communication overhead of key distribution is very costly than key agreement. 16384-bit length communication messages forms total the communication cost during the whole key agreement protocol due to $4n$

communication complexity. There are 2 public messages as a point with 512-bit length for each coordinate, generated by one user in each two rounds. As to key distribution protocol, 22528-bit length communication messages forms the total communication cost during the entire protocol due to $n^2 + 2$ communication complexity. There are n public messages created by per user. They are 1024-bit length encrypted messages between the protocol entities and one 3*1024-bit length commitment elements and one 1024-bit length signed commitment generated by leader. It is main drawback of the key distribution protocol since communication complexity is n^2.

- Round complexity: Both protocols are effective on this requirement providing only 2 rounds independent of user number.

Bilinear pairing is the most trending tool in key establishment protocol after 2000 (Joux (2000)). Ease computation, communication and round complexity in key establishment protocols. At the present time, research interest for group key establishment protocol are focused on this tool for its efficiency.

Key agreement protocol can be applied small size group oriented application on condition that usage of efficient signature scheme since there is n^2 verification algorithm complexity.

Verifiable secret sharing is applied in 1987 by Feldman. It was elegant tool comparing that old times. But, it is not practical in real life group key establishment applications. Although its results of computation time look more efficient than the first protocol for small group size, it has very costly communication overhead. When group size increases, computation time will be getting inefficient, too due to dependency of (n, t) parameters. It fits in application that key escrow, secure storage, collective control, secure multiparty computation and e-voting.

Table 6.1 represents general comparison of these two protocols.

Table 6.1. Key Distribution vs. Key Agreement.

	Key Distribution	***Key Agreement***
Key Generation	Center (By Leader)	Each Member's Contribution jointly
Crypto Primitive	t-Degree Polynomial, Discrete Logarithm, Secret Key Encryption,	Extended Diffie-Hellman, Bilinear Pairing Function
Communication	Point to point	Broadcast
Computation Overhead	Large (not similar)	Large(similar complexity)
Group Size	<20	<20
Equal Contributory	No (Dependent on leader)	Yes (jointly established)
Number of Round	2	2
Open network	Yes	Yes
Group modification	Only $n = 2t - 1$ relation preserved	No
Allocation of each party	Not static	Static and Ring structure (should preserve their location)
Verifiability of received public messages	DLog Commitments	No
Security against active adversary and passive adversary	Provided (Secrecy by encryption, correctness by commitments)	Provided (Diffie-Hellman assumption and Bilinear Diffie-Hellman assumption)

Open Problems and Future Tasks:

The first protocol model (key agreement),

Static participant number and allocation is an important drawback of the protocol. Also, there is an open problem defined by this study, which is;

- If any party establishes the wrong secret, he/she will realize the failure only when the communication with the common secret starts via symmetrical cryptography. He/She will encrypt messages with wrong common secret and the recipient cannot decrypt it. If any failure occurs with decryption, the recipient sends error message and the protocol restarts by picking new ephemeral secret key.
- It is an *open problem* and the protocol needs a cryptographic tool which provides the *verifiability*. So, users can be sure that the established secret is consistent before the communication with common secret starts.

The second protocol model (key distribution),

Security analysis of the protocol is just defined. It needs formal security inspections in order to prove its security over open channel in the presence of passive and active adversary. Since this protocol's computation cost depends on both user number n and threshold number t, increase of user number triggers very long execution time comparing to the first protocol. It can be seen in the results of efficiency analysis in Chapter 5. In order to prove it, an implementation is needed to be developed for greater group size of both protocols as future tasks.

REFERENCES

Adams, C. and S. Lloyd (2003). Understanding PKI: Concepts, Standards, and Deployment Considerations, 2nd Edition, Addison-Wesley.

Afergan, M., J. Wein, and A. LaMeyer (2005). Experience with Some Principles for Building an Internet-Scale Reliable System. In WORLDS'05, pp. 1-6.

Allen, B. (2008). Implementing Several Attacks on Plain ElGamal Encryption. MS. Thesis, Iowa State University, Mathematics.

ANSI/IEEE (1999). Wireless LAN media access control (MAC) and physical layer (PHY) specifications. ANSI/IEEE Std. 802.11:1999 (E) Part 11, ISO/IEC 8802-11.

Ateniese, G., M. Steiner, and G. Tsudik (1998). Authenticated group key agreement and friends. *5th Conference on Computer and Communications Security*, pp. 17–26. ACM Press.

Ateniese, G., M. Steiner, and G. Tsudik (2000). New multi-party authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications, 18(4)*, pp. 628–639.

Barua, R, R. Dutta, P. Sarker (2003). Extending Joux's protocol to multi party key agreement. *Proceedings of Indocrypt '03*, LNCS 2904, pp.205-217.

Bellare M. and P. Rogaway (1993). Entity authentication and key distribution. CRYPTO '93 *Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, Springer-Verlag New York, Inc. New York, NY, USA, pp. 22-249.

Bellare M. and P. Rogaway (1995). Provably secure session key distribution: the three party case. STOC '95 *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing* , New York, NY, USA, pp. 57-66.

Bellare, M., D. Pointcheval, and P. Rogaway (2000). Authenticated key Exchange secure against dictionary attacks. EUROCRYPT'00 *Proceedings of the 19th international conference on Theory and application of cryptographic techniques* , Springer-Verlag Berlin, Heidelberg, pp. 139-155.

Berkovits, S. (1991). How to Broadcast a Secret. EUROCRYPT'91 *Proceedings of the 10th annual international conference on Theory and application of cryptographic techniques*, Springer-Verlag Berlin, Heidelberg, pp. 535-541.

Blakely, G. R. (1979). Safeguarding cryptographic keys. *Proceedings of the National Computer Conference, Volume 48*, pp 313-317.

Blake-Wilson, S. and A. Menezes (1997). Security proofs for entity authentication and authenticated key transport protocols employing asymmetric techniques. *In proceedings of the 5th International Workshop on Security Protocols*, Paris, France, Springer-Verlag, pp. 137-158.

Blake-Wilson, S. and A. Menezes (1999). Authenticated Diffie-Hellman key agreement protocols. SAC '98 *Proceedings of the Selected Areas in Cryptography*, Springer-Verlag London, UK, pp. 339–361.

Blake-Wilson, S., D. Johnson, and A. Menezes (1997). Key Agreement Protocols and their Security Analysis. *Proceedings of the 6th IMA International Conference on Cryptography and Coding*, Springer-Verlag London, UK, pp. 30-45.

Bleichenbacher, D. (1998). Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS #1. In Advances in Cryptology, CRYPTO'98, LNCS vol. 1462, pp. 1-12.

Boldyreva, A. (2003). Efficient threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. Public Key Cryptography – PKC 2003, Lecture Notes in Computer Science, pp. 31–46.

Boneh, D. (1998). The Decision Diffie-Hellman Problem, *Proceedings of the Third International Symposium on Algorithmic Number Theory*, Springer-Verlag, pp. 48–63.

Boneh, D. and M. K. Franklin (2001). Identity-Based Encryption from the Weil Pairing. CRYPTO '01 *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, Springer-Verlag London, UK, pp. 213-229.

Boneh, D., B. Lynn, and H. Shacham (2001). Short signatures from the Weil pairing. ASIACRYPT '01 Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, Springer-Verlag London, UK, pp. 514-532.

Boyko, V., P. MacKenzie, and S. Patel (2000). Provably secure password authenticated key exchange using Diffie-Hellman. EUROCRYPT'00 *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, Springer-Verlag Berlin, Heidelberg, pp. 156-171.

Bresson, E. and D. Catalano (2004). Constant Round Authenticated Group Key Agreement via Distributed Computation. *Proceedings of* PKC'04, *Volume 2947 of* LNCS, pp. 115-129.

Bresson, E., O. Chevassut, D. Pointcheval (2001). Provable authenticated group Diffie-Hellman key exchange. *Proceedings of 8th ACM Conference on Computer and Communications Security*, pp. 255-264.

Bresson, E., O. Chevassut, D. Pointcheval (2002). Dynamic group Diffie-Hellman key exchange under standard assumptions. *Proceedings of* EUROCRYPT '02 *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, Springer-Verlag London, UK, pp. 321-336.

Burmester, M. and Y. Desmedt (1995). A secure and efficient conference key distribution system. *Proceedings of Workshop on the Theory and Application of Cryptographic Techniques*, LNCS 950, pp.275-286.

Cheng, J. C. and C. S. Laih (2009). Conference Key Agreement Protocol with Non Interactive Fault-Tolerance over Broadcast Network. *International Journal of Information Security, Volume 8 Issue 1*, Springer-Verlag Berlin, Heidelberg, pp. 37-48.

Chor, B., S. Goldwasser, S. Micali, and B. Awerbuch (1985). Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. SFCS '85 Proceedings *of the 26th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Washington, DC, USA, pp. 383-395.

Damgard, I. (1999). Commitment schemes and zero-knowledge protocols. In Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, Denmark, volume 1561 of LNCS, pages 63-86.

Diffie, W. and M. Hellman (1976). New directions in cryptography. IEEE Transactions on Information Theory. 22 (6), pp. 644–654.

ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transaction on, IEEE Information Theory Society* 31 (4), pp. 469–472.

Feldman, P. (1987). A practical scheme for non-interactive verifiable secret sharing. SFCS '87 *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Washington, DC, USA, pp. 427-437.

FIPS 186 (1994). Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186, National Institute of Standards and Technology, 1994.

FIPS PUB 180-2, (2002). Secure Hash Standard. Federal Information Processing Standards Publications. Available from http://csrc.nist.gov/publications/fips.

Gasca, M. and T. Sauer (2000). Polynomial interpolation in several variables. Advances in Computational Mathematics, Volume 12, Issue 4, pp. 377-410.

General Packet Radio Services (GPRS) (2002). Service Description (Stage 2), TS 122 060, ETSI; 2002.

Gennaro, R., S. Jarecki, H. Krawczyk, and T. Rabin (2007). Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. Journal of Cryptology, 20(1) pp. 51-83.

Hadzilacos, V. and S. Toueg (1994). A modular approach to fault-tolerant broadcast and related problems. Technical report, Cornell University, Computer Science, May 1994.

Harn, L. and C. Lin (2010). Authenticated Group Key Transfer Protocol Based on Secret Sharing. *Computers, IEEE Transactions on* , vol.59, no.6, pp.842,846.

Huang, K. H., Y. F. Chung, H. H. Lee, F. Lai, and T.S. Chen (2009). A Conference Key Agreement Protocol with Fault-Tolerant Capability. *Computer Standards & Interfaces, Volume 31 Issue 2*, Elsevier Science Publishers B. V. Amsterdam, The Netherlands, The Netherlands, pp. 401-405.

Ingemarsson, I, D. T. Tang, and C. K. Wong (1982). A conference key distribution system. *IEEE Transactions on Information Theory, Volume 28, Issue 5*, pp. 714-720.

ISO/IEC 18033-2 (2006). Information technology - Security techniques - Encryption algorithms - Part 2: Asymmetric ciphers. Ed. Victor Shoup, 2006.

Jho, N.-S., M.-H. Kim, D. W. Hong, and B.-G. Lee (2007). Multiparty Key Agreement using Bilinear Map. *ETRI Journal*, pp. 439-439.

Johnson, A. M. and P. S.Gemmell (2002). Authenticated key exchange provably secure against the man in the middle attack. *Journal of. Cryptography, Volume 15*, Springer-Verlag, pp. 139-148.

Joux, A (2000). A one round protocol for tripartite Diffie-Hellman. ANTS-IV *Proceedings of the 4th International Symposium on Algorithmic Number Theory*, Springer-Verlag London, UK, pp. 385-394.

Karagodin, A. M. (2005). Public Key Infrastructures Enabled Services. KORUS 2005, pp. 989-994.

Kate, A. and I. Goldberg (2009). Distributed Key Generation for the Internet. In IEEE ICDCS'09, pp. 119-128.

Laih, C. S., J. Y. Lee, and L. Harn (1989). A New Threshold Scheme and Its Application in Designing the Conference Key Distribution Cryptosystem. *Information Processing Letters, Volume 32 Issue 3*, Elsevier North-Holland, Inc. Amsterdam, The Netherlands, The Netherlands, pp. 95-99.

Lamport, L., R. Shostak, and M. Pease (1982). The Byzantine generals problem. ACM Transactions on Programming Languages and Systems, vol. 4, pp. 382-401.

Li, C.-H. and J. Pieprzyk (1999). Conference key agreement from secret sharing. *Proceedings of ACISP'99, Volume 1587 of* LNCS, 1999, pp. 64-76.

Lin, C.-H., H.-H. Lin, and J.-C. Chang (2006). Multiparty Key Agreement for Secure Teleconferencing. *IEEE Conference on System, Man, and Cybernetics*, SMC'06, pp. 3702-3707.

Makri, E. and E. Konstantinou (2011). Constant round group key agreement protocols: A comparative study. *Computers & Security, Elsevier, Volume 30, Issue 8*, November 2011, pp. 643-678.

McCurly, K. S. (1990). The discrete logarithm problem. *In Proceedings of Symposia in Applied Mathematics*, AMS press, pp. 49-74.

Menezes, A., P. Oorschot, and S. Vanstone (1996). Handbook of Applied Cryptography, First Edition, CRC Press Inc, 1996.

Pedersen, T. (1991). Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. CRYPTO '91 *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, Springer-Verlag London, pp. 129-140.

Pereira, O. and J.-J. Quisquater (2001). A security analysis of the Cliques protocol suites. In *Computer Security Foundations Workshop*, IEEE Computer Society Press, pp. 73–81.

Perkins, C. (2001). Ad hoc networking. Addison-Wesley, 2001.

Rivest, R., A. Shamir and L. Adleman (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM* 21 (2), pp. 120–126.

RSA Laboratories (2006). PKCS #5 v2.1: Password-Based Cryptography Standard.

Sáez, G. (2003). Generation of Key Predistribution Schemes Using Secret Sharing Schemes. *Discrete Applied Mathematics - Special issue: International workshop on coding and cryptography, Volume 128, Issue 1*, Elsevier Science Publishers B. V. Amsterdam, The Netherlands, The Netherlands, pp. 239 – 249.

Schnorr, C. P. (1989). Efficient Identification and Signatures for Smart Cards. CRYPTO '89 *Proceedings on Advances in cryptology*, Springer-Verlag New York, Inc. New York, NY, USA, pp. 239-252.

Shamir, A. (1979). How to share a secret. *Communications of the ACM, Volume 22, Issue 11*, ACM, New York, NY, USA, pp. 612-613.

Shamir, A. (1984). Identity-Based Cryptosystems and Signature Schemes. *In Advances in Cryptology* CRYPTO'84, pp 47-53.

Shannon, C. E. (1949). Communication Theory of Secrecy Systems. *Bell System Technical Journal* (USA: AT&T Corporation) 28 (4), pp. 656–715.

Stallings, W. (2006). Cryptography and Network Security Principles and Practices. Fourth edition, Prentice Hall, 2006.

Steiner, M., G. Tsudik, and M. Waider (1996). Diffie-Hellman Key Distribution Extended to Group Communication. CCS '96 *Proceedings of the 3rd ACM conference on Computer and communications security*, New York, NY, USA, pp. 31-37.

Stinson, D. (1995). Cryptography: Theory and Practice. First Edition, CRC Press Inc., 1995.

Tseng, Y.-M. (2007). A resource-constrained group key agreement protocol for imbalanced networks. *Computers & Security*, Elsevier, Volume 26, Issue 4, June 2007, pp. 331-337.

Tzeng, W. G. (2002). A Secure Fault-Tolerant Conference Key Agreement Protocol. *IEEE Transactions on Computers, Volume 51 Issue 4*, IEEE Computer Society Washington, DC, USA, pp. 373-379.

Vatră, N. (2009). Public key infrastructure overview. Scientific Studies and Research, Series Mathematics and Informatics, no. 2, vol. 19, 2009, pp.471 – 478.

Yoa, A. C. (1982). Protocols for secure computations. SFCS '82 *Proceedings of the 23rd Annual Symposium on Foundations of Computer Sciesnce*, IEEE Computer Society Washington, DC, USA, pp. 160-164.

Yongdae, K. (2001). Group Key Agreement. Lecture Notes. ICS 268, Lecture14, University of California, 2001.

Yuen, T. H. (2010). Contributions to Pairing-based Cryptography. University of Wollongong, School of Computer Science and Software Engineering, 2010.

WEB_1 (2013). PBC Library Home Page, http://crypto.stanford.edu/pbc/.

WEB_2 (2013). GMP Library Home Page, http://gmplib.org/.

WEB_3 (2013). OpenSSL Home Page, http://www.openssl.org/.

APPENDIX A

IMPLEMENTATION OF GROUP KEY AGREEMENT PROTOCOL

All implementation details and the every values of each variable in the related steps of Section 4.2.4 can be found and followed under Appendix A.

Table A.1 includes the input parameters. There are super singular curve E, 512-bit prime q, 160-bit prime p, and 512-bit (for each coordinate of point) generator $P \in E/F_q$.

Table A.1. Input Parameters.

Input Parameters
User number is 4 namely; U_1, U_2, U_3 and U_4. A super singular curve E defined by $y^2 = x^3 + x$ over F_q. $P \in E/F_q$ is a generator. The Weil pairing on the curve E/F_{q^2} is a mapping: $G_q \times G_q \rightarrow G_T$. $q = p \cdot h + 1$ where h is multiple of 12.
q=8780710799663312522437781984754049815806883199414208211028653399266475630880222957078625179422662221423155858769582317459277713367317481324925129998224791 h=120160122648911460793888213667405342048029544012513118229196151310472072893597045311028448021839065377867 76 p= 730750818665451621361119245571504901405976559617 P=[8173511322047916344408808106529574277334778842693556600580519618017213748909540300548540282437675522760000845925931053576935927991050163426073743235291144, 219258587910687079802892337387881927726037906057425586997906733320976051807545667795752611251050633614260803077670354173126882665006438099896518931966434 8]

Each user generates his/her static public-private key pair as seen in the Table A.2 for setup parameters. Static private keys are picked from 160-bit group order p and

public keys are 512-bit length (for each coordinate of the point). Then, he/she applies a legitimate Certification Authority (CA) such as VeriSign (WEB_4, (2013)) and exchanges his/her certificate with other participants. It is offline task of the protocol and public-private key pairs in the Table A.2 are calculated using PBC Library.

Table A.2. Setup Parameters.

Setup	
User U_1	$Y_1 = a_1 P$ a_1=360822344586300127099187863178723711012295013567 Y_1=[5985319997768049285218541686677812136628523385065923548703376967098329443527983502785409622920561499414539840970929476743603292317994925950571195534367809, 6020199897100177407568414829947876791705804870783041415603247095629693076994070081708035457607192326370751834981903793272779650818116682112728245904032292]
User U_2	$Y_2 = a_2 P$ a_2=288567973311584200176994578390869940543208619073 Y_2=[2351050740955673808920663408217485731482590738386264407512636250263260663435468216725299939201774630620755066403879864709066380859631799604525267064956755, 7494402258598578644856020650771003554574917789488378758235523739772768039990238838677907367841094767599577369580741776745826993101300363405562027609173 86]
User U_3	$Y_3 = a_3 P$ a_3=57558848928274421825129641653333750709066216329 Y_3=[4981932300130439214162032823527646153049136203784142520756816744366049589399829602432352340179676937635372596366473869330321844021232368816966158300588012, 3556770422965662881542399806024906812076901539587296340535162704259252331609545366985858689581007749818086733275005312512903587331997597848196293892418827]

(cont. on next page)

Table A.2. (cont.)

User U_4	$Y_4 = a_4 P$ a_4=599838879599011924079709024407036352993025876990 Y_4=[266452297869576707902913185868832442031417871854164366567864290292659 780322700506918862010615306108076154891912952967222460060904002198735534 8101400425168, 179961868061014168919115008915797476017529115393068163073163338179571985 540244521950821106032805501769444881282753384860152483458872059890839173 0383699068]

In Table A.3, each user picks ephemeral secret key x_i from 160-bit prime p and multiplies it with his/her public key Y_i. Approximate time for each user's $T_i = x_i Y_i$ computation is 3,600 milliseconds. T_i is 512-bit length. Specific time measures are available in the following table.

Table A.3. Public Message Computation on Round-1.

	Round-1 $T_i = x_i Y_i$
User U_1	$T_1 = x_1 Y_1$ x_1=481979732857326628012634319296289415681640637577 T_1=[654994789206384117853399715725936931707895469639923026512514921671275768319916209112457537867478573937682159249784870531986189081937377364344178538038254, 794642882047673763844349260850017837373133859795223166973273758413262377085795346840452862647053527916971580865656077196556916993255579369716118160434453] Total computation time of U_1 is 3,592 milliseconds.

(cont. on next page)

Table A.3. (cont.)

User U_2	$$T_2 = x_2 Y_2$$ x_2=311294406533681462247751145863399150516619783049 T_2=[875626208230727873265686780808292228990187810953579523982158591938384790278552949460354429388311973084709519824899467166723818595279472073123839797754923 8, 729966919155190704243056343106212794608157558550779018214370737380946511449881888567418758882554681348170289671411583407380210946711233302751967 2351101143] Total computation time of U_2 is 3,559 milliseconds.
User U_3	$$T_3 = x_3 Y_3$$ x_3=22643660495041221015576424259588560103808623345 9 T_3=[207435337592605059431186875211013446351390635826061318050291282325107124681292332105751972190274794011865556503556250382937165990963796900346197770817394 1, 225905865678677201745158187434898786415499897380247583820189195442971480128807770287341536062833350496044479202252097336543551750685409398723715 4093019296] Total computation time of U_3 is 3,753 milliseconds. .
User U_4	$$T_4 = x_4 Y_4$$ x_4=166673198623961988060656494023777974634863506700 T_4=[850025462950285335724040964864897525949008198861046276932002323115974479724739283306509570134452178206802645122147316275476721037747483146830056211157847 2, 465283493695616357173048486996587380839544858420652485723524977620012796534202017588714063291356415748116551113394951509528111287953226802943807 2497114211] Total computation time of U_4 is 3,493 milliseconds

In Table A.4, each users applies SHA-512 hashing algorithm from OpenSSL Library for T_{i}. It outputs 512-bit result. Approximate time for each user's SHA-512 hashing computation is 37 μ seconds. Specific time measures are available in the following table.

Table A.4. SHA-512 Algorithm on Round-1.

	Round-1 $H_{\mathrm{i}} = SHA - 512(T_i)$
User U_1	$H_1 = SHA - 512(T_1)$ H_1=01da74438ce8facd60f9e020a9fecd1389ff538f69cb75cbc9eae047fd5b4810f5a8b37c92d6f94550c58f5329b79d06a55c0c582028a361450ca0483a5f07b0 Total computation time of U_1 is 37 μ seconds.
User U_2	$H_2 = SHA - 512(T_2)$ H_2=075b5611a23ab98a22eb7d867b0833bcfec8fab239d30ee16c71cfebb7b71fd222c1442a9a5d9c83aaa05643ba2c5da10cbff4177edad83e6f534afe4a5bf690 Total computation time of U_2 is 37 μ seconds.
User U_3	$H_3 = SHA - 512(T_3)$ H_3=6771ce0ea70ba07e7c9b6c08690915c831faaee82dbe44672d198b388fae46af2bb3ba77985dcf5f0f7fc84f4227bd72ad5b5f1ee23ea560585eb7cee25c48fe Total computation time of U_3 is 38 μ seconds. .
User U_4	$H_4 = SHA - 512(T_4)$ H_4=a8b92164d9c3d5551ac2ee1cfee42792ef49de88ead77f99712ac9c93146993bca9b1d46b754afcca01191aa3aceb44573afc2889558bfa9ede7c3a93cc3f34f Total computation time of U_4 is 36 μ seconds.

In Table A.5, each user applies BLS Short Signature algorithm for hashed value H_i. Firstly they map this hashed value H_i to a point on the curve E in order to be applied in pairing function for BLS Short Signature (Mapping algorithm is available on PBC Library (WEB_1, (2013))). Mapping results are available in the Table A.5. Then, they sign this value with their static private keys a_i and broadcast to each others. These broadcast messages are 512-bit length (for each coordinate of the point). Approximate time for signing calculation of each user is 10,127 milliseconds. Specific time measures are available in the following table.

Table A.5. BLS Short Signature on Round-1.

	Round-1
	$Sig_{\mathrm{i}} = a_{\mathrm{i}} H_{\mathrm{i}}$
User U_1	$Sig_1 = a_1 H_1$ Hash H_1 is mapped to a point on E: [113942828904183175973951544717356067390734447913481465511131792620337661187971462585188555428776262431758344960403069355557861218350860497598337116127396 7, 321091434390007406762901172021149874279773411088789322798956399896447805783900799000595230346760635538706465529751530685267803762071696692400545910147777 4] Sig_1=[356522783924137889302968402380571315194696394202047185690054362621977712795134885810819608138952301731062670577072102021784314974966511459883515549055987 6, 475153495067989011297468575425586705540562785583951045337105703845562075256926661231981240725966548902348890367038353597172349503716407874639669642747459 5] Total computation time of U_1 is 10,140 milliseconds.
User U_2	$Sig_2 = a_2 H_2$ Hash H_2 is mapped to a point on E: [545962605928333672835804339726371532883411123150463613802317723698884469121770418726191560948684132265136911354823464916517277475864802281394472012569874 4, 312417757341775160670197279282689275676474427290360308217978576606972446159914142625228745184586046014756330085564535760841727158481550404648315099267140 8] Sig_1=[509141248607062459628115790224145541450455779296745861730744725248785503446507717376095585298236820596900990830121586730880337079943298029430988656470688 0, 383648283905794188664158363848549922594750183056967019749355777083320068398395031408332139322820397167269482274142740707312464496263558597470029697953704 7] Total computation time of U_2 is 9,901 milliseconds.

(cont. on next page)

Table A.5. (cont.)

User U_3	$$Sig_3 = a_3 H_3$$ Hash H_3 is mapped to a point on E: [3257896190711935822808692221717990350574265539767666149103974546145444187331178893682764758282813679046620444361217796544364320678053617949049518693548 20, 7031533192003869633354581525984171015501749558500824158722630067502470007641061953968490925472769629637513217219804649744634491336448401456762261843968 29] Sig_3=[3332394478602305975125998919548094551825109571655929444012725181843286173287468406886026888815393808640336545706119735265356449374268098632448746569009715, 1626984375273472587096168526570470851119104177746492233184101598455547492752413241719123531589497052359480130682067862869091055631544698504911338645661849] Total computation time of U_3 is 10,284 milliseconds. .
User U_4	$$Sig_4 = a_4 H_4$$ Hash H_4 is mapped to a point on E: [3166997709214605686871965772308348652378907933560394208727213304817939614612536965434220933906163659746493126147451206055849645679631626909909197770369315, 5324876429302458323049021066831728886533876331874474998260905852638415694451188449092543311845851585914039206317674982143865590870162585033587009405108169] Sig_4=[6474848874276409874680211746530607235573391097488360071089143062728303396226105959790530817678808012316082952544849349051919776952640277230104069456210712, 7487782310374750685878929007181506084417294508340029268504238300468856424267951201205147288448932235236437982717420544603240354138608619063559392999984693] Total computation time of U_4 is 10,181 milliseconds.

In Table A.6, each user applies two pairing functions; one inputs hashed value H_i and static public key Y_i; another inputs signature result Sig_i and generator P. If result of each function is equal, then Sig_i is verified. Approximate time for BLS signature algorithm including hashing algorithm for each user is 13,751 milliseconds. Specific time measures are available in the following table.

Table A.6. Verification of BLS Short Signature on Round-2.

	Round-2 $\mathrm{e}(Sig_{\mathrm{i}}, \mathrm{P}) = e(H_{\mathrm{i}}, Y_{\mathrm{i}})$
User U_1	$\mathrm{e}(Sig_2, \mathrm{P}) = e(H_2, Y_2)$ for U_2. Verified and result of both equations: [84989557624178270743422998102102081870366665395557022983327542365255294 43132140016507406971610627874015842774532809985773251424360590456854649 3 7023030969, 30026436250311244672153691449510789263627742572384814733433909014266519 3 85603942083255189332069008688876003828878980275042478503564863255747127 5 7895007707] Total time 4,334 milliseconds. $\mathrm{e}(Sig_3, \mathrm{P}) = e(H_3, Y_3)$ for U_3. Verified and result of both equations: [31055763863790624281041368059421966604990559436620673967759202824211637 46964258416389374421745161708425851386138015048224332833485403719589230 5 52880400680, 55890531236017845710447226793461866449934742223185459096828078100643375 1 03559185916417621645964159542673276628479494248979804859933886928428401 9 6711076000] Totall time 4,601 milliseconds. $\mathrm{e}(Sig_4, \mathrm{P}) = e(H_4, Y_4)$ for U_4. Verified and result of both equations: [42316553346631305965880408902543119862014675035526866872371681438863567 7 34301261607102877812796148486196517006003768892151396017884003429361419 8 2420130871, 13867920225865251074095687322639898968275350926056496412426857196151018 2 51676642228838323029482553268344406369271619928119033609185292893867208 6 4241451235] Totall time 4,622 milliseconds. Total computation time for verification of U_1 is 13,557 milliseconds without hashing algorithms. Total calculation time for verification of 13,668 milliseconds with hashing algorithms in the Table A.4.

(cont. on next page)

Table A.6. (cont.)

User U_2	e(Sig_1, P) = $e(H_1, Y_1)$ for U_1. Verified and result of both equations: [135091565516199143715944441564764352262869560856852905920195720530989972503922075282536785211202393121283331790367387965554354719871028407135145679141048 5, 730209471897317597724921534654894734917827297175983651702850233562375913627016875096536288782166323746376336371987770744943426751426320196776559388515 15] Totall time 4,630 milliseconds. e(Sig_3, P) = $e(H_3, Y_3)$ for U_3. Verified and result of both equations: [310557638637906242810413680594219666049905594366206739677592028242116374696425841638937442174516170842585138613801504822433283348540371958923055288040068 0, 558905312360178457104472267934618664499347422231854590968280781006433751035591859164176216459641595426732766284794942489798048599338869284284019671107600 0] Totall time 4,601 milliseconds. e(Sig_4, P) = $e(H_4, Y_4)$ for U_4. Verified and result of both equations: [423165533466313059658804089025431198620146750355268668723716814388635677343012616071028778127961484861965170060037688921513960178840034293614198242013087 1, 138679202258652510740956873226398989682753509260564964124268571961510182516766422288383230294825532683444063692716199281190336091852928938672086424145123 5] Totall time 4,622 milliseconds. Total computation time for verification of U_2 is 13,565 milliseconds without hashing algorithms. Total computation time for verification of 13,676 milliseconds with hashing algorithms in the Table A.4.

(cont. on next page)

Table A.6. (cont.)

User U_3	e(Sig_1, P) = $e(H_1, Y_1)$ for U_1. Verified and result of both equations: [135091565516199143715944441564764352262869560856852905920195720530989972503922075282536785211202393121283331790367387965554354719871028407135145679141048 5, 730209471897317597724921534654894734917827297175983651702850233562375913627016875096536288782166323746376336371987770744943426751426320196776559388515 15] Totall time 4,630 milliseconds. e(Sig_2, P) = $e(H_2, Y_2)$ for U_2. Verified and result of both equations: [849895576241782707434229981021020818703666653955570229833275423652552944313214001650740697161062787401584277453280998577325142436059045685464937023030969, 300264362503112446721536914495107892636277425723848147334339090142665193856039420832551893320690086888760038288789802750424785035648632557471275789500770 7] Totall time 4,334 milliseconds. e(Sig_4, P) = $e(H_4, Y_4)$ for U_4. Verified and result of both equations: [423165533466313059658804089025431198620146750355268668723716814388635677343012616071028778127961484861965170060037688921513960178840034293614198242013087 1, 138679202258652510740956873226398989682753509260564964124268571961510182516766422288383230294825532683444063692716199281190336091852928938672086424145123 5] Totall time 4,622 milliseconds. Total computation time for verification of U_3 is 13,586 milliseconds without hashing algorithms. Total computation time for verification of 13,696 milliseconds with hashing algorithms in the Table A.4.

(cont. on next page)

Table A.6. (cont.)

User U_4	e(Sig_1, P) = $e(H_1, Y_1)$ for U_1. Verified and result of both equations: [135091565516199143715944441564764352262869560856852905920195720530989972503922075282536785211202393121283331790367387965554354719871028407135145679141048 5, 730209471897317597724921534654894734917827297175983651702850233562375913627016875096536288782166323746376336371987770744943426751426320196776559388515 15] Totall time 4,630 milliseconds. e(Sig_2, P) = $e(H_2, Y_2)$ for U_2. Verified and result of both equations: [849895576241782707434229981021020818703666653955570229833275423652552944313214001650740697161062787401584277453280998577325142436059045685464937023030969, 300264362503112446721536914495107892636277425723848147334339090142665193856039420832551893320690086888760038288789802750424785035648632557471275789500 7707] Totall time 4,334 milliseconds. e(Sig_3, P) = $e(H_3, Y_3)$ for U_3. Verified and result of both equations: [310557638637906242810413680594219666049905594366206739677592028242116374696425841638937442174516170842585138613801504822433283348540371958923055288040068 0, 558905312360178457104472267934618664499347422231854590968280781006433751035591859164176216459641595426732766284794942489798048599338869284284019671107600 0] Totall time 4,601 milliseconds. Total computation time for verification of U_4 is 13,853 milliseconds without hashing algorithms. Total computation time for verification of 13,965 milliseconds with hashing algorithms in the Table A.4.

In Table A.7, each user performs $X_i = e((Y_{i+1} + T_{i+1}), (Y_{i+2}+T_{i+2}) - (Y_{i-1}+T_{i-1}))^{(a_i+a_i x_i)}$ calculation. X_i is 512-bit length. Approximate time for this calculation of each user is 2,619 milliseconds. Specific time measures are available in the following Table.

Table A.7. Public Message Computation on Round-2.

	Round-2 $X_i = e((Y_{i+1} + T_{i+1}), (Y_{i+2}+T_{i+2}) - (Y_{i-1}+T_{i-1}))^{(a_i+a_i x_i)}$
User U_1	$X_1 = e((Y_2 + T_2), (Y_3+T_3) - (Y_4+T_4))^{(a_1+a_1 x_1)}$ X_1= [681046290617751087216364147490552277777412229224285897333867753205299875341060584357539470867793845250183022366993126010578637257510245740742823395985936 5, 54374752077027273839659194698286630133188178518272762761973883714797858694327849810641337235604699747879696705052423094710794937671124341946137977304025 32]
	Total computation time of U_1 is 2,710 milliseconds.
User U_2	$X_2 = e((Y_3 + T_3), (Y_4+T_4) - (Y_1+T_1))^{(a_2+a_2 x_2)}$ X_2=[267941030253026562307168236973804209206248106594943930098865506442544938076157469998760183918342424087107696313791036522110926964082739071737056605126 0778, 47162070861985620121204450385213773982612474007659995730069946885043567949620970044580800971429002707368579416092679021743510879235175227356711436737586 91]
	Total computation time of U_2 is 2,537 milliseconds.
User U_3	$X_3 = e((Y_4 + T_4), (Y_1+T_1) - (Y_2+T_2))^{(a_3+a_3 x_3)}$ X_3=[569603088288681372164962384644912088082933993471221856301691710610341068628675417034470101072594838326063445809636401791044800801493456710631520532090 5869, 51188699933411743892551960232513340294817994441196942687917505337063487033919818454456192074302123999113167566165235220321800389259280782537732273256193 70]
	Total computation time of U_3 is 2,572 milliseconds.
User U_4	$X_4 = e((Y_1 + T_1), (Y_2+T_2) - (Y_3+T_3))^{(a_4+a_4 x_4)}$ X_4=[173004626696273916720127410490583318819197570659880735886722409008867901995820423246961249731320387416407661587267863601291219808804874840212259456731202, 86969186658535128258329181281755481440913789881658641934675626437869776360937508331919287204787715684055018954953430140683756307830260258986577087011615 50]
	Total computation time of U_4 is 2,658 milliseconds.

In Table A.8, each users applies SHA-512 hash algorithm from OpenSSL Library for X_i. It outputs 512-bit result. Approximate time for SHA-512 hash algorithm of each user is 20 μ seconds. Specific time measures are available in the following Table.

Table A.8. SHA-512 Algorithm on Round-2.

	Round-2 $H_i = SHA - 512(X_i)$
User U_1	$H_1 = SHA - 512(X_1)$ H_1=0100000000000000d350a39c707f000050a542d6ff7f000045c5df9c707f00000000d642be39e1fe0100000000000000080a842d6ff7f00000600000000000000000 Total computation time of U_1 is 21 μ seconds.
User U_2	$H_2 = SHA - 512(X_2)$ H_2=65d5ba07e70a6fa3ab4c9a6ee7c859b570ee80e2b89692bb6e866dc7173b6c7fa9d241ec8bd130bdadced67d152ea8860992486665065532f3f34d34daf995ff Total computation time of U_2 is 19 μ seconds.
User U_3	$H_3 = SHA - 512(X_3)$ H_3=8d15fc75c00f1dc0b7936adeafe21dd291836768c6cabaf3a4eec049b01c255ee910092d0bf5f324681c16ecef207e81e530302981190f27507b14f57707c840 Total computation time of U_3 is 21 μ seconds. .
User U_4	$H_4 = SHA - 512(X_4)$ H_4=e97c43c5ae5bc549a25e827a1a78d7cc114cdddacc56d83e9ad6feeea48370bb62aa50ecb326ccef45827ef0ce6d40605f9b23010d4f4d2eb0cc573accc0f6fd Total computation time of U_4 is 20 μ seconds.

In Table A.9, each user applies BLS Short Signature algorithm for hashed value. Firstly they map this hashed value to a point on the curve E in order to be applied in pairing function for BLS Short Signature. Mapping results are available in the Table A.9. Then, they sign this value with their static private keys a_i and broadcast to each

others. Each broadcasted message is 512-bit length. Approximate time for signing computation of each user is 9,978 milliseconds. Specific time measures are available in the following table.

Table A.9. BLS Short Signatures on Round-2.

	Round-2 $Sig_{\mathrm{i}} = a_{\mathrm{i}} H_{\mathrm{i}}$
User U_1	$Sig_1 = a_1 H_1$ Hash H_1 is mapped to a point on E: [584016226623664607714488434665961128967499116144090968256091713919620192318752207970493708527630199103105609213290221275823633863949879634566395536288 4877, 325622431542330543892981380973681277021498459179566052097515149546345896564827236162593644778803433570577625237532367563614951794511085197283339223161505] Sig_1=[746646782775406656461066359344885315398194607077743682737353559535592009127939415155486120795640055307613955139237255045105857399279090826874987251702128 6, 680348508250427691572957164453410315137849642634035934354444924753994124678451756539946545449928458644372680473271943688960962788164933310839524663528722 4]
User U_2	$Sig_2 = a_2 H_2$ Hash H_2 is mapped to a point on E: [757886524480780628830063021229773838401925365162872687199936459289929809891251692830625580860889372986591143706298498089551716762041455314338905855323734 8, 497863954323731309529072296663764087114516099791023711008572659069680690098026249676850946802885086074965802104547508788201822602208251998445903716226889 6] Sig_1=[481720175485644945866788387694910213707583436817877950988334893676902699146109745015732886057251387017738952575239311196637073768170955973947934411846809 3, 443171478554231022386062257980259481280363020582342470339085792010001399428013555367600373195348631450055008346654726864144446552555922827421539243809069 9]

(cont. on next page)

Table A.9. (cont.)

User U_3	$$Sig_3 = a_3 H_3$$ Hash H_3 is mapped to a point on E: [6367749321524882234217789427584544105804112281204311512039478729291413328524043342402774405667621695765958468064096385328964156621038017241143975891177695, 7886038920357207409303139739887329617481385409239892777479673544848301202734231425958357942016420354805449024242080434506013475729218907912539528131845686] Sig_3=[3376041775575516649128394044193924800732753440643205064951166118452996495366443399978801745856352472680622732938461526726642483000984018740940982220012450, 6499919351246050819254692101937681061507934371221308011585543355591202059442882544223613025998916867739277235190690708891842075506363087815733326180284636] Total computation time of U_3 is 9,738 milliseconds. .
User U_4	$$Sig_4 = a_4 H_4$$ Hash H_4 is mapped to a point on E: [5078205004449761654956639815676121966531588148545640644454573362146797848083683710396521468481391102520393388141469619056221088695209145959991054477441934, 2521744191801263891415104947448215613029249613964796495443794224412397957242845959598660882571272258875314019128125587413183065735163487801736172786741048] Sig_4=[6372835325479341339914319478667783801279944456215777050962692753365516195149503518035291114712768419988567793274470645733315623251815538750779757353099789, 7285860335942689502930962011165912527739291694620011802839262553775254534521735901214327210487118723670112373297080321911935827792975317200313284517735730] Total computation time of U_4 is 10,222 milliseconds.

In Table A.10, each user applies two pairing functions; one inputs hashed value H_i and static public key Y_i; another inputs signature result Sig_i and generator P. If result of each function is equal, then Sig_i is verified. Approximate time for BLS signature verification including hash algorithm of each user is 13,299 milliseconds. Specific time measures are available in the following table.

Table A.10. Verification of BLS Short Signature on Key Agreement.

	Key Agreement $\mathrm{e}(Sig_{\mathrm{i}}, \mathrm{P}) = e(H_{\mathrm{i}}, Y_{\mathrm{i}})$
User $\boldsymbol{U_1}$	$\mathrm{e}(Sig_2, \mathrm{P}) = e(H_2, Y_2)$ for U_2. Verified and result of both equations: [308430275061879083679951492534601356138749672572632982191140948700551226957051024386188690791488822915942338106449707207037154296844208814359767826692814 6, 535690461683446292343990689624473191430546046804721314286695932497295027732683695115653369221838813058282810468005097855369063155618714762871869243348539 1] Totall time 4,532 milliseconds. $\mathrm{e}(Sig_3, \mathrm{P}) = e(H_3, Y_3)$ for U_3. Verified and result of both equations: [421103814267229577703200975496069321294589292021542621890618037090552292320936937540693019233561612457091340185151463880261015381863584260458337636599298 6, 335854001297228876012391959958609373586118786906276639005490510825202604907463426742376777510834862142578128360270193138368814872794322891777008807728608 1] Totall time 4,377 milliseconds. $\mathrm{e}(Sig_4, \mathrm{P}) = e(H_4, Y_4)$ for U_4. Verified and result of both equations: [560834408582962372774918053889655159383253726434428138982352461921211341274965936143327125209845882886328876850508784753522913251500703100933723910903158 2, 382670355318454814883139469919849098096917706913139170029291120227019829733817215564216380573066881535899738918702577397418266294764145184150748445952151 3] Totall time 4,402 milliseconds. Total computation time for verification of U_1 is 13,311 milliseconds without hash algorithms. Total computation time for verification of 13,371 milliseconds with hash algorithms in the Table A.8.

(cont. on next page)

Table A.10. (cont.)

User U_2	e(Sig_1, P) = $e(H_1, Y_1)$ for U_1. Verified and result of both equations: [770210323516612844466933389763339100798178709608339391689766366479000171305173221078353734333074793461471162195011837114062330438554957394130616264727871 0, 453510745067020540842607150738945320690618751273976147626672200940147285830595445685448154801791624856620253313567973077078664906517383430321857694883269 1] Totall time 4,340 milliseconds. e(Sig_3, P) = $e(H_3, Y_3)$ for U_3. Verified and result of both equations: [421103814267229577703200975496069321294589292021542621890618037090552292320936937540693019233561612457091340185151463880261015381863584260458337636599298 6, 335854001297228876012391959958609373586118786906276639005490510825202604907463426742376777510834862142578128360270193138368814872794322891777008807728608 1] Totall time 4,377 milliseconds. e(Sig_4, P) = $e(H_4, Y_4)$ for U_4. Verified and result of both equations: [560834408582962372774918053889655159383253726434428138982352461921211341274965936143327125209845882886328876850508784753522913251500703100933723910903158 2, 382670355318454814883139469919849098096917706913139170029291120227019829733817215564216380573066881535899738918702577397418266294764145184150748445952151 3] Totall time 4,402 milliseconds. Total calculation time for verification of U_2 is 13,119 milliseconds without hash algorithms. Total calculation time for verification of 13,181 milliseconds with hash algorithms in the Table A.8.

(cont. on next page)

Table A.10. (cont.)

User U_3	e(Sig_1, P) = $e(H_1, Y_1)$ for U_1. Verified and result of both equations: [770210323516612844466933389763339100798178709608339391689766366479000171305173221078353734333074793461471162195011837114062330438554957394130616264727871 0, 453510745067020540842607150738945320690618751273976147626672200940147285830595445685448154801791624856620253313567973077078664906517383430321857694883269 1] Totall time 4,340 milliseconds. e(Sig_2, P) = $e(H_2, Y_2)$ for U_2. Verified and result of both equations: [308430275061879083679951492534601356138749672572632982191140948700551226957051024386188690791488822915942338106449707207037154296844208814359767826692814 6, 535690461683446292343990689624473191430546046804721314286695932497295027732683695115653369221838813058282810468005097855369063155618714762871869243348539 1] Totall time 4,532 milliseconds. e(Sig_4, P) = $e(H_4, Y_4)$ for U_4. Verified and result of both equations: [560834408582962372774918053889655159383253726434428138982352461921211341274965936143327125209845882886328876850508784753522913251500703100933723910903158 2, 382670355318454814883139469919849098096917706913139170029291120227019829733817215564216380573066881535899738918702577397418266294764145184150748445952151 3] Totall time 4,402 milliseconds. Total computation time for verification of U_3 is 13,274 milliseconds without hash algorithms. Total computation time for verification of 13,334 milliseconds with hash algorithms in the Table A.8.

(cont. on next page)

Table A.10. (cont.)

User U_4	e(Sig_1, P) = $e(H_1, Y_1)$ for U_1. Verified and result of both equations: [770210323516612844466933389763339100798178709608339391689766366479000171305173221078353734333074793461471162195011837114062330438554957394130616262647278710, 453510745067020540842607150738945320690618751273976147626672200940147285830595445685448154801791624856620253313567973077078664906517383430321857694883269 1] Totall time 4,340 milliseconds. e(Sig_2, P) = $e(H_2, Y_2)$ for U_2. Verified and result of both equations: [308430275061879083679951492534601356138749672572632982191140948700551226957051024386188690791488822915942338106449707207037154296844208814359767826692814 6, 535690461683446292343990689624473191430546046804721314286695932497295027732683695115653369221838813058282810468005097855369063155618714762871869243348539 1] Totall time 4,532 milliseconds. e(Sig_3, P) = $e(H_3, Y_3)$ for U_3. Verified and result of both equations: [421103814267229577703200975496069321294589292021542621890618037090552292320936937540693019233561612457091340185151463880261015381863584260458337636599298 6, 335854001297228876012391959958609373586118786906276639005490510825202604907463426742376777510834862142578128360270193138368814872794322891777008807728608 1] Totall time 4,377 milliseconds. Total computation time for verification of U_4 is 13,249 milliseconds without hash algorithms. Total computation time for verification of 13,310 milliseconds with hash algorithms in the Table A.8.

In the Table A.11, each user performs $K_i = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i + a_i x_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \ldots X_{i-2}$ calculation for common key agreement. Approximate time for this calculation of each user is 2,593 milliseconds. Specific time measures are available in the following table.

Table A.11. Common Key Agreement.

	Key Agreement
	$K_i = e\big((Y_{i+1} + T_{i+1}), n(Y_{i-1} + T_{i-1})\big)^{a_i + a_i x_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \dots X_{i-2} = e(P,P)^{\left[\begin{smallmatrix}(a_n+a_n x_n)(a_1+a_1 x_1)(a_2+a_2 x_2)+(a_1+a_1 x_1)(a_2+a_2 x_2)(a_3+a_3 x_3)+\cdots \\ +(a_{n-1}+a_{n-1} x_{n-1})(a_n+a_n x_n)(a_{n+1}+a_{n+1} x_{n+1})\end{smallmatrix}\right]}$.
User U_1	$K_1 = e\big((Y_2 + T_2), 4(Y_4 + T_4)\big)^{a_1 + a_1 x_1} \cdot X_1^3 \cdot X_2^2 \cdot X_3$ K_1=[5125574494847115316399676514623981300216628367118981355057769382952745425750796821255868638660969378754082693425616582977490919056820246043662093936375526, 2511388902002571832779373221836357824917351535395495312046311145855436475526025751560834740713029226997340019228876400746565907197666070785972511781262733] Total computation time of U_1 for *key agreement* is 2,657 milliseconds.
User U_2	$K_2 = e\big((Y_3 + T_3), 4(Y_1 + T_1)\big)^{a_2 + a_2 x_2} \cdot X_2^3 \cdot X_3^2 \cdot X_4$ K_2=[5125574494847115316399676514623981300216628367118981355057769382952745425750796821255868638660969378754082693425616582977490919056820246043662093936375526, 2511388902002571832779373221836357824917351535395495312046311145855436475526025751560834740713029226997340019228876400746565907197666070785972511781262733] Total computation time U_2 for *key agreement* is 2,672 milliseconds.
User U_3	$K_3 = e\big((Y_4 + T_4), 4(Y_2 + T_2)\big)^{a_3 + a_3 x_3} \cdot X_3^3 \cdot X_4^2 \cdot X_1$ K_3=[5125574494847115316399676514623981300216628367118981355057769382952745425750796821255868638660969378754082693425616582977490919056820246043662093936375526, 2511388902002571832779373221836357824917351535395495312046311145855436475526025751560834740713029226997340019228876400746565907197666070785972511781262733] Total computation time U_3 for *key agreement* is 2,528 milliseconds.
User U_4	$K_4 = e\big((Y_1 + T_1), 4(Y_3 + T_3)\big)^{a_4 + a_4 x_4} \cdot X_4^3 \cdot X_1^2 \cdot X_2$ K_4=[5125574494847115316399676514623981300216628367118981355057769382952745425750796821255868638660969378754082693425616582977490919056820246043662093936375526, 2511388902002571832779373221836357824917351535395495312046311145855436475526025751560834740713029226997340019228876400746565907197666070785972511781262733] Total computation time of U_4 for *key agreement* is 2,516 milliseconds.

Table A.12 presents the summary of the original protocol (Lin et al., (2006)) for four users. It approximately needs 17,1 milliseconds computation time for per user.

Table A.12. Summary of Results of the Original Protocol.

	User U_1 (millisecond)	***User U_2 (millisecond)***	***User U_3 (millisecond)***	***User U_4 (millisecond)***
Round-1 (T_i Computation)	3,592	3,559	3,753	3,493
Round-2 (X_i Computation)	2,644	2,471	2,505	2, 592
Key Agreement (Key Generation)	2,657	2,672	2,528	2,516

Table A.13 presents the summary of the proposed protocol for four users. Total computation time is approximately 56 milliseconds for per user.

Table A.13. Summary of Results of the Proposed Protocol.

		User U_1 (millisecond)	***User U_2 (millisecond)***	***User U_3 (millisecond)***	***User U_4 (millisecond)***
Round-1	***T_i Calculation***	3,592	3,559	3,753	3,493
	SHA-512 Algorithm	0,037	0,037	0,038	0,036
	BLS Short Signature	10,140	9,901	10,284	10,181
	Total for Round-1	**13, 769**	**13, 497**	**14, 075**	**13, 710**
Round-2	***Signature verification with hash algorithm***	13,668	13,676	13,696	13,965
	X_i Calculation	2,644	2,471	2,505	2, 592
	SHA-512 Algorithm	0,021	0,019	0,021	0,020
	BLS Short Signature	9,887	10,063	9,738	10,222
	Total for Round-2	**26,199**	**26,229**	**26,960**	**26,799**
Key Agreement	***Signature verification with hash algorithm***	13,311	13,119	13,274	13,249
	Key Generation	2,657	2,672	2,528	2,516
	Total for KeyGen	**15,968**	**15,791**	**15,802**	**15,765**

APPENDIX B

IMPLEMENTATION OF GROUP KEY DISTRIBUTION PROTOCOL

All implementation details and the every values of each variable in the related steps of Section 4.3.4 can be found and followed under Appendix B.

Table B.1 includes the input parameters which are 1024-bit prime p with primitive root g of Z_p^* and 512-bit prime q where $q \mid (p-1)$. This test data is acquired from (Allen (2008)).

Table B.1. Input Parameters.

Input Parameters
There are Leader L and 4 users, namely; U_1, U_2, U_3 and U_4. p=11838184372471710174946159675664648223090589766046312362456039456380760993395604226539234152095602888644631771664207057053879231168634640942410140411181283316085659935320027832070906986302148069534969208735860164025083645711880093251235268088221149165473251353285154670278619087767951265337570934555271330240 1 g=59660846376012012299732062167070449913568079369759726390943366472735655747125035179031089451125540857538031073871730574393537580324435989370818322776711385170287961641652843108956199416275969391836776116950834964228187667553031050881716218984733944262206823833143460937854518070649325296332195764146328701846 q=10643279190065436658189986618064406421644965048931123759059399961267188560280838103148616561846017372648276481588281249312389181981519220200679285520165533

Each user generates his/her static public private key pair as seen in the Table B.2 for setup parameters. Their bit lengths are 1024-bit. Then he/she applies a legitimate Certification Authority (CA) such as VeriSign (WEB_4, (2013)) and exchanges his/her certificate with other participants. It is offline task of the protocol and public-private key pairs in the Table B.2 are calculated by using GMP Library.

Table B.2. Setup Parameters.

Setup Parameters	
Leader L	$Y_{\mathrm{L}} = g^{x_{\mathrm{L}}} \ (mod\ p)$ x_{L}=624721139944302845120158894651212669095900885513726009996762322717651078479068508194474483103779297569505765109175065706985743526704094031728640592829796875876141636920161074022615639704350378568028603802872871474649244938316848302108067439455680528809370088197272651933576166120125518618708568134140026145 9 Y_{L}=464332380989113754890128925023491582195978133970797553669546316092509883301014707965752096541010640620604267360451295859376474188192653097119611459216424008800604363380379691872021706480483394524481755422748544534634894408199252392242101194723541893752051293940365348643519043234829548574785217484736765013 66
User U_1	$Y_1 = g^{x_1}(mod\ p)$ ID_1=1000 x_1=939274252929733329245706686610136571633853179494326867573700787748431285678647901603705763527822307692929503263880532988528159166649674990851639312951401311462924709956732184873710080287985136525536785462470302311746871890895223456503805215456479662461445571437278397466833314835644593120157310014667310 1435 Y_1=112164028721832983324553677381342841886294536651878840592942883942405027110089791630377228963799418727728227818748733909739160812906443902605520617373524522684653317339802698700280150116828740628329540327062963574962424598753509925163574566653239016123019756169509839736998025892778255970477163440305373675 64
User U_2	$Y_2 = g^{x_2}(mod\ p)$ ID_2=2000 x_2=456282530886490211251777670500544716546607383842395646271801632397848587008323012492153146735848251666889420333457194479543000250307699738611994891595198431257674908982283571132183203847644078809570004049044623114285272514313978118079621979860224598583162257365627770394658619706596092265611294515609574222 12 Y_2=741962194969843363798367215186349285636218111566693303258407655922147391575981821136762082207427958073528084931675182819066612883203281237279709068679503104782776627279314546847867519110354596511116265082076162183286356694683953935898043768950694346836443337056956703115384437276600343173857046910310402299 59

(cont on next page)

Table B.2. (cont.)

User U_3	$Y_3 = g^{x_3}(mod\ p)$ ID_3=3000 x_3=98434463799787067345355698394395217219199513588809040359782269482337483100937892023725884269466092527620101194261949829729721813388306594528096014260477867941356411841910346262657708003347666631840373270459493988671838908150983348800712696156741688374155443077747350025543102353343076305769235288505497163892 Y_3=42315155905008745181367957168084316745007881775344513724828828628938632920091049351133391472910372694903360377627590403368479645833973580494446819208033572667276756879226309096834949523821746097114817796492707919345938834520924961375415790637736768559851446130124087312838221618009293806779304743242940741287
User U_4	$Y_4 = g^{x_4}(mod\ p)$ ID_4=4000 x_4=54516940191644350109190136615681809157143220562421881357042424455097266818410758097589746068573698599873648684004581084117520861958079081982518566821440677725094602575229219732682356627903505665281097151784293320152912717397012691890097506723712797485397614104229912857572831819815369001738492078631022953980 Y_4=72149955597230169390850947027165016278008580867384810043204113717714369082102034064553719330184666864292647903859296906478835328897852942903221717356085044062566372461029051111808177804884029559887972950842768310921830064951821454082729220673745570385447578163876035852021632918675694699344085415138211139835

In Table B.3, Leader generates 2-degree polynomial $f(x)$ with 512-bit constants (degree size is depending on user number, see Section 4.3.2) and picks the common secret which is $f(0)$. Polynomial generation takes 21 μ seconds. Then, Leader calculates commitment vector which takes is 5,483 milliseconds and each element in the vector is 1024-bit length. Finally, Leader computes subsecrets which will be distributed to the four users. Subsecret computation takes 16 μ seconds for four users.

Table B.3. Polynomial Generation, Commitment and Subsecret Computation.

Leader L	***Round-1***
	$f(x) = a_0 + a_1 x + a_2 x^2 \, mod \, q$ a_0 = (SECRET KEY) 838355556594409371859376473939134800750013303688371648654174765418316578029863928010946716563772272950582199151914080658849097286834249098372357246295352 0 a_1=2428293149901509306139132918171431769109835394353665109028442819095155450746087464752542029173775648669017036973153478972376423606449458938829218494575260 a_2=1306496017242772172743264447276527112501750362327928440722559572994619766769639623457450192156124916040769111470258706639743561430669385023580620797552756 Total computation time for *polynomial generation* is 21 μ seconds.
	$C_j = (C_0 = g^{a_0}, C_1 = g^{a_1}, C_2 = g^{a_2}) \, (mod \, p)$ C_0=3143683525979079438765391635881157522909064933040545518349199971361588261895827382439388405853069156030373690708329549287178887502102736142773874942941849717816945595565030875769232896795907421462390080692170753943989097652273935499342168915103391261202132300508823916443821001876727030357144873506277992222 5 C_1=1107047503677799751679087477009387825694569797888756011850250854928233191824128484441235793720633978366933163028163051526247085287825474284940932748895483820177596722999575479485417801645429845436923898364193775517704975993964899408315999138887765467994741058626762071503649519360903720666069269812273033050 90 C_2=1048824605645551624071155651959993263958273610139954748153470921715143588279720502418170538696598535735464335265763817970693645020365903568374305355953208503532302672994737511961379943918161785083210076602629199622532654542070965443086453535551042806288384375020322751294011550809347075977234157259339364359 22 Total computation time for *commitments* is 5,483 milliseconds
	$s_i = f(ID_i) \, (mod \, q)$ s_1=9325956120950456012396966411050721266062076715274114314315248077218657079779600596816435174208891372960705333746565862799667975060862904938477787796634114 s_2=7258564187564715080115513694945253799292600327514083617479800299666613167745682238529116112541065068191647261655339414988181588037721883477575462403135543 s_3=6074902912222228838841242664783080182269380906981485526567774207483031039040129610354651827306215776895999213184626794088356980500210829533548052242126741 s_4=8024890976133023535606870168682957697181514458860740917754846492438307882142140278306867223562358636057402869550799972892882156875072348862444833833773241 Total computation time for *subsecrets* is 16 μ seconds.

In Table B.4, Leader uses ElGamal signature algorithm in order to sign the commitment vector. Elements in commitment vector are concatenated and input SHA-512 hash algorithm from OpenSSL Library. Hash calculation takes 58 μ seconds and computation time for ElGamal signature algorithm on the hashed value is 1,548 milliseconds. It outputs 1024-bit for each message pair such that (sig_1, sig_2).

Table B.4. SHA-512 Algorithm and ElGamal Signature.

	Round-1
Leader L	$H_{Commit} = SHA - 512(C_0 \| C_1 \| C_2)$ $sig_1 = g^r \ (mod\ p)\ \ where$ random r is gcd$(r, p-1) = 1$ $sig_2 = (H_{Commit} - x_L s_1) r^{-1} \ \ mod\ (p-1)$ H_{Commit}=b5b9320edd95efaca0a762eb640d6c21b2b614b923caeafb7a7ffd099f58c40812f69bd734522672a8e3905e50e23786e298850ced6d243a153a5d336e7e74fc Total computation time for SHA-512 is 58 μ seconds. sig_1=69557538949702495473089263059933479850837144965700900793009680758477984928813981894211106849656469541643011472662177863103146574331283065723831310655817698460145018844658114177716288523215160736065814295728004113263899009383350149630323563209674254006648928977357491061807224413843307732639800548754281182658 sig_2=79874166570045410084344745667789734558577349799358070963461933394526843737732201377707665754278209097394489030047055274586596732183724052073820715635830701373221170434619101500475861275107865965025053115765422128612898691368461335408583294353925036265232204518086859225728163168117502525855451867944431835 10 Total computation time for signing hashed value is 1,548 milliseconds.

In Table B.5, Leader encrypts calculated subsecrets using ElGamal encryption. It outputs 1024-bit encrypted messages. Total calculation time for all users is 10,391 milliseconds. Specific time measures are available in the following table.

Table B.5. ElGamal Encryption of Subsecrets.

	Round-1
Leader L	Encryption for U_1: $$ca_1 = g^{r_1}\ (mod\ p) where\ random\ r_1\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$ca_2 = s_1(Y_1)^{r_1}\ mod\ p$$ r_1=458036043066402672239738090844830599284531710784926641299938476705875786720609664810367247761797902559740727665604505052603182525694437489768962236940367880390010983684153689152095440315709189192041047392702156639940130822815089911392621015355986224941055870101254476859949095006527661914888562821501211186 06 ca_1=557633755030562490500103654543188854400824841173042905737338571441392325415040079283386125335206560928116395898138320205774898004166123640866146492446954761923048560250253228506809175423694919873126937076219517249033050062567348809982188501230094522805378766601940888395112901150958569950759066198814733060 ca_2=611751051594167668610237021674746033062017510849856011304068114205595841207607579042167182008118334237388043397005474315786742117225200272334462165333848402913754388366085356022287645885964918147531313010433909524620321898551928787202967133874822156474729820155130077633964122431923902289343038758585385417857 Total encryption time for U_1 is 2,558 milliseconds.
	Encryption for U_2: $$cb_1 = g^{r_2}\ (mod\ p) where\ random\ r_2\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cb_2 = s_2(Y_2)^{r_2}\ mod\ p$$ r_2=435229279011303188502737890174534813027993774751761837755736948271260615889540028358130998461009840221007364848964278967763901408764042509885972045181385020964981016142933267905371142225753471059576545584040175710592327556330276119392456695530775500153702603359178671399676873600454899765412619922235988033 55 cb_1=618699587376416226796132960700883376872407584115930037770851113950146097978626753610275777331199757895737208584455650867071451425034286114379809754418846511395255232409629981072743941678409493457358990586043705961419596057126295990015196379641713618911240410466315969581126325975535452598207402114818492819 58 cb_2=814882576568089089197088545554518400510925187005436018304901223675665539663912772257320485912686214891891905772014126337414624004477921168903518191634202736700490770931705600439384636378131478295653498445352495165026500984091531376175766460563974330950977607011838654067168255002670046014188469416569281105 0 Total encryption time for U_2 is 2,533 milliseconds.

(cont. on next page)

Table B.5. (cont.)

Leader L	Encryption for U_3: $$cc_1 = g^{r_3}\ (mod\ p) where\ random\ r_3\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cc_2 = s_3(Y_3)^{r_3}\ mod\ p$$ r_3=10155066550053087382438587765857818665637176312909077315409033353480224045954148525799371750531775279647792703126348563296923385592731868697991564079931377578228123572092410168620673336950410544233915009791130637341592830526855961205617212636167713170610606897284917624711230168501719716089922580747447565542 cc_1=48371371113772096519312350051469503290477887696323346132004870289394776987861502149848626853254151295047804856186940539215996698021408115743219800537901979761774853371481741861321356085367284619197357340713540392794262856418702320159669972438117858560113652316998973426746528830280019205580204739892072400955 cc_2=10991384788337848451451517629709944799259802538120593692383670703110226715531019053080866045338512932888369076882932740542408184846910496023138929817771729543781791383508576756369997886518579756113924461957242557829144758100726525360664722388979462199326422935937325825495314599912926429180525365433390384678 3 Total encryption time for U_3 is 2,678 milliseconds.
	Encryption for U_4: $$cd_1 = g^{r_4}\ (mod\ p) where\ random\ r_4\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cd_2 = s_4(Y_4)^{r_4}\ mod\ p$$ r_4=11216402872183298332455367738134284188629453665187884059294288394240502711008979163037722896379941872772822781874873390973916081290644390260552061737352452268465331733980269870028015011682874062832954032706296357496242459875350992516357456665323901612301975616950983973699802589277825597047716344030537367564 cd_1=69443031593589862076514140665409240155642557627930088663540809215240693363381150390339739168827108446933125630896368538880374139597280531635762085713491209380303604202714386872135007319139406746842022679420959187999987558358554408693047746285236498154722646992943435539507406010774996770626376901161715946483 cd_2=40564553005618513584764500295803124437089863691988608419374465711225223918272535491547692391242045133930127760746774184534582837991211384888094958405915155887553375602359598603518723646626695093865926572274190438958568109649799107897308539210347546350686709766732060469595398193712320564098056193208472673709 Total encryption time for U_4 is 2,622 milliseconds. Total encryption time for all users performed by Leader is 10,391 milliseconds.

In Table B.6, each user verifies signature on commitment vector. Total verification time using ElGamal verification algorithm of per user is 3,402 with hashing algorithm.

Table B.6. ElGamal Signature Verification.

	Round-2
	Each $U_i$verifies Leaders Signature using Leader's public key. $$g^{H_{commit}} = (Y_L)^{sig_1} {sig_1}^{sig_2} \ mod\ p$$ $g^{H_{commit}}$=11228589940609839711793437717613729369537046663346496578583321884086381704861992398389583736044254603682415189731671039209018318814347332167368919370922095320858615291422191084449636584941726534002771195148088837602757520636255184003414068414847133054530490444267938832976805684552631201570794738559555174664 9 $(Y_L)^{sig_1} {sig_1}^{sig_2}$=11228589940609839711793437717613729369537046663346496578583321884086381704861992398389583736044254603682415189731671039209018318814347332167368919370922095320858615291422191084449636584941726534002771195148088837602757520636255184003414068414847133054530490444267938832976805684552631201570794738559555174664 9 Total verification time of per user is 3,344 without hashing algorithm. Total verification time of per user is 3,402 with hashing algorithm.

In Table B.7 each user decrypts received encrypted message to output his/her subsecret. Approximate decryption time for each user is 1,337 milliseconds. Specific computation time is available in the following table.

Table B.7. ElGamal Decryption.

	Round-2
User U_1	$$s_1 = ca_2 \cdot ((ca_1)^{x_1})^{-1} \; mod \; p$$ s_1=93259561209504560123969664110507212660620767152741143143152480772186570797796005968164351742088913729607053337465658627996679750608629049384778796634114 Total decryption time is 1,283 milliseconds.
User U_2	$$s_2 = cb_2 \cdot ((cb_1)^{x_2})^{-1} \; mod \; p$$ s_2=72585641875647150801155136949452537992926003275140836174798002996666131677456822385291161125410650681916472616553394149881815880377218834775757462403135543 Total decryption time is 1,365 milliseconds.
User U_3	$$s_3 = cc_2 \cdot ((cc_1)^{x_3})^{-1} \; mod \; p$$ s_3=60749029122222288388412426647830801822693809069814855265677742074830310390401296103546518273062157768959992131846267940883569805002108295335480522421267 41 Total decryption time is 1,360 milliseconds.
User U_4	$$s_4 = cd_2 \cdot ((cd_1)^{x_4})^{-1} \; mod \; p$$ s_4=80248909761330235356068701686829576971815144588607409177548464924383078821421402783068672235623586360574028695507999728928821568750723488624448338337732 41 Total decryption time is 1,360 milliseconds.

In Table B.8, each user verifies correctness of his/her subsecret using commitment vector. Approximate verification time for each user is 748 μ seconds. Specific computation time is available in the following table.

Table B.8. Verifying Correctness of Subsecrets by Using Commitment Vector.

	Round-2
User U_1	U_1 verifies his subsecret received from Leader and sends to other users. $g^{s_1}(mod\ p) =$ 814982349632022237078488014254493365207617001769928008613745139106963 170712303831723722630394076758145179589226775506361229182413035264313 912420677429028816772170469040150494808406630446072310567843091628364 367381326434708520739200477409709449713648480151009619208767535524171 39503228595018672595604890699877 $\prod_{j=0}^{2} C_j^{1000^j} = (g^{a_0})^{1000^0} \cdot (g^{a_1})^{1000^1} \cdot (g^{a_2})^{1000^2} (mod\ p)=$ 814982349632022237078488014254493365207617001769928008613745139106963 170712303831723722630394076758145179589226775506361229182413035264313 912420677429028816772170469040150494808406630446072310567843091628364 367381326434708520739200477409709449713648480151009619208767535524171 39503228595018672595604890699877 Total time of U_1 for these estimations is 774 μ seconds.
User U_2	U_2 verifies his subsecret received from Leader and sends to other users. $g^{s_2}(mod\ p) =$ 252114500680700631019890837607515580297843054754568213763763094298530 803909656904541264899548194611939886693036763442996187924885356805264 582349182738556749330052150695291921177717274387558083793380675989079 763302215986555924536539726648510215993229826809798394215720557780505 694170922891959208446661442732 60 $\prod_{j=0}^{2} C_j^{2000^j} = (g^{a_0})^{2000^0} \cdot (g^{a_1})^{2000^1} \cdot (g^{a_2})^{2000^2} (mod\ p)=$ 252114500680700631019890837607515580297843054754568213763763094298530 803909656904541264899548194611939886693036763442996187924885356805264 582349182738556749330052150695291921177717274387558083793380675989079 763302215986555924536539726648510215993229826809798394215720557780505 694170922891959208446661442732 60 Total time of U_2 for these estimations is 707 μ seconds.

(cont. on next page)

Table B.8. (cont.)

User U_3	U_3 verifies his subsecret received from Leader and sends to other users. $g^{s_3}(mod\ p) =$ 98725135771536217184597941419132737168641497641833707802360561427396090272427203022172866999877257175666054824555705546276610795964769080046273082478773825396255027276970316920459107799130444425232047788587120005923915321707100220386528991993312657164025922390249634113429843041828691934686073037555254183005 $\prod_{j=0}^{2} C_j^{3000^j} = (g^{a_0})^{3000^0} \cdot (g^{a_1})^{3000^1} \cdot (g^{a_2})^{3000^2}(mod\ p)=$ 98725135771536217184597941419132737168641497641833707802360561427396090272427203022172866999877257175666054824555705546276610795964769080046273082478773825396255027276970316920459107799130444425232047788587120005923915321707100220386528991993312657164025922390249634113429843041828691934686073037555254183005 Total time of U_3 for these estimations is 719 μ seconds.
User U_4	U_4 verifies his subsecret received from Leader and sends to other users. $g^{s_4}(mod\ p) =$ 50137952632898556178583544928682702002542107291305672376321953236159519025114192226797232361415205360367027031122575271407411253427656407838116670876106138423657940075261853524016667328294676358987096284306880097536741770463300000969164571771562085161536013837735369058135759342924311237487593337447184971242 $\prod_{j=0}^{2} C_j^{4000^j} = (g^{a_0})^{4000^0} \cdot (g^{a_1})^{4000^1} \cdot (g^{a_2})^{4000^2}(mod\ p)=$ 50137952632898556178583544928682702002542107291305672376321953236159519025114192226797232361415205360367027031122575271407411253427656407838116670876106138423657940075261853524016667328294676358987096284306880097536741770463300000969164571771562085161536013837735369058135759342924311237487593337447184971242 Total time of U_4 for these estimations is 794 μ seconds.

In Table B.9, U_1 encrypts his subsecret using ElGamal encryption and related user's public key who will receive the encrypted message. Encrypted messages are 1024-bit length. Total encryption time performed by U_1 for three users who will receive the message is approximately 7,596 milliseconds.

Table B.9. U_1's Encryption.

	Round-2
User U_1	Encryption for U_2: $$cab_1 = g^{ra_2}\ (mod\ p) where\ random\ ra_2\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cab_2 = s_1(Y_2)^{ra_2}\ mod\ p$$ ra_2=34377253757476211649950124505029163912086176029324304540532623103435 78971517738970580003526703019910121304832502451940376003698476062366717036 36002019292713471183012566212039970457982938532813057732438993774335518955 55494094390735655647623456681555619147775274050843924814841428279589718429 8979388811682474470226 0 cab_1=10415989421987043702506742030624883126052677065584731481841718481725 42357892973811878267043169155729952542213137587933037190438444139255040 52232900072208382644646153361094092363049079697091072180747642017076873631 31317024372246517809150684745640301392075902294156394338990074161000639414 4493955266620908138241323 cab_2=24261714556153162463570809961222786711846277747628866842093792226111 37407132414546700187376534267407822897996244826089578032864894141912444000 04218713044479166689921543624498278877291500220430400246398173904316248 57888001171639035532711812748841572291140222348837310247095825396558649022 97168556313815179771 02 Total encryption time for U_2 is 2,580 milliseconds. Encryption for U_3: $$cac_1 = g^{ra_3}\ (mod\ p) where\ random\ ra_3\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cac_2 = s_1(Y_3)^{ra_3}\ mod\ p$$ ra_3=93104220810032701203108881730113021262180030914130426040344459035008 76540191330926575114261898577266947289226912214712390025246389423045373091 28495381177457264853808130216098093893592564805972476550905126501424054507 66317814137352702119708494381250237444016282001114500038595433108440797 31618692110449992 4960 cac_1=96513432955318175192959735690468173310609445941490526161868890137058 57024063097868399122737301644882379259686062933188225417467952166677635825 61478998510547413541837576742896187543547901159112335543401192411386863 61504224146052620461854199820484990070000864813611733928091176804824700 27884511152193572373 4539 cac_2=91108618140916332005401611200983995791897551156060412237024210342875 22177858183622164882829191013316483432932288415719954605790572692116552 83254973416198554219612953103594212510325672373871071023647204932520513063 62502472882024637215350020256996868301849903733225286209004307831032714 71610416964410883118 5436 Total encryption time for U_3 is 2,503 milliseconds.

(cont. on next page)

Table A.9.(cont.)

User U_1	Encryption for U_4: $$cad_1 = g^{ra_4}\ (mod\ p) where\ random\ ra_4\ is\ \{1, 2, .., \mathrm{p}-1\}$$ $$cad_2 = s_1(Y_4)^{ra_4}\ mod\ p$$ ra_4=52767667087391306488593155300780141793144427998561459609982474273877165852426447017865427255851990315529870926377048510452504251866894776282557441378818136587576472012352294753870584773611721806583110688747281337609166164120300147276175361844631694217057330444190107578966873133289288641622958482163322663233 cad_1=63818981056612164489118375440026271667478030512492924656091745622249292321705089950902753537058902042891373499710120428517023566104505143913136439962115503593574075063381990530362539702117454952150803195958726714657955201815150089789830737150248230450367440016985184827896776520627139524977512536314016447065 cad_2=10206620256281375745211031784086835374715267267025069846247696683938215072088532500108505928034589149186549971850393459212379692233896789805737229898919523676158618889273900057759171892149575795038576558483434449276321626267053521616754631059044639302865037150248619605356724432894131618123305344677483720860003 Total encryption time for U_4 is 2,513 milliseconds. Total encryption time performed by user U_1 for all users is 7,596 milliseconds.

In Table B.10, U_2 encrypts his subsecret using ElGamal encryption and related user's public key who will receive the encrypted message. Encrypted messages are 1024-bit length. Total encryption time performed by U_2 for three users who will receive the message is approximately 7,434 milliseconds.

Table B.10. U_2's Encryption.

User U_2	Encryption for U_1: $$cba_1 = g^{rb_1}\ (mod\ p) where\ random\ rb_1\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cba_2 = s_2(Y_1)^{rb_1}\ mod\ p$$ rb_1=113874249723801184526149961064304556997818570851722823429645579541943831849738834427081356259976319856095635165570951242738465263574196530002832838224925409916414747337808588217147787612261116386582749985930854049392933953960630988214160095749594574531062240251582972270007043831339322215104847444871389462974 cba_1=103564890343488279967812352928237987591797511562124696487535325455072696774055509080643862596648739514195875770239826660705492592071691472648200206878814446693527745554341584471324380085780404684161323663005093291958850086216055067821224753710213905212860418915059311394121921550051966020622156431774280321501 cba_2=113658447126988614855090828085246436414918154811758368122604373451159033675902636563241262692473265387095855289957831774844724058978343925705580746995943740554106046938963483329546105211596568482805579064863689442131577806842527224972152912439034953198579381644273659105124095330965484624259824835541524311643 Total encryption time for U_1 is 2,481 milliseconds. Encryption for U_3: $$cbc_1 = g^{rb_3}\ (mod\ p) where\ random\ rb_3\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cbc_2 = s_2(Y_3)^{rb_3}\ mod\ p$$ rb_3=799853416713812175854851486075722204688819727851117101874257242543557896134529058662757672518135756513226478532881048877118408151761567249916763883483903748345226314062375607142823687822569376965822094500331666818797296569153785838973170428997456594861910416091564214485147974447924151961532134816787174195510 cbc_1=339196092543018690655664945949674069581071317108527574435953138022458351814632839972555731179214945693494806277883136900814832437321617927676262046057489409565779325356596174631506149344077346826809958147042859938974367141963095688469079801449557372972492757842422963522991265221799429670972985417179101722450 cbc_2=216926562199351958198766517204647570322991694789135395873143037901556202772034867446247386358157698347814718469916493149543511994485237766880931616661883937723814267377055745360497260624590730828698368347565913690036882811269023995476351110648110679737864077858725623039801141406919861049271486240650191869600 Total encryption time for U_3 is 2,482 milliseconds.

(cont. on next page)

Table B.10. (cont.)

User U_2	Encryption for U_4: $$cbd_1 = g^{rb_4}\ (mod\ p) where\ random\ rb_4\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cbd_2 = s_2(Y_4)^{rb_4}\ mod\ p$$ rb_4=90143349875696264787635100362320063653142071661096740826555014605328 05857090992952796370684588668340132341068235791457022350418923294945117 8 00856785116262373482697067556638396230823172020601602989488644584538282 6 06147502846591684219083256820894405747028589234990281438253530069222119 6 35102109185035358635875 4 cbd_1=16237946350690621545573049187424966250211720212901088312243442905 60 00934214656544893548204527942669452727571115934244699081416822938105880 6 65896763592101939066746195008019212891089816116101660956247528640666885 3 52067642654506759635962499713878531293014558833491708288527763001868028 3 62335386635438507989359 6 cbd_2=10578609107620943592266934355165083816479833368133665877476700299 80 52240170145488835542347993131728404459933438401645000636088876523591609 0 28293016947003764330858496008683983502177598666165604191549131043984001 0 99223795387039954765808971035329511448687569063465068558960180707183267 5 36058455719166361722586784 Total encryption time for U_4 is 2,471 milliseconds. Total encryption time performed by user U_2 for all users is 7,434 milliseconds.

In Table B.11, U_3 encrypts his subsecret using ElGamal encryption and related user's public key who will receive the encrypted message. Encrypted messages are 1024-bit length. Total encryption time performed by U_3 for three users who will receive the message is approximately 7,933 milliseconds.

Table B.11. U_3's Encryption.

***User* U_3**	Encryption for U_1: $$cca_1 = g^{rc_1}\ (mod\ p) where\ random\ rc_1\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cca_2 = s_3(Y_1)^{rc_1}\ mod\ p$$ rc_1=20893427244861830045984468967121438306847421125869880603932429163564 10858797166181686952933610953122054798840748322109990921293640181090989 9 27208488944453350140365347020001622233834719112129830581160287410341621 5 26212511524188697320386849622963335008290383220621773301133033782902999 5 49902448292586095511243 3 cca_1=76734827792560641052808885218274364696866519334820542473048979837 31 89260105479119617191667774211900291858395059049442138087070338031642995 8 20852253235084198082006961768141134047238887012845882994486899731234545 0 29437034529622234736341279600839370053523518825594407010160856623448233 7 12908918488754044791560 97 cca_2=11635927742946225318200138586717971733321089741214886839696342684 40 28821672862426040920014062963284260083868798277530499260805040671373039 5 58999600173423927419632920280553088881725735540824075160256471088788423 6 55024263853713036164040745066097208462174592283756238767406012878486798 7 13247909907451586663799618 Total encryption time for U_1 is 2,591 milliseconds. Encryption for U_2: $$ccb_1 = g^{rc_2}\ (mod\ p) where\ random\ rc_2\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$ccb_2 = s_3(Y_2)^{rc_2}\ mod\ p$$ rc_2=47085482303605358024781675717053823874796291543822451891949662753282 88946576875987870546327589061985721914125668646544044549754905805131929 1 90636667181605547844989014120744576449780243255803304538319942654528320 1 57709424232180002005665357722749276065189607401537084352693158680300845 9 48054380371241960044983 3 ccb_1=25594327403200505888070878541585960864004852915273750248118551819 30 65977862838076213203818785164707333660107984999784462397096707273855495 9 15396701715111944079725489197890536933605358723556170215403866405313626 0 16973623736936014871220995191215665172937214815879811597344357766251728 7 09922772167075866809318 2 ccb_2=64670421458348441229421704977614588815041396775360824002716204330 97 78848543376602462758589476492429244911584849468530582323166376970461485 3 52324638482223189784799280886394450195459444214712578993124817015374924 8 47049770443001112463526809968842971837318482483049626403892654280389652 8 32233556656663129180070 38 Total encryption time for U_2 is 2,662 milliseconds.

(cont. on next page)

Table B.11. (cont.)

User U_3	Encryption for U_4: $$ccd_1 = g^{rc_4}\ (mod\ p) where\ random\ rc_4\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$ccd_2 = s_3(Y_4)^{rc_4}\ mod\ p$$ rc_4=365932675405003096044817453136638280403017260206611993457895487376079786835940462189835433528416070758114045515089466574635462870734296585795396585524855369807984662872207668115576460371481591059697326948259400206073607579383929827460303742103030725345303198576754798070836149816520468806314619548147330096 92 ccd_1=1123719326377191096530682343469931915443174624004411421401706710386994682336653745254380362979137922147215331845916608660703360733687625247201891361955719199387340662570085875909521824949047554563929164762587983940992469024787704667461817471408740671927662164790625273893763259473239581793977928516594835049 26 ccd_2=2091212362807010624540816689469091014911934968116844309097513888096857401364907094747161767880540663376452524985966593537017957535105511108852811671837700504838858187290378129812635234603891337139508259322140013355443332497219620500686268093094653313444003856977946079048391543507814721718361915592113668393 8 Total encryption time for U_4 is 2,680 milliseconds. Total encryption time performed by user U_3 for all users is 7,933 milliseconds.

In Table B.12, U_4 encrypts his subsecret using ElGamal encryption and related user's public key who will receive the encrypted message. Encrypted messages are 1024-bit length. Total encryption time performed by U_4 for three users who will receive the message is approximately 7,831 milliseconds.

Table B.12. U_4's Encryption.

User U_4	Encryption for U_1: $$cda_1 = g^{rd_1}\ (mod\ p) where\ random\ rd_1\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cda_2 = s_4(Y_1)^{rd_1}\ mod\ p$$ rd_1=4442001713745959994042872794875827392141446716765182304592131474856484314529826263028744780256508004350463706712978229289754730058692476020329954372348125575152956075887437925504243036800153905327339679114611314411419954432514025444108665282491544008010862237367624576331952737079448591064753035866010920402 0 cda_1=9089280169280286822278531551312242752253909937890955686872495983926475086245025336377049167869592311210165672044077269632615320006598774597285261914645695595120253020622515927557492505813723054803338620925410733100895124142646260068513443721968986287940356712222917081997955577293178813098503051823547074558 4 cda_2=1021312279713805972645352539798483255460384209766919555508560972831644135966827454704481512749241343949411095595772585727354984678497135259544056443445749356183903238105188728826268786655230098981728948831580401235573209839757126523789826167201779793334946828025625848942503847063529444518108579589534153678 08 Total encryption time for U_1 is 2,658 milliseconds. Encryption for U_2: $$cdb_1 = g^{rd_2}\ (mod\ p) where\ random\ rd_2\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cdb_2 = s_4(Y_2)^{rd_2}\ mod\ p$$ rd_2=2749154991784307799156044405200750069810217756643132452317289446393362107815735001496485567526416668261365304037060360599962097204770167077984202678803122618000339159914654076305145359799754726507418509604773393364861481923886452432775982784306276013211528761761525352430965695617998451633241582186856027440 1 cdb_1=3894316728074574066692616735825753355450550713123475782160687329685167203572227120015535716551826205333218716567721834426259257224840995550794342962079338131408011439077097726668821487910226944411157279923476423611419793378360501985488356009561909049245838537039163953218034534979249126245458955908608564687 0 cdb_2=1122070086722164479565279101784815659068412695344938748301268695320518275448969160004917063511346691994651636942522055099699312683211814525705964449655437526583428266169442663466402757778290440853113351639287470022423812086764808268110711130787885620263132057119897782597471936062722979339204776287785210125 Total encryption time for U_2 is 2,594 milliseconds.

(cont. on next page)

Table B.12. (cont.)

User U_4	Encryption for U_3: $$cdc_1 = g^{rd_3}\ (mod\ p) where\ random\ rd_3\ is\ \{1, 2, .., \mathrm{p} - 1\}$$ $$cdc_2 = s_4(Y_3)^{rd_3}\ mod\ p$$ rd_3=6808948909726556472921634570380238376309069086297418087635579878447498203976987049292800106800478135412406923339515182979241718259736591301082943566630974693036930965437660647083399364168131606032990165572504405312164101420822258567471827589375060218596654215207245838174234962593455352950480010434063801982 4 cdc_1=2653715219268543912948646830083784434885772724771474657986885700620023503064461431784574592976915355733049218566050143175296794908919938237191086219275262805608222614986955066273359599682603154926067401600197204350043059289624916532877693473091492026848113917022979555380884976488408877009584428885507927682 5 cdc_2=7549804667946216460450479700451763319877587754708081182281139527111650356158927371023492223604963169567072597610558043679344068920799620527001112105444628951711791952202263566493083746078537917912126049187650575791691716125211149682813646730081588010455955505379570047643669049614250984880950862560819161799 5 Total encryption time for U_3 is 2,579 milliseconds. Total encryption time performed by user U_4 for all users is 7,831 milliseconds.

In Table B.13 each user decrypts received encrypted message which outputs the sender's subsecret. Approximate decryption time of all three subsecrets for each user is 4,004 milliseconds. Specific computation time is available in the following Table.

Table B.13. Decryption of Received Subsecrets of Senders.

	Key Establishment
User U_1	Decryption for U_2's message $$s_2 = cba_2 \cdot ((cba_1)^{x_1})^{-1} \; mod \; p$$ s_2=725856418756471508011551369494525379929260032751408361747980029966661316774568223852911611254106506819164726165533941498818158803772188347757546240313554 3 Total decryption time is 1,348 milliseconds. Decryption for U_3's message $$s_3 = cca_2 \cdot ((cca_1)^{x_1})^{-1} \; mod \; p$$ s_3=607490291222222883884124266478308018226938090698148552656777420748303103904012961035465182730621577689599921318462679408835698050021082953354805224212674 1 Total decryption time is 1,367 milliseconds. Decryption for U_4's message $$s_3 = cda_2 \cdot ((cda_1)^{x_1})^{-1} \; mod \; p$$ s_4=802489097613302353560687016868295769718151445886074091775484649243830788214214027830686722356235863605740286955079997289288215687507234886244483383377324 1 Total decryption time is 1,367 milliseconds. Total decryption time for all received messages is 4,067 milliseconds.

(cont. on next page)

Table B.13. (cont.)

User U_2	Decryption for U_1's message $$s_1 = cab_2 \cdot ((cab_1)^{x_2})^{-1} \; mod \; p$$ s_1=9325956120950456012396966411050721266062076715274114314315248077218657079779600596816435174208891372960705333746565862799667975060862904938477787966341 14 Total decryption time is 1,372 milliseconds. Decryption for U_3's message $$s_3 = ccb_2 \cdot ((ccb_1)^{x_2})^{-1} \; mod \; p$$ s_3=6074902912222228838841242664783080182269380906981485526567774207483031039040129610354651827306215776895999213184626794088356980500210829533548052242126741 Total decryption time is 1,357 milliseconds. Decryption for U_4's message $$s_4 = cdb_2 \cdot ((cdb_1)^{x_2})^{-1} \; mod \; p$$ s_4=8024890976133023535606870168682957697181514458860740917754846492438307882142140278306867223562358636057402869550799972892882156875072348862444833833773241 Total decryption time is 1,269 milliseconds. Total decryption time for all received messages is 3,998 milliseconds.

(cont. on next page)

Table B.13. (cont.)

User U_3	Decryption for U_1's message $$s_1 = cac_2 \cdot ((cac_1)^{x_3})^{-1} \bmod p$$ s_1=93259561209504560123969664110507212660620767152741143143152480772186570797796005968164351742088913729607053337465658627996679750608629049384778796634114 Total decryption time is 1,410 milliseconds. Decryption for U_2's message $$s_2 = cbc_2 \cdot ((cbc_1)^{x_3})^{-1} \bmod p$$ s_2=72585641875647150801155136949452537992926003275140836174798002996666131677456822385291161125410650681916472616553394149881815880377218834775754624031355 43 Total decryption time is 1,303 milliseconds. Decryption for U_4's message $$s_4 = cdc_2 \cdot ((cdc_1)^{x_3})^{-1} \bmod p$$ s_4=80248909761330235356068701686829576971815144588607409177548464924383078821421402783068672235623586360574028695507999728928821568750723488624448338337732 41 Total decryption time is 1,270 milliseconds. Total decryption time for all received messages is 3,983 milliseconds.

(cont. on next page)

Table B.13. (cont.)

User U_4	Decryption for U_1's message $$s_1 = cad_2 \cdot ((cad_1)^{x_4})^{-1} \; mod \; p$$ s_1=93259561209504560123969664110507212660620767152741143143152480772186570797796005968164351742088913729607053337465658627996679750608629049384778796634114 Total decryption time is 1,312 milliseconds. Decryption for U_2's message $$s_2 = cbd_2 \cdot ((cbd_1)^{x_4})^{-1} \; mod \; p$$ s_2=72585641875647150801155136949452537992926003275140836174798002996661316774568223852911611254106506819164726165533941498818158803772188347757546240313554 3 Total decryption time is 1,296 milliseconds. Decryption for U_3's message $$s_3 = ccd_2 \cdot ((ccd_1)^{x_4})^{-1} \; mod \; p$$ s_3=60749029122222288388412426647830801822693809069814855265677742074830310390401296103546518273062157768959992131846267940883569805002108295335480522421267 41 Total decryption time is 1,362 milliseconds. Total decryption time for all received messages is 3,970 milliseconds.

In Table B.14 each user verifies received subsecrets by using commitment vector. Approximate calculation time of each user for subsecret verification is 2,245 milliseconds. Specific calculations of users are available in the following table.

Table B.14. Subsecrets Verification using Commitment Vector.

	Key Establishment
User U_1	U_1 verifies correctness of received subsecrets s_2, s_3 and s_4. Verification for s_2: $g^{s_2}(mod\ p) =$ 252114500680700631019890837607515580297843054754568213763763094298530803 909656904541264899548194611939886693036763442996187924885356805264582349 182738556749330052150695291921177717274387558083793380675989079763302215 986555924536539726648510215993229826809798394215720557780505694170922891 95920844666144273260 $\prod_{j=0}^{2} C_j^{2000^j} = (g^{a_0})^{2000^0} \cdot (g^{a_1})^{2000^1} \cdot (g^{a_2})^{2000^2} (mod\ p) =$ 252114500680700631019890837607515580297843054754568213763763094298530803 909656904541264899548194611939886693036763442996187924885356805264582349 182738556749330052150695291921177717274387558083793380675989079763302215 986555924536539726648510215993229826809798394215720557780505694170922891 95920844666144273260 Verification for s_3: $g^{s_3}(mod\ p) =$ 987251357715362171845979414191327371686414976418337078023605614273960902 724272030221728669998772571756660548245557055462766107959647690800462730 824787738253962550272769703169204591077991304444252320477885871200059239 153217071002203865289919933126571640259223902496341134298430418286919346 86073037555254183005 $\prod_{j=0}^{2} C_j^{3000^j} = (g^{a_0})^{3000^0} \cdot (g^{a_1})^{3000^1} \cdot (g^{a_2})^{3000^2} (mod\ p) =$ 987251357715362171845979414191327371686414976418337078023605614273960902 724272030221728669998772571756660548245557055462766107959647690800462730 824787738253962550272769703169204591077991304444252320477885871200059239 153217071002203865289919933126571640259223902496341134298430418286919346 86073037555254183005 Verification for s_4: $g^{s_4}(mod\ p) =$ 501379526328985561785835449286827020025421072913056723763219532361595190 251141922267972323614152053603670270311225752714074112534276564078381166 708761061384236579400752618535240166673282946763589870962843068800975367 417704633000009691645717715620851615360138377353690581357593429243112374 87593337447184971242 $\prod_{j=0}^{2} C_j^{4000^j} = (g^{a_0})^{4000^0} \cdot (g^{a_1})^{4000^1} \cdot (g^{a_2})^{4000^2} (mod\ p) =$ 501379526328985561785835449286827020025421072913056723763219532361595190 251141922267972323614152053603670270311225752714074112534276564078381166 708761061384236579400752618535240166673282946763589870962843068800975367 417704633000009691645717715620851615360138377353690581357593429243112374 87593337447184971242 Total computation time for verification of subsecrets is 2,218 milliseconds.

(cont. on next page)

Table B.14. (cont.)

User U_2	U_2 verifies correctness of received subsecrets s_1, s_3 and s_4. Verification for s_1: $g^{s_1}(mod\ p) =$ 814982349632022237078488014254493365207617001769928008613745139106963170712303831723722630394076758145179589226775506361229182413035264313912420677429028816772170469040150494808406630446072310567843091628364367381326434708520739200477409709449713648480151009619208767535524171395032285950186725956048906998 77 $\prod_{j=0}^{2} C_j^{1000^j} = (g^{a_0})^{1000^0} \cdot (g^{a_1})^{1000^1} \cdot (g^{a_2})^{1000^2} (mod\ p)=$ 814982349632022237078488014254493365207617001769928008613745139106963170712303831723722630394076758145179589226775506361229182413035264313912420677429028816772170469040150494808406630446072310567843091628364367381326434708520739200477409709449713648480151009619208767535524171395032285950186725956048906998 77 Verification for s_3: $g^{s_3}(mod\ p) =$ 987251357715362171845979414191327371686414976418337078023605614273960902724272030221728669998772571756660548245557055462766107959647690800462730824787738253962550272769703169204591077991304444252320477885871200059239153217071002203865289919933126571640259223902496341134298430418286919346860730375555254183005 $\prod_{j=0}^{2} C_j^{3000^j} = (g^{a_0})^{3000^0} \cdot (g^{a_1})^{3000^1} \cdot (g^{a_2})^{3000^2} (mod\ p)=$ 987251357715362171845979414191327371686414976418337078023605614273960902724272030221728669998772571756660548245557055462766107959647690800462730824787738253962550272769703169204591077991304444252320477885871200059239153217071002203865289919933126571640259223902496341134298430418286919346860730375555254183005 Verification for s_4: $g^{s_4}(mod\ p) =$ 501379526328985561785835449286827020025421072913056723763219532361595190251141922267972323614152053603670270311225752714074112534276564078381166708761061384236579400752618535240166673282946763589870962843068800975367417704633000009691645717715620851615360138377353690581357593429243112374875933374471849712 42 $\prod_{j=0}^{2} C_j^{4000^j} = (g^{a_0})^{4000^0} \cdot (g^{a_1})^{4000^1} \cdot (g^{a_2})^{4000^2} (mod\ p)=$ 501379526328985561785835449286827020025421072913056723763219532361595190251141922267972323614152053603670270311225752714074112534276564078381166708761061384236579400752618535240166673282946763589870962843068800975367417704633000009691645717715620851615360138377353690581357593429243112374875933374471849712 42 Total computation time for verification of subsecrets is 2,202 milliseconds.

(cont. on next page)

Table B.14. (cont.)

User U_3	U_3 verifies correctness of received subsecrets s_1, s_2 and s_4. Verification for s_1: $g^{s_1}(mod\ p) =$ 814982349632022237078488014254493365207617001769928008613745139106963170712303831723722630394076758145179589226775506361229182413035264313912420677429028816772170469040150494808406630446072310567843091628364367381326434708520739200477409709449713648480151009619208767535524171395032285950186725956048906998 77 $\prod_{j=0}^{2} C_j^{1000^j} = (g^{a_0})^{1000^0} \cdot (g^{a_1})^{1000^1} \cdot (g^{a_2})^{1000^2} (mod\ p)=$ 814982349632022237078488014254493365207617001769928008613745139106963170712303831723722630394076758145179589226775506361229182413035264313912420677429028816772170469040150494808406630446072310567843091628364367381326434708520739200477409709449713648480151009619208767535524171395032285950186725956048906998 77 Verification for s_2: $g^{s_2}(mod\ p) =$ 252114500680700631019890837607515580297843054754568213763763094298530803909656904541264899548194611939886693036763442996187924885356805264582349182738556749330052150695291921177717274387558083793380675989079763302215986555924536539726648510215993229826809798394215720557780505694170922891959208444666144273260 $\prod_{j=0}^{2} C_j^{2000^j} = (g^{a_0})^{2000^0} \cdot (g^{a_1})^{2000^1} \cdot (g^{a_2})^{2000^2} (mod\ p)=$ 252114500680700631019890837607515580297843054754568213763763094298530803909656904541264899548194611939886693036763442996187924885356805264582349182738556749330052150695291921177717274387558083793380675989079763302215986555924536539726648510215993229826809798394215720557780505694170922891959208444666144273260 Verification for s_4: $g^{s_4}(mod\ p) =$ 501379526328985561785835449286827020025421072913056723763219532361595190251141922267972323614152053603670270311225752714074112534276564078381166708761061384236579400752618535240166673282946763589870962843068800975367417704633000009691645717715620851615360138377353690581357593429243112374875933374471849712 42 $\prod_{j=0}^{2} C_j^{4000^j} = (g^{a_0})^{4000^0} \cdot (g^{a_1})^{4000^1} \cdot (g^{a_2})^{4000^2} (mod\ p)=$ 501379526328985561785835449286827020025421072913056723763219532361595190251141922267972323614152053603670270311225752714074112534276564078381166708761061384236579400752618535240166673282946763589870962843068800975367417704633000009691645717715620851615360138377353690581357593429243112374875933374471849712 42 Total computation time for verification of subsecrets is 2,248 milliseconds.

(cont. on next page)

Table B.14. (cont.)

User U_4	U_4 verifies correctness of received subsecrets s_1 s_2, s_3. Verification for s_1: $g^{s_1}(mod\ p) =$ 814982349632022237078488014254493365207617001769928008613745139106963170712303831723722630394076758145179589226775506361229182413035264313912420677429028816772170469040150494808406630446072310567843091628364367381326434708520739200477409709449713648480151009619208767535524171395032285950186725956048906998 77 $\prod_{j=0}^{2} C_j^{1000^j} = (g^{a_0})^{1000^0} \cdot (g^{a_1})^{1000^1} \cdot (g^{a_2})^{1000^2} (mod\ p)=$ 814982349632022237078488014254493365207617001769928008613745139106963170712303831723722630394076758145179589226775506361229182413035264313912420677429028816772170469040150494808406630446072310567843091628364367381326434708520739200477409709449713648480151009619208767535524171395032285950186725956048906998 77 Verification for s_2: $g^{s_2}(mod\ p) =$ 252114500680700631019890837607515580297843054754568213763763094298530803909656904541264899548194611939886693036763442996187924885356805264582349182738556749330052150695291921177717274387558083793380675989079763302215986555924536539726648510215993229826809798394215720557780505694170922891959208444666144273260 $\prod_{j=0}^{2} C_j^{2000^j} = (g^{a_0})^{2000^0} \cdot (g^{a_1})^{2000^1} \cdot (g^{a_2})^{2000^2} (mod\ p)=$ 252114500680700631019890837607515580297843054754568213763763094298530803909656904541264899548194611939886693036763442996187924885356805264582349182738556749330052150695291921177717274387558083793380675989079763302215986555924536539726648510215993229826809798394215720557780505694170922891959208444666144273260 Verification for s_3: $g^{s_3}(mod\ p) =$ 987251357715362171845979414191327371686414976418337078023605614273960902724272030221728669998772571756660548245557055462766107959647690800462730824787738253962550272769703169204591077991304444252320477885871200059239153217071002203865289919933126571640259223902496341134298430418286919346860730375555254183005 $\prod_{j=0}^{2} C_j^{3000^j} = (g^{a_0})^{3000^0} \cdot (g^{a_1})^{3000^1} \cdot (g^{a_2})^{3000^2} (mod\ p)=$ 987251357715362171845979414191327371686414976418337078023605614273960902724272030221728669998772571756660548245557055462766107959647690800462730824787738253962550272769703169204591077991304444252320477885871200059239153217071002203865289919933126571640259223902496341134298430418286919346860730375555254183005 Total computation time for verification of subsecrets is 2,311 milliseconds.

In Table B.15 each user applies Lagrange Interpolation. When a user verifies all subsecrets as shown in the Table B.14, then he/she can establish the common secret key using Lagrange Interpolation from any $t + 1$ subsecrets. Approximate calculation time

of each user for Lagrange Interpolation is 35 μ seconds. Specific calculations of users are available in the following table.

Table B.15. Key Establishment by Lagrange Interpolation.

	Key Establishment
User U_1	If verification succeeds, U_1 use lagrange interpolating formula for any $t+1$ subsecrets. Assume that $t+1$ set includes s_1, s_2, s_3. $a_0 = f(0) = s_1 \frac{ID_2}{ID_2 - ID_1} \cdot \frac{ID_3}{ID_3 - ID_1} + s_2 \frac{ID_1}{ID_1 - ID_2} \cdot \frac{ID_3}{ID_3 - ID_2} + s_3 \frac{ID_1}{ID_1 - ID_3} \cdot \frac{ID_2}{ID_2 - ID_3}$ a_0=838355556594409371859376473939134800750013303688371648654174765418316578029863928010946716563772272950582199151914080658849097286834249098372357246 2953520 Total computation time for lagrange interpolation is 40 microseconds.
User U_2	If subsecrets verification succeeds, U_2 use lagrange interpolating formula for any $t+1$ subsecrets. Assume that $t+1$ set includes s_2, s_3, s_4. $a_0 = f(0) = s_2 \frac{ID_3}{ID_3 - ID_2} \cdot \frac{ID_4}{ID_4 - ID_2} + s_3 \frac{ID_2}{ID_2 - ID_3} \cdot \frac{ID_4}{ID_4 - ID_3} + s_4 \frac{ID_2}{ID_2 - ID_4} \cdot \frac{ID_3}{ID_3 - ID_4}$ a_0=838355556594409371859376473939134800750013303688371648654174765418316578029863928010946716563772272950582199151914080658849097286834249098372357246 2953520 Total computation time for lagrange interpolation is 29 microseconds.
User U_3	If subsecrets verification succeeds, U_3 use lagrange interpolating formula for any $t+1$ subsecrets. Assume that $t+1$ set includes s_3, s_4, s_1. $a_0 = f(0) = s_3 \frac{ID_4}{ID_4 - ID_3} \cdot \frac{ID_1}{ID_1 - ID_3} + s_4 \frac{ID_1}{ID_1 - ID_4} \cdot \frac{ID_3}{ID_3 - ID_4} + s_1 \frac{ID_3}{ID_3 - ID_1} \cdot \frac{ID_4}{ID_4 - ID_1}$ a_0=838355556594409371859376473939134800750013303688371648654174765418316578029863928010946716563772272950582199151914080658849097286834249098372357246 2953520 Total computation time for lagrange interpolation is 31 microseconds.
User U_4	If subsecrets verification succeeds, U_4 use lagrange interpolating formula for any $t+1$ subsecrets. Assume that $t+1$ set includes s_4 s_1, s_2. $a_0 = f(0) = s_4 \frac{ID_1}{ID_1 - ID_4} \cdot \frac{ID_2}{ID_2 - ID_4} + s_1 \frac{ID_2}{ID_2 - ID_1} \cdot \frac{ID_4}{ID_4 - ID_1} + s_2 \frac{ID_1}{ID_1 - ID_2} \cdot \frac{ID_4}{ID_4 - ID_2}$ a_0=838355556594409371859376473939134800750013303688371648654174765418316578029863928010946716563772272950582199151914080658849097286834249098372357246 2953520 Total computation time for lagrange interpolation is 39 microseconds.

Table B.16 presents the summary of the original protocol (Feldman, (1987)) for four users. It approximately needs 3,03 milliseconds computation time for per user and 5,5 milliseconds computation time for the Leader.

Table B.16. Summary of Results of the Original Protocol.

		Leader -millisecond	***User U_1 -millisecond***	***User U_2 -millisecond***	***User U_3 -millisecond***	***User U_4 -millisecond***
Round-1	***Polynomial Generation***	0,021	--	--	--	--
	Commitment Calculation	5,483	--	--	--	--
	Subsecret Calculation	0,016	--	--	--	--
	Total	**5,520**	--	--	--	--
Round-2	***Subsecret Verification-Total***	--	**0,774**	**0,707**	**0,719**	**0,794**
Key Establishment	***Subsecret Verification***	--	2,218	2,202	2,248	2,311
	Lagrange Interpolation	--	0,040	0,029	0,031	0,039
	Total	--	**2,258**	**2,231**	**2,279**	**2,350**

Table B.17 presents the summary of the original protocol for four users. It approximately needs 19,468 milliseconds computation time for per user and 17,517 milliseconds computation time for the Leader.

Table B.17. Summary of Results of the Proposed Protocol.

		***Leader* -millisecond**	***User* U_1 -millisecond**	***User* U_2 -millisecond**	***User* U_3 -millisecond**	***User* U_4 -millisecond**
Round-1	***Polynomial Generation***	0,021	--	--	--	--
	Commitment Calculation	5,483	--	--	--	--
	Subsecret Calculation	0,016	--	--	--	--
	Signing on Commitment	0.058 SHA-512 + 1,548 ElGamal Signature	--	--	--	--
	Encryption of Subsecrets	10,391 ElGamal Encryption	--	--	--	--
	Total	**17,517**	--	--	--	--
Round-2	***Signature Verification***	--	3,402	3,402	3,402	3,402
	Decryption	--	1,283	1,365	1,360	1,338
	Subsecret Verification	--	0,774	0,707	0,719	0,794
	Subsecret Encryption	--	7,596	7,434	7,933	7,831
	Total	--	**13,055**	**12,908**	**13,414**	**13,365**
Key Establishment	***Decryption***	--	4,067	3,998	3,983	3,970
	Subsecret Verification	--	2,218	2,202	2,248	2,311
	Lagrange Interpolation	--	0,040	0,029	0,031	0,039
	Total	--	**6,325**	**6,225**	**6,262**	**6,320**