UNIVERSITY OF BUCHAREST
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Secret Sharing-based Group Key Establishment

PhD THESIS

Supervisor:
Prof.univ.dr.Adrian ATANASIU

PhD Candidate:
Ruxandra-Florentina OLIMID

Bucharest
2013

Author contact information:
ruxandra.olimid@fmi.unibuc.ro

Examination committee:
Conf. Dr. Radu Gramatovici (University of Bucharest, Romania - committee chair)
Prof. Dr. Adrian Atanasiu (University of Bucharest, Romania - supervisor)
Prof. Dr. Victor Valeriu Patriciu (Military Technical Academy, Romania - national referee)
Prof. Dr. Ferucio Laurentiu Tiplea (University of Iasi, Romania - national referee)
Prof. Dr. Bogdan Warinschi (University of Bristol, United Kingdom - international referee)

Abstract

Group applications permit multiple users to share resources or perform collaborative tasks while providing differentiate rights or responsibilities within the group. Examples include text communication, audio, video or web conferences, data sharing or collaborative computing. Security represents an important aspect for group applications. It is a challenging task to deal with, especially when the group size is large and the members are spread across different (location or networks) areas, with diverse protection mechanisms. In order to obtain the main cryptographic properties as confidentiality, authenticity and integrity it is usually required that the group members previously share a common secret group key. This is achieved as the output of a *group key establishment protocol*.

The thesis restricts to group key establishment protocols based on *secret sharing*, a primitive that divides a secret into multiple shares such that only authorized subset of shares allow reconstruction. Although secret sharing brings several advantages when it is used as a building block of group key establishment protocols, two important shortcomings currently exist: (1) several insecure proposals were published in the last years and (2) very few constructions rely on a security proof. We address both this issues in the present work.

The first part of the dissertation focuses on the underlying secret sharing schemes. We review a non-classical approach of secret sharing, define a new visual secret sharing scheme and analyze the possibility of malicious manufacturing of the sharing device. The second part of the thesis concentrates on group key establishment constructions that use secret sharing. We introduce a multitude of attacks against recent protocols and therefore highlight the necessity of security proofs. We review the properties that impose a sufficient level of security and briefly analyze the formal models of security. Finally, we introduce a new provable secure group key establishment protocol that achieves a good level of security while it maintains a constant number of communication rounds regardless the group size.

To all of those who guided my life and my work.

Contents

List of Abbreviations

AKE	=	Authenticated Key Exchange
BCP model	=	Bresson, Chevassut and Pointcheval's model
BCPQ model	=	Bresson, Chevassut, Pointcheval and Quisquater's model
CDH	=	Computational Diffie-Hellman problem
DDH	=	Decisional Diffie-Hellman problem
DLP	=	Discrete Logarithm Problem
DoS	=	Denial of Service
EKL	=	Ephemeral Key Leakage
eGBG model	=	extended GBG model
GBG model	=	Gorantla, Boyd and Gonzáles Nieto's model
GKA	=	Group Key Agreement
GKE	=	Group Key Establishment
GKT	=	Group Key Transfer
i.e.	=	*id est*
Ind-CPA	=	Indistinguishable under a Chosen Plaintext Attack
KCI	=	Key Compromise Impersonation
KDC	=	Key Distribution Center
KGC	=	Key Generation Center
KS model	=	Katz and Shin's model
LSSS	=	Linear Secret Sharing Scheme(s)
MA	=	Mutual Authentication
PPT	=	Probabilistic Polynomial Time
ROM	=	Random Oracle Model
SETUP	=	Secretly Embedded Trapdoor with Universal Protection
SSS	=	Secret Sharing Scheme(s)
s.t.	=	such that
TTP	=	Trusted Third Party
UC	=	Universally Composability
UF-CMA	=	UnForgeable under Chosen Message Attack
VSSS	=	Visual Secret Sharing Scheme(s)

List of Symbols and Notations

$\mathbb{N}, \mathbb{Z}, \mathbb{R}$	=	the set of natural numbers, integers, respectively real numbers
p, q	=	prime natural numbers
$\mathbb{Z}_p$	=	the set of integers modulo $p \in \mathbb{N}$, i.e. {0, . . . , p-1}
$\mathbb{Z}_p^*$	=	the multiplicative group of $\mathbb{Z}_p$, i.e. {1, . . . , p-1}
m	=	the number of possible users (participants / group members / parties / principals / entities / players)
$\mathcal{U} = \{U_1, \ldots, U_m\}$	=	the set of possible users
U	=	a generic user, $U \in \mathcal{U}$
(s_i)	=	the i-th session (of a protocol)
t	=	the number of participants to a given session
$\mathcal{U}_{(s_i)}$	=	the set of participants to the session (s_i)
k, $k_{(s_i)}$	=	the group key, the group key of the session (s_i)
$Sp(k)$	=	the space of all possible session keys
$\mathcal{D}$	=	the dealer
$\mathcal{AS}$	=	the access structure of a SSS
S	=	the shared secret
S^i	=	the i-th shared secret
$Sp(S)$	=	the space of all possible shared secrets
s_i, s_i^j	=	the i-th share of the secret S, the i-th share of the secret S^j
$Sp(s)$	=	the space of all possible shares
$S(x, y)$	=	the pixel (x, y) of the secret S (in case of VSSS)
$s_i(x, y)$	=	the pixel (x, y) of the i-th share of the secret S (in case of VSSS)
$x \leftarrow^R X$	=	a uniformly random choice of x from the set of values X
$A \rightarrow B : M$	=	a message M originating from A and sent to B
$A \rightarrow^* : M$	=	a broadcast message M originating from A
$\mathsf{H}, \mathsf{H}_i, h, h_i$	=	hash functions, $i \in \mathbb{N}$
(pk, sk)	=	a public-private key pair
(pk_i, sk_i)	=	a public-private key pair of the user U_i
$\Sigma.\mathsf{Sign}_{U_i}(M)$	=	the signing algorithm performed by U_i on the message M
$\Sigma.\mathsf{Verify}_{U_i}(M, \sigma)$	=	the verification algorithm on a message M and a signature σ originated from U_i
$\mathcal{K}$	=	the security parameter
q_r	=	the maximum number of possible queries to a hash function
q_s	=	the maximum number of (concurrent) sessions

$\neg\mathsf{X}$	=	the negation (or complementary event) of the event X
$Pr[\mathsf{X}]$	=	the probability of the event X
$Pr[\mathsf{X}\|\mathsf{Y}]$	=	the probability of the event X conditioned by the event Y
$H[\mathsf{X}]$	=	the entropy of X
$H[\mathsf{X}\|\mathsf{Y}]$	=	the entropy of X conditioned by Y
$\mathcal{A}$	=	a PPT adversary
Π_U^s	=	the instance of a user U in the session s
$\Pi_{U_i}^{s_i}$	=	the instance of the particular user U_i in the session s_i
$k_{U_i}^{s_i}$	=	the secret group key of $\Pi_{U_i}^{s_i}$
$\mathsf{pid}_{U_i}^{s_i}$	=	the partner id of $\Pi_{U_i}^{s_i}$
$\mathsf{sid}_{U_i}^{s_i}$	=	the session id of $\Pi_{U_i}^{s_i}$
$\mathsf{Adv}_{\mathcal{A}}^{\{\mathsf{AKE},\mathsf{MA},\mathsf{Con}\}}$	=	the advantage probability of the adversary $\mathcal{A}$ against the AKE security, MA security, respectively contributiveness of a protocol
$\mathsf{Game}^{\{\mathsf{AKE},\mathsf{MA},\mathsf{Con}\}}$	=	the adversarial AKE security game, MA security game, respectively contributiveness game
$\mathsf{Win}^{\{\mathsf{AKE},\mathsf{MA},\mathsf{Con}\}}$	=	the winning probability of the adversary $\mathcal{A}$ in $\mathsf{Game}^{\mathsf{AKE}}$, $\mathsf{Game}^{\mathsf{MA}}$, respectively $\mathsf{Game}^{\mathsf{Con}}$

List of Figures

List of Tables

Overview and Organization

Part I concentrates on secret sharing, but restricts only to notions and schemes that are used throughout the current work and personal contributions. It consists of the following chapters:

- **Chapter 1** gives a brief introduction of secret sharing schemes and their classification. Its main goal is to familiarize the reader with the notions required for the rest of the thesis and does not aim to give a complete survey on the field.
- **Chapter 2** describes first two classic secret sharing schemes, which will be used later as building blocks of other protocols. Second, it presents the untraditional perspective of secret m-sharing, an approach that brings several advantages to group key establishment. Last, the chapter introduces a novel visual secret sharing scheme, analyze its properties and gives a software implementation.
- **Chapter 3** extends SETUP (Secretly Embedded Trapdoor with Universal Protection) to secret sharing schemes, showing their susceptibility when enough randomness is employed. It analyzes the proposed attack and considers some prevention techniques. To exemplify the practical utility, the mechanism is particularized to Shamir's secret sharing scheme, the most popular in the literature.

Part II focuses on group key establishment protocols based on secret sharing. Considering the reasoning approach on security, it mainly divides into: informal and formal secure protocols. It consists of the following chapters:

- **Chapter 4** introduces the notion of group key establishment and its classification into group key transfer and group key agreement. It also links to secret sharing schemes, as building blocks for group key establishment.
- **Chapter 5** centers on informally secure group key establishment protocols based on secret sharing. It lists four such protocols that were introduced in the last years (2010-2013), describes several attacks against them and indicates possible countermeasures.
- **Chapter 6** concentrates on formally secure group key establishment protocols based on secret sharing. First, it describes the basics of formal proofs, the sequence of games technique, the GBG model and the limitations of current security models. Second, it presents two existing protocols that benefit of formal security proofs. The main part of the chapter introduces a novel protocol proved secure in the GBG model, which

maintains the same number of rounds as the existing work and benefits of several other advantages due to the underlying secret m-sharing scheme.

The **Research Contributions** section of each chapter summarizes the personal results and indicates the original work (except Chapters 1 and 4 which are introductory).

The last section, **Conclusions and Further Research**, resumes our results and suggests possible future work.

The **List of Symbols and Notations** indicates the notations used throughout the paper. Even though we sometimes remind the reader to refer to this list, we usually do not repeat the significance of symbols along the thesis. As a general remark, we also omit to specify each time the mathematical space in which the operations are performed when this is evident from the description (for example, we do not mention $\mod p$ every time we work within a finite field of order p).

The **Appendixes** review the required background and are referenced accordingly in the text, whenever needed:

- **Appendix A. Background on Cryptography** describes the cryptographic primitives used throughout the thesis. We assume that the reader is familiar with these notions, but remind them for completeness and notations.
- **Appendix B. Background on Mathematics** presents the mathematical concepts and lemmas applied along this work.

Part I

Secret Sharing

Chapter 1

Introduction to Secret Sharing

1.1 Overview

Blackley [7] and Shamir [68] independently introduced the idea of secret sharing as a solution to the key management problem: the key is split into multiple shares that are distributed to distinct parties such that only qualified sets of entities can recover it. Unlike the trivial method of creating copies (which also prevents loosing the key) secret sharing does not increase the risk of exposure. It also facilitates access control by dividing trust between multiple entities.

Definition 1.1. ***(Secret Sharing Scheme (SSS))****. A Secret Sharing Scheme (SSS) is a process or a protocol that divides a secret given as input into pieces called shares such that only specific subset of shares allow reconstruction of the original secret [46].*

More formal, it is a pair of algorithms $(\mathsf{Share}, \mathsf{Rec})$*, where:*

1. $\mathsf{Share}(S, m)$ *is a randomized sharing algorithm that on input a secret* S *outputs a set of* m *shares* $\{s_1, \ldots, s_m\}$*;*

2. $\mathsf{Rec}(s_{i_1}, \ldots, s_{i_t})$*,* $t \leq m$ *is a deterministic reconstruction algorithm from shares that outputs* S *if* $\{s_{i_1}, \ldots, s_{i_t}\}$ *is authorized and halts otherwise.*

A set of shares that allows reconstruction is addressed as *authorized* or *qualified*. The set of all authorized set of shares is called the *access structure* and it is denoted by $\mathcal{AS}$. Usually the shares are distributed to distinct participants and hence we will also refer to *autorized* (or *qualified*) set of users as the ones able to reconstruct the secret by putting their shares together; no other set of parties should be able to disclose it. An access structure is called *monotone* if any set containing an authorized set is also authorized. Informally, this means that if a group of users can recover the secret, then no matter how many others members join, the extended group will also be able to determine it. Monotone structures are a natural assumption in practice.

In general, SSS consists of multiple phases:

1. *Initialization.* It defines the environment of the scheme: the parameters, the possible space of secrets and shares and any other prerequisites;

2. *Sharing.* It describes the splitting algorithm of the secret into multiple shares. Usually, a TTP called *dealer* performs the sharing; however, dealer-free SSS also exist;

3. *Distribution.* It indicates the share(s) that are send to each participant via secure channels; we highlight that SSS require the existence of secure channels. Usually, during this phase, each participant receives one share. However, a single party may also own multiple shares (and therefore obtain more power within the group) or even an authorized set of shares (and hence becomes able to reconstruct the secret by himself[1]). More, some shares may be made public.

4. *Reconstruction.* It explicits the key recover formulas or algorithms performed to determine the secret from an authorized set of shares.

1.2 Classification

Regarding the quantity of information disclosed, SSS divide into: *perfect* and *non-perfect*.

Definition 1.2. ***(Perfect Secret Sharing)*** [46]. A secret sharing scheme is *perfect* if the shares corresponding to each unauthorized subset provide absolutely no information about the secret.

More formally, a perfect SSS[2]:

- permits perfect reconstruction for each authorized set, i.e. $H[S|A] = 0$, for all $A \in \mathcal{AS}$;
- discloses no information about the secret for each unauthorized set, i.e. $H[S|B] = H[S]$, for all $B \notin \mathcal{AS}$.

Based on the dimension of the shares, SSS categorize into: *ideal* and *non-ideal*.

Definition 1.3. ***(Information Rate)*** [46]. The *information rate* of a user is defined as the size of the shared secret divided by the size of the user's share. The *information rate* of a SSS is the minimum rate over all participants.

Definition 1.4. ***(Ideal Secret Sharing)*** [46]. A secret sharing scheme is *ideal* if it is perfect and its information rate equals 1.

In order to be practical, the amount of secret information distributed as shares should be as small as possible. For perfect secret sharing, the size of any share is larger than the size of the shared secret. Therefore, the information ratio is upper bounded by 1, which becomes the optimal case.

Considering the access structure, SSS classify into:

- *Special SSS.* The access structure satisfies some specific properties;

[1] This is the case of secret m-sharing described in Section 2.2.
[2] Please refer to Appendix B.1 for more details.

- *General SSS.* The access structure is not restricted in any way, except (usually) monotony, as required for the majority of practical schemes.

From the multitude of special SSS, we only mention two types that we will later use throughout this work:

- *Threshold SSS.* The access structure contains all sets with the cardinal at least t, $1 \leq t \leq m$. Informally, any t or more shares are enough to restore the secret, while few than t shares are not. We denote such a scheme by (t, m)-secret sharing;

- *Unanimous* (or *all-or-nothing*) *SSS.* The access structure contains only the set of all shares. Informally, all shares are required for secret reconstruction. We remark that a unanimous SSS is a (m, m)-threshold SSS.

Depending on the type of the shared secret, SSS separate into:

- *Bit-String SSS.* The shared secret is a sequence of bits, such as an element from a finite field or a text. These are the most popular SSS and present a high utility in practice. Next section introduces two examples of such schemes;

- *Visual SSS (VSSS).* The shared secret is an image. Section 2.3 refers to VSSS in more detail;

- *Audio or Video SSS.* The shared secret is an audio or video file.

Although multiple other classifications (e.g.: dynamic vs. static, single secret vs. multiple secret, proactive vs. retroactive, dealer vs. dealer-free) and special properties of secret sharing (e.g.: cheaters detection, cheaters identification, verifiability) exist, we have restricted to the notions we will use throughout the current work.

Chapter 2

Secret Sharing Schemes

2.1 Traditional Secret Sharing

The current section presents two SSS that we will later use in this work. Both schemes are bit-string SSS, were introduced more than 30 years ago and are very popular in the literature; this motivates us to call them *traditional*. All the computations are performed modulo q, where q is prime, but we omit to specify this every time since it is clear from the context.

2.1.1 Shamir (1979)

Shamir introduced in 1979 a threshold SSS [68]. Fig.2.1 describes it in detail.

The scheme is based on polynomial interpolation: a $t-1$ degree polynomial is uniquely defined by t points (x_i, y_i), $i = 1, \ldots, t$ with $x_i \neq x_j$, $i, j = 1, \ldots, t$, $i \neq j$. We remark that authorized users can derive the secret directly, without first computing the polynomial $f(x)$:

$$f(0) = \sum_{i=1}^{t} s_i \prod_{1 \leq j \leq t, i \neq j} \frac{x_j}{x_j - x_i} \tag{2.1}$$

Shamir's SSS satisfies two important properties: (1) it is *ideal*, since both the secrets and the shares lie in $\mathbb{Z}_q$ and (2) it is *perfect*, because less than t users obtain no information regarding the secret: for every value of the secret $S \in \mathbb{Z}_q$, the participants may reconstruct with the same probability a polynomial that passes through the $t-1$ points they own and $(0, S)$ (we have considered the worst case scenario, when $t-1$ participants cooperate).

2.1.2 Karnin et al. (1983)

Karnin et al. presented in the introduction of their paper a unanimous SSS, which we will refer for the rest of this work as Karnin et al.'s SSS [36]. Fig.2.2 describes it in detail.

The scheme satisfies the properties mentioned for Shamir's SSS: (1) it is ideal, since both the secrets and the shares lie in $\mathbb{Z}_q$ and (2) it is *perfect*, because less than m users obtain no information regarding the secret: for every value of the secret $S \in \mathbb{Z}_q$, the last share is computed with the same probability as $s_{m-1} = S - \sum_{i=1}^{m-1} s_i$ (we have considered the worst case scenario, when $m-1$ participants cooperate). We emphasize the high efficiency in computation.

Initialization. Let $q \geq m+1$ be a prime number and $S \in \mathbb{Z}_q$ the secret;

Sharing Phase. The dealer $\mathcal{D}$:

1.1. chooses $x_i \leftarrow^R \mathbb{Z}_q$, distinct, $i = 1, \ldots, m$ (and makes them public);

1.2. picks a $t-1$ degree random polynomial $f(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$, where $a_0 = S$ and $a_i \in \mathbb{Z}_q$, $i = 1, \ldots, t-1$;

1.3. computes $s_i = f(x_i)$;

Distribution Phase. The dealer $\mathcal{D}$:

2.1. sends (via a secure communication channel):

$$\mathcal{D} \rightarrow U_i : s_i,\ i = 1, \ldots, m;$$

Reconstruction Phase. Any set of t or more than t distinct participants $\{U_1, \ldots, U_t\}$ (without loss of generality, after a possible reordering):

3.1. compute the polynomial $f(x)$ by interpolation $f(x) = \sum_{i=1}^{t} s_i \prod_{1 \leq j \leq t, i \neq j} \frac{x - x_j}{x_i - x_j}$;

3.2. recover the shared secret $S = f(0)$.

Figure 2.1: Shamir's Secret Sharing Scheme [68]

Initialization. Let $q \geq m+1$ be a prime number and $S \in \mathbb{Z}_q$ the secret;

Sharing Phase. The dealer $\mathcal{D}$:

1.1. chooses $s_i \leftarrow^R \mathbb{Z}_q$, $i = 1, \ldots, m-1$;

1.2. computes $s_m = S - \sum_{i=1}^{m-1} s_i$;

Distribution Phase. The dealer $\mathcal{D}$:

2.1. sends (via a secure communication channel):

$$\mathcal{D} \rightarrow U_i : s_i,\ i = 1, \ldots, m;$$

Reconstruction Phase. The m participants:

3.1. recover the shared secret $S = \sum_{i=1}^{m} s_i$.

Figure 2.2: Karnin et al.'s Secret Sharing Scheme [36]

2.2 Secret m-Sharing

Traditionally in the literature, during the Distribution Phase each participant receives a single share and the secret can be recovered from an authorized set of shares belonging to distinct users. However, a different approach is possible: to split the same secret multiple times and give each participant a qualified set of shares (under the appropriate circumstances, usually at distinct moments in time). Two GKT protocols analyzed in Chapter 5 use this method.

Definition 2.1. ***(Secret m-Sharing).*** *A secret m-sharing scheme is a scheme that splits a secret m times into the same number of shares t using the same underlying SSS (i.e. the same* Share *and* Rec *algorithms).*

In other words, a *secret m-sharing scheme* runs a SSS m times on the same input S and t. Let $\mathcal{AS}$ be the access structure of a secret m-sharing scheme and $\mathcal{AS}_i$, $i = 1, \ldots, m$, be the access structure of the i-th run of the scheme which is based on. It is immediate that an authorized subset in any of the m splits remains authorized in the extended construction:

$$\mathcal{AS}_1 \cup \cdots \cup \mathcal{AS}_n \subseteq \mathcal{AS}. \tag{2.2}$$

Definition 2.2. ***(Perfect Secret m-Sharing)*** [62]. A secret m-sharing scheme is called *perfect* if the following conditions hold: (1) its access structure is $\mathcal{AS} = \mathcal{AS}_1 \cup \cdots \cup \mathcal{AS}_n$; (2) unauthorized subsets of shares provide absolutely no information about the secret.

Definition 2.2 states that no authorized subsets exist except the ones already authorized within the m perfect sharing instances and that combining shares originating from distinct runs give no additional information about the secret.

We affirm that perfect secret m-sharing exists. In order to support our claim we introduce in Fig.2.3 an example based on Karnin et al.'s scheme [36], used by Sun et al. [73]. We remark that the distribution of shares in the Distribution Phase is performed via secure channels and at different moments in time (i.e. the user U_j does not receive both s_1^j and

Initialization. Let $m = 2, t = 2, q \geq 3$ be a prime (large) number and $S \in \mathbb{Z}_q$ the secret;

Sharing Phase 1. The dealer $\mathcal{D}$:
1.1. chooses $s_1^1 \leftarrow^R \mathbb{Z}_q$;
1.2. computes $s_2^1 = S - s_1^1$;

Sharing Phase 2. The dealer $\mathcal{D}$:
1.1. chooses $s_1^2 \leftarrow^R \mathbb{Z}_q$;
1.2. computes $s_2^2 = S - s_1^2$;

Distribution Phase 1. The dealer $\mathcal{D}$:
2.1. sends:
$\mathcal{D} \rightarrow U_1 : s_i^1$, $i = 1, 2$;

Distribution Phase 2. The dealer $\mathcal{D}$:
2.1. sends:
$\mathcal{D} \rightarrow U_2 : s_i^2$, $i = 1, 2$;

Reconstruction Phase 1. User U_1:
3.1. computes $S = s_1^1 + s_2^1$;

Reconstruction Phase 2. User U_2:
3.2. computes $S = s_1^2 + s_2^2$;

Figure 2.3: Perfect Secret 2-Sharing Scheme based on Karnin et al.'s scheme [62]

s_2^j at the same time, $j = 1, 2$). The secret 2-sharing scheme is perfect because its access structure is the union of the access structures of the two runs of the underlying schemes ($\mathcal{AS} = \{\{s_1^1, s_2^1\}, \{s_1^2, s_2^2\}\} = \{s_1^1, s_2^1\} \cup \{s_1^2, s_2^2\} = \mathcal{AS}_1 \cup \mathcal{AS}_2$) and no information is leaked for other sets of shares.

2.3 Visual Secret Sharing

2.3.1 Overview

Visual secret sharing was introduced by Naor and Shamir in 1994 [51]. Unlike the previous described SSS, where the secret is an element from a finite field, *Visual SSS* (VSSS) permit the sharing of an image (black-and-white, gray or colored).

Definition 2.3. ***(Visual Secret Sharing Scheme (VSSS)).*** *A Visual Secret Sharing Scheme (VSSS) is a Secret Sharing Scheme (SSS) for which the secret and therefore the shares are images.*

The images are represented as a set of pixels, denoted by their coordinates ($S(x, y)$ is a pixel of the secret image S, $s(x, y)$ is a pixel of a share s) and defined by their color. Intuitively, the reconstruction consists in overlying the shares.

In case of black-and-white images, a pixel can be either black (defined as 1) or white (defined as 0), which permits to model visual overlying of the shares as bitwise OR. An example of visual sharing based on Naor and Shamir's scheme is given in Fig.2.4: (1) a white pixel is shared into four pixels, such that by reconstruction two white and two black pixels are obtained; (2) a black pixel is shared into four pixels, such that by reconstruction only black pixels are obtained. The scheme is not ideal, since the number of pixels of a share is four times larger than the number of pixels of the shared secret. As a consequence, the recovered image does not conserve the dimension of the original one. Also, the reconstruction decreases the contrast, as white is perceived as gray (two black and two white pixels). If desired, an additional step can perfectly restore the original image: each group of four pixels is replaced by a black pixel if no pixel is white, respectively by a white pixel, otherwise. We have considered a variant of Naor and Shamir's scheme that conserves the image ratio. Other similar constructions are also available [51]. We invite the reader to address the original paper for more information.

In case of color images, a pixel is usually represented in the *RGB model*[1]: any color is a combination of Red, Green and Blue, in different proportions; White (W) is obtained when this proportions are equal and Black (Bl) denotes the absence of any color. Fig.2.5 intuitively explains the model. Each color is coded as a triplet of bytes that denotes the proportion of appearance for each of the primary colors; we list the most important values in Table 2.1. The scheme described in the next subsection uses the RGB model.

[1]Others models exists, such as the *CMY (Cyan, Magenta, Yellow) model*, but we only use the *RBG model* throughout this work.

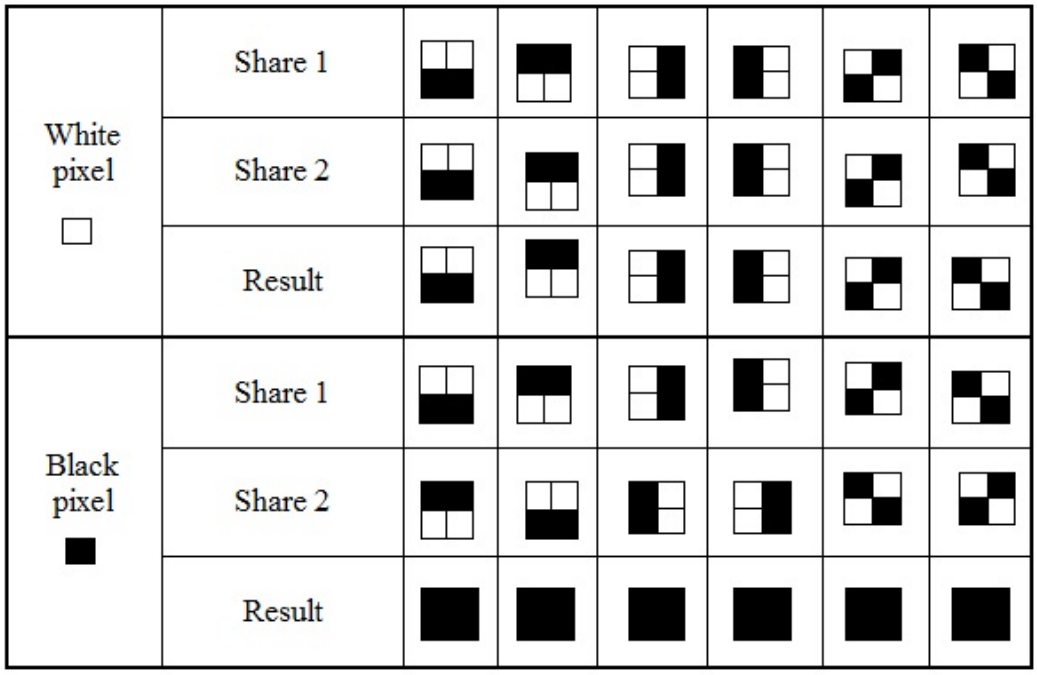

Figure 2.4: All Possible Sharings for a White and a Black Pixel in Naor-Shamir's VSSS ($m = 2$) [53]

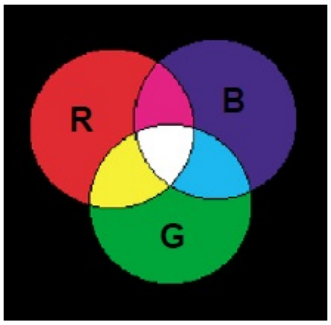

Figure 2.5: RGB Model

Table 2.1: RGB Coding

Color	Coding	Color	Coding
White	$W = (255, 255, 255)$	Black	$Bl = (0, 0, 0)$
Red	$R = (255, 0, 0)$	Yellow	$Y = (255, 255, 0)$
Green	$G = (0, 255, 0)$	Magenta	$M = (255, 0, 255)$
Blue	$B = (0, 0, 255)$	Cyan	$C = (0, 255, 255)$

2.3.2 A Novel VSSS

2.3.2.1 Description

We introduced in 2010 a VSSS that permits to share a black-and-white secret image into colored shares [52]. Fig.2.6 describes the scheme in detail, using the usual notations from the List of Symbols and Notations.

We briefly review the main idea: a black-and-white secret image is shared pixel by pixel into multiple color images such that: (1) if the pixel of the secret image is white, then three different shares with the corresponding pixel R, G and respectively B exist; (2) if the pixel of the secret image is black, then all the corresponding pixels of the shares are randomly choose from $\{R, G\}$, $\{R, B\}$ or $\{G, B\}$ such that at least two of different colors exist. Secret reconstruction is performed by overlaying the shares in the sense that they are added pixel by pixel, truncated at 255 (i.e. white added to R, G or B remains white) and recoded to black-and-white: (1) if the recovered pixel is white, no additional operation is required; (2) if the recovered pixel is colored, it converts to a black pixel.

Fig.2.7 exemplifies a possible set of shares in the case of a 2x2-pixel white image and a 2x2-pixel black image for $m = 4$ participants. It is easy to see that the sharing of a black pixel uses different couple of colors ($\{R, G\}$, $\{R, B\}$ or $\{G, B\}$) in order to maintain a similar repartition of R, G and B in the shares; otherwise, a participant could disclose some secret information from its only share.

Ito et al. also defined a VSSS that splits a black-and-white image into colored shares [33]. A significant add-on is that we eliminate the need of fixed matrices (idea inherited from Naor-Shamir) and randomly choose the color of the pixels for each share (with some restrictions). A second improvement is the conversion of the reconstructed image from color to black-and-white. This way, the contrast of the reconstructed image grows, getting the picture closer to the original one.

2.3.2.2 Analysis

As it derives from previous section, at least three shares are needed to split a white pixel (one R pixel, one G pixel and one B pixel, each one belonging to a component). Therefore, the scheme can only be used for three or more participants.

Each party stores an image of the same size of the shared image (in pixels). Although the scheme is *non-ideal* (three color coding introduce an overload to black-and-white coding), the scheme remains efficient.

We analyze next the quantity of information revealed to t participants that cooperate, $1 \leq t \leq m$:

- $t = 1$. A single user can find no information about the secret image from his own share because his image contains randomly chosen R, G or B pixels, which can lead to either white or black pixels by overlaying with other shares;
- $t = 2$. Two participants always reconstruct a black image because at least 3 shares are needed to obtain a white pixel;
- $2 < t < m$. More than two parties disclose additional information about the secret image. A white pixel obtained by reconstruction definitely corresponds to a white pixel

Initialization. Let S be the secret image;

Sharing Phase. The dealer $\mathcal{D}$, for every pixel $S(x, y)$:

1.1. if $S(x, y) = W$:

1.1.1. chooses $s_i(x, y) \leftarrow^R \{R, G, B\}$, $i = 1, \ldots, m$;

1.1.2. chooses $i_1, i_2, i_3 \leftarrow^R \{1, \ldots, m\}$ distinct;

1.1.3. $s_{i_1} \leftarrow R$, $s_{i_2} \leftarrow G$, $s_{i_3} \leftarrow B$;

1.2. if $S(x, y) = Bl$:

1.2.1. chooses $X \leftarrow^R \{\{R, G\}, \{R, B\}, \{G, B\}\}$;

1.2.2. chooses $s_i(x, y) \leftarrow^R X$, $i = 1, \ldots, m$;

1.2.3. chooses $i_1, i_2 \leftarrow^R \{1, \ldots, m\}$ distinct;

1.2.4. $s_{i_1} \leftarrow^R X$, $s_{i_2} \leftarrow X \setminus \{s_{i_1}\}$;

Distribution Phase. The dealer $\mathcal{D}$:

2.1. sends (via a secure communication channel):

$$\mathcal{D} \rightarrow U_i : s_i,\ i = 1, \ldots, m;$$

Reconstruction Phase. The m participants:

3.1. recover the shared secret S, where:

3.1.1. $S(x, y) = W$ if three distinct colored corresponding pixels $s_i(x, y)$ exist, $i = 1, \ldots, m$;

3.1.2. $S(x, y) = Bl$ if only two distinct colored corresponding pixels $s_i(x, y)$ exist, $i = 1, \ldots, m$.

Figure 2.6: Olimid's Visual Secret Sharing Scheme [52]

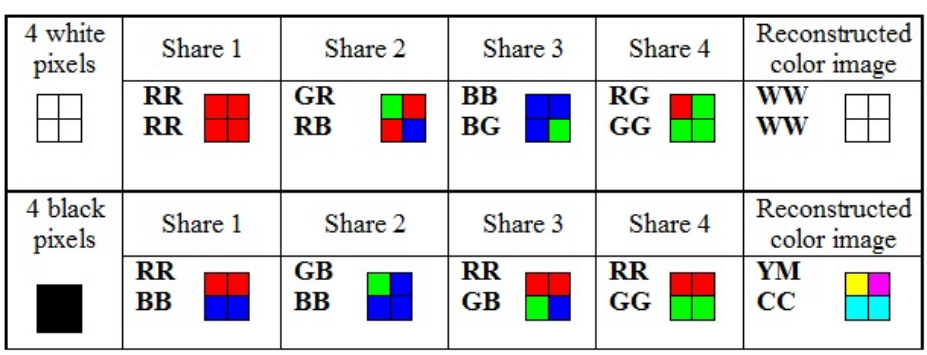

Figure 2.7: A Possible Sharing for 4 White and 4 Black Pixels in Olimid's VSSS ($m = 4$) [53]

of the original secret image because even if more shares are overlapped, white remains unchanged. However, no information is revealed about the pixels of the secret image that correspond to black pixels of the recovered image; this is because by adding a new share, a black pixel may convert to a white pixel (a black pixel corresponds to adding pixels of only two distinct colors; by adding a share with the corresponding pixel of the third color, it converts to white). The probability to correctly determine the white pixels increases with the number t of participants (the probability to meet R, G and B pixels on three different shares on the same position increases). Therefore, the scheme is monotone.

- $t = m$. All participants recover the shared secret image by construction.

In conclusion, the scheme is *perfect* for less than three users and it permits exact reconstruction of the secret image if all participants cooperate. For others number of collaborating parties, the reconstructed image quality depends of the randomly chosen colors of pixels and increases with t.

2.3.2.3 Implementation

We implemented the proposed VSSS in Python programming language using IDLE (Integrated DeveLopment Environment 2.6.6) as the development environment [75].

Python is a powerful dynamic programming language, accessible for all major operating systems. Besides other benefits, we mainly chose Python due to the fact that it provides a powerful and easy to use image library, named PIL (Python Image Library), also available in a free version. PIL provides image processing capabilities, available for multiple file formats. A complete documentation is available online [74].

Some of the special features that PIL provides and we used during the implementation include:

- different image modes, defining the type and color of a pixel in the image. From the multiple standard modes it supports, we employed: (1) `1` - black-and-white, stored as 8-bit pixels; (2) `RGB` - true color, stored as 3x8-bit pixels;
- operations on images at pixel level (set pixel value, define or read the size of an image as a 2-tuple consisting in the horizontal and vertical size in pixels: `<image>.size`);
- manipulation of color bands (R, G and B bands for color images);
- specific image functions as: open an existing image (`image.open(<image>, <mode>)`), create a new image (`image.new(<mode>,<size>)`), create an image based on multiple bands of color (`image.merge(<mode>, (<band1>, <band2>, <band3>))`).

Python also provides other functions that were mandatory for the implementation of the VSSS as:

- *intertools module*, which standardizes efficient and useful tools for combinatoric generators: permutations (`intertools.permutations(<m>)`) or combinations (`itertools.combinations(<m>,<t>)`);

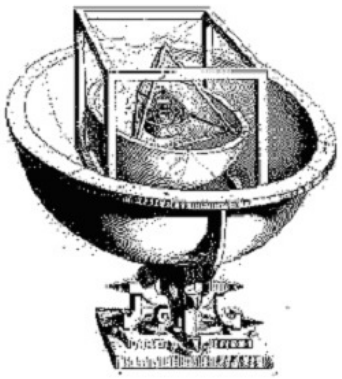

Figure 2.8: Input Test Image

- *random module*, which implements pseudo-random generators: `random.randrange`([<`start`>], <`stop`>, <`step`>) returns a randomly selected element in the specified range.

Fig.2.8 contains the original secret image. Fig.2.9 exemplifies possible shares for $m = 4$. Fig.2.10 indicates possible reconstructions for the case when two, three or all participants cooperate.

The practical implementation supports the theoretical analysis [52], [53]:

- the dimension of the shares equals in pixels the dimension of the secret[2];
- the reconstructed image maintains the ratio of the initial secret image;
- 1 share discloses no information about the secret;
- 2 shares derive to a totally black image;
- more than 2 shares provide enough information;
- all shares lead to the perfect reconstruction of the initial secret image.

2.4 Research Contributions

First, we reviewed an untraditional perspective of secret sharing that we call *secret m-sharing* and defined the notion of *perfect secret m-sharing* as a natural generalization of perfect secret sharing [62] (Section 2.2).

Second, we introduced a new VSSS, which allows to split a black-and-white secret image into m color shares [52] (Subsection 2.3.2). The construction reveals no information about the secret to one or two collaborating participants and permits perfect reconstruction when all parties cooperate. We implemented the scheme in Python programming language, using the PIL library and showed that practice supports the theoretical analysis [52], [53].

[2]Fig.2.8, Fig.2.9 and Fig.2.9 do not maintain the real size because of page layout reasons.

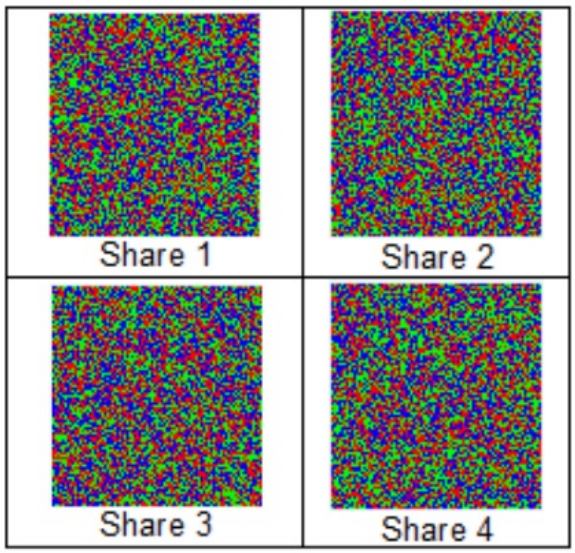

Figure 2.9: A Possible Sharing in Olimid's VSSS ($m = 4$) [52]

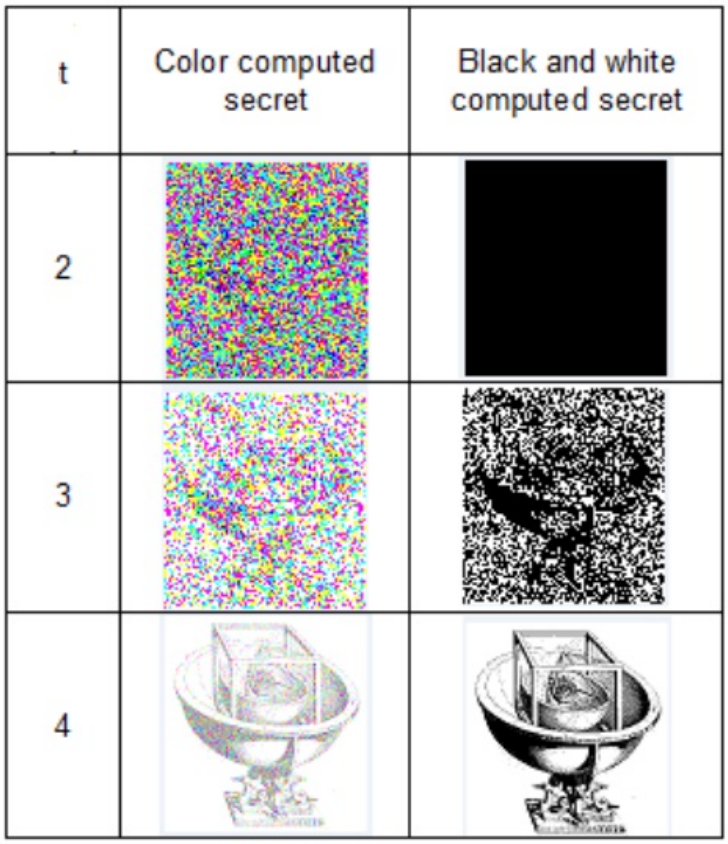

Figure 2.10: Sample of Reconstructed Images in Olimid's VSSS ($m = 4$) [52]

Chapter 3

SETUP against SSS using Random Number Generators

3.1 Overview

SETUP (Secretly Embedded Trapdoor with Universal Protection) mechanism was defined by Young and Yung [79]. It represents a malicious technique performed by the manufacturer of a cryptosystem that consists in implementing a subliminal channel that leaks encrypted secret information. The encryption is performed using the attacker's public key and therefore he is the only one that gains access to the leaked information.

From its introduction, SETUP attack was applied to different encryption systems, signatures schemes, key generation algorithms and e-voting [27], [28], [79], [80], [81], [82]. More recent, SETUP was defined for cryptosystems based on elliptic curves [48], [49]. Defense techniques against this kind of attack were also analyzed [41], [42].

SETUP assumes that the cryptosystem is implemented as a black box: a user can only access its input and output, while the internal design, the implementation of the algorithm and the internal memory are not externally accessible. This model permits the attacker to change the internal design in order to obtain a unique advantage. While the modification does not apparently affect the input or the output of the cryptosystem, the user cannot suspect any malicious behavior.

Definition 3.1. ***(Secretly Embedded Trapdoor with Universal Protection (SETUP))*** [80]. Assume that C is a black box cryptosystem with a publicly known specification. A *Secretly Embedded Trapdoor with Universal Protection (SETUP)* mechanism is an algorithmic modification made to C to get C' such that:

1. The input of C' agrees with the public specifications of the input of C;
2. C' computes efficiently using the attacker's public encryption function (and possibly other functions as well), contained within C';
3. The attacker's private decryption function is not contained within C' and is known only by the attacker;
4. The output of C' agrees with the public specifications of the output of C;

5. The output of C and C' are polynomially indistinguishable to everyone except the attacker;

6. After the discovery of the specifics of the SETUP algorithm and after discovering its presence in the implementation (e.g.: reverse-engineering of hardware tamper-proof device), users (except the attacker) cannot determine past (or ideally, future) keys.

A modified cryptosystem that implements SETUP is called *contaminated* [79].

3.2 SETUP Attack

The main goal of SETUP attack against a SSS is to offer the attacker an overwhelming advantage to reconstruct the shared secret. A trivial approach is immediate: in the Distribution Phase, the attacker receives the secret instead of a valid share. The honest participants will not be able to determine the dishonest behavior if the reconstruction is not performed. However, in case of reconstruction, the attack is revealed unless the adversary is able to provide a valid share, which is extremely improbable. It could seem that the problem can be solved if the attacker receives a valid share besides the secret. But this makes the implementation susceptible to traffic monitoring because the amount of information sent to this particular participant is doubled. More, this proposal does not achieve output indistinguishability from the genuine implementation: a party that has access to the contaminated SSS black box (the dealer) notices that one of the shares always copies the input.

The current section introduces a general method to implement SETUP against SSS that employ enough randomness, analyzes the proposal and considers some prevention techniques. Subsection 3.2.3 particularizes the attack for Shamir's SSS [55].

3.2.1 Description

We propose a technique for which (in the worst-case scenario) the attacker can reveal the secret with the knowledge of a few additional shares, while any participant other than the attacker (and if desired, his allies) needs all the shares belonging to an authorized subset to recover the secret [55].

The proposed SETUP attack becomes possible under the following assumptions:

1. The sharing mechanism is implemented as a black box that can store in a non-volatile memory information from multiple runs of the SSS;

2. The sharing mechanism generates a set of random values that are subsequently used to compute the shares (both these operations are performed within the black box);

3. The number of random values in each share is greater or equal to the number of the components of the shared secret (or the share itself can be considered a random value);

4. The shares are distributed via secure channels;

5. The attacker is always one of the participants;

6. Several secrets are shared.

Assumptions 1 and 4 derive from the definitions of SETUP, respectively SSS.

Assumptions 2 and 3 specify the properties a SSS must satisfy such that it becomes vulnerable to the proposed attack.

Assumption 5 states that the adversary must be a qualified participant, denoted as U_1 for the rest of this section (without loss of generality, after a possible reordering). In addition, he will always receive a specific share, which is computed in a special way within the contaminated device. In case of ideal SSS, the attacker needs access to at least one more share of a participant, which may be an ally.

The attack fails if only one secret is shared. However, assumption 6 does not restrict the applicability of the SETUP attack, since it is uncommon to consider that a sharing device is used only once or it is reset to factory defaults after each run.

Fig.3.1 describes the internal algorithm of the contaminated device and Fig.3.2 explains the reconstruction of the secret as performed by the attacker. We consider the notations from the List of Symbols and Notations and assume that the reader is familiar with the notion of Ind-CPA secure public key encryption scheme reviewed in Appendix A. The selection of the public key cryptosystem is performed such that: (1) it is Ind-CPA secure (security against

Input: m the number of participants, $S^1, S^2, \ldots$ the secrets to be shared;
Output: $s_1^i, s_2^i, \ldots, s_m^i$ the shares that correspond to the secret S^i, $i \geq 1$;

1: **if** i==1 **then**
2: $\alpha \leftarrow^R Sp(s)$;
3: $s_1^1 = \alpha$;
4: $\mathsf{H}(ID||s_1^1)$ is stored in the non-volatile memory;
5: $s_2^1, s_3^1, ..., s_m^1$ are computed accordingly to the genuine SSS, taking into consideration the share s_1^1;
6: **else**
7: $(s_1^i, ..., s_u^i) = \mathsf{Enc}(pk, S^i \otimes \mathsf{H}(ID||s_1^{i-1}))$, $1 \leq u \leq m$;
8: $\mathsf{H}(ID||s_1^{i-1})$ is replaced in the non-volatile memory by $\mathsf{H}(ID||s_1^i)$;
9: $s_{u+1}^i, s_{u+2}^i, ..., s_m^i$ are computed accordingly to the genuine SSS, taking into consideration the shares $s_1^i, ..., s_u^i$.
10: **end if**

Figure 3.1: SETUP Attack against Secret Sharing Schemes using Random Values [55]

Step 1. U_1 stores his share s_1^{i-1} from the previous execution;
Step 2. U_1 uses his own share s_1^i and $s_2^i, ..., s_u^i$ to compute the secret S^i:

$$S^i = \mathsf{Dec}(sk, (s_1^i, ..., s_u^i)) \otimes (\mathsf{H}(ID||s_1^{i-1}))^{-1}.$$

Figure 3.2: Attacker's Advantage to Recover the Shared Secret Originated from a Contaminated SSS [55]

passive adversaries is enough due to assumption 4); (2) it accept any value in $Sp(s)^u$ as a valid ciphertext (i.e. $\forall C \in Sp(s)^u$ and pk, $\exists M \in Sp(S)$ s.t. $\mathsf{Enc}(pk, M) = C$); (3) $u \in \mathbb{Z}^*$ is minimum. The high speed of the encryption represents an advantage. In addition, let $\otimes$ be the group operation in $Sp(S)$, ID a random secret bit string of considerable length that uniquely identifies the sharing device and $\mathsf{H} : \{0,1\}^* \to Sp(S)$ a collision resistant hash function.

We highlight the main idea: the contaminated sharing mechanism uses a public key encryption system; whenever a secret is given as input, the device encrypts its (processed) value using the public key of the attacker that is stored in the non-volatile memory of the black box and outputs it as one or more shares. The corresponding secret key is not stored within the device and cannot be disclosed from the public key as the encryption system is secure; therefore, the attacker is the only one that can benefit of the leaked information. Besides the public key of the attacker, the random string ID is also stored in the non-volatile memory of the contaminated device (and known by the attacker).

As we have already mentioned before, the attack assumes that multiple secrets are shared: one for each run of the SSS. Each execution traditionally consists of the three phases mentioned in Section 1.1: (1) *Sharing Phase* - a secret is given as input and the internal contaminated algorithm is executed; (2) *Distribution Phase* - the shares are distributed to participants via secure channels; (3) *Reconstruction Phase* - the participants collaborate to recover the secret or the adversary computes the secret as described in Fig.3.2.

3.2.2 Analysis

The attacker U_1 can benefit of an advantage only if he convinces $U_2, ..., U_u$ to divulge him their shares $s_2^i, \ldots, s_u^i$. We remark that on distinct runs of the protocol different participants may play the roles of $U_2, ..., U_u$, but the attacker must know their identity. If this is the case, then a predefined algorithm that maps these participants for each execution of the protocol must be implemented in the black box and known by U_1.

Possibly the most convenient situation is when $U_2, \ldots, U_u$ are fixed and they are allies of U_1. This way, they will always provide their secret shares to the attacker who becomes able to reconstruct the secret. Remark that although $U_2, \ldots, U_u$ are allies, they cannot find the secret unless they own the secret key sk. However, if it is desired that some of the allies have the same advantage in restoring the secret as the adversary, then they will be given the secret key.

The proposal is practical when u is small. For large values of u the attack may become difficult to implement or even useless. For example, in case of a (t, m) threshold scheme the attacker has no advantage in restoring the secret if $u \geq t$.

In case of ideal SSS the value of u is lower bounded by 2 because no Ind-CPA secure public key cryptosystem for which the dimension of the plaintext (the secret) equals the dimension of the ciphertext (the share) exists. In case of non-ideal SSS u may equal 1, since the dimension of the share is larger than the dimension of the secret; this means that the attacker restores the secret by himself, without any help from other participants.

No matter the case, U_1 has no advantage in recovering the first secret S^1. This is because s_1^1 must be uniformly random in $Sp(s)$ to achieve *output indistinguishability*, a notion that we explain in detail next.

A SETUP mechanism must be indistinguishable from the genuine one for everyone except the attacker. If the attack were easily identifiable, then the users would change the contaminated sharing device for a trustable one.

In order to prove that the proposed SETUP attack satisfies this requirement, we show that the output of the contaminated black box conserves the space of possible values and maintains the same distribution as the genuine device. The main idea of the demonstration is the following. The shares $s_1^i, ..., s_u^i$ are computed starting from a random value α using computations that conserve the space of possible values and maintain the same distribution: the group operation in $Sp(S)$ and the encryption function from $Sp(S)$ to $Sp(s)^u$. This makes them indistinguishable by construction. The rest of the shares are computed in the same way as in the genuine scheme, therefore are also indistinguishable. We give next the indistinguishability proof for the proposed SETUP attack.

Theorem 3.1. [55]. The contaminated SSS achieves output indistinguishability from the genuine SSS.

Proof. We prove by induction on the number of the shared secrets.

For the first secret S^1, the share s_1^1 of the attacker is a uniformly random value α in $Sp(s)$. All the other shares are computed accordingly to the genuine version of the scheme. Therefore, the indistinguishability property holds.

We assume by induction that for a fixed $i > 1$, $s_1^{i-1}, ..., s_m^{i-1}$ are indistinguishable from uniformly distributed values in $Sp(s)^m$. Since H acts like a random oracle, the value $\mathsf{H}(ID||s_1^{i-1})$ remains uniformly random in $Sp(S)$. The group operation in $Sp(S)$ conserves this property, hence $S^i \otimes \mathsf{H}(ID||s_1^{i-1})$ is also uniformly random. Due to the selection of the public key cryptosystem, which is Ind-CPA secure (therefore randomized) and covers the whole space of possible ciphertexts, $\mathsf{Enc}(pk, \cdot)$ applied to a random message maintains the indistinguishability of the ciphertext from a random value in $Sp(s)^u$. All the other shares are computed accordingly to the genuine version of the scheme. Therefore, the indistinguishability property holds.

□

A SETUP attack must also achieve confidentiality against reverse engineering. In this scenario, the content of the non-volatile memory, which contains the public key pk, the random string ID and the hashed value $\mathsf{H}(ID||s_1^{i-1})$, can be accessed. We highlight that s_1^{i-1} cannot be disclosed from the hashed value under the assumption that H is a strong hash function.

We show that the additional information revealed by reverse engineering brings no advantage: the scheme remains secure for unauthorized set of participants, even though they have access to the information stored in the non-volatile memory of the device.

Theorem 3.2. [55] The contaminated SSS remains as secure as the genuine SSS for anyone except the owners of the secret key (the attacker and, if desired, his allies).

Proof. Let $\mathcal{U}_r$ be a coalition of r participants, $1 \leq r \leq m$ that gain access to the public key pk, the string ID and the hashed value $\mathsf{H}(ID||s_1^{i-1})$ through reverse engineering.

If $\mathcal{U}_r$ is an authorized set of users, then the theorem holds since its members can recover the secret in both the genuine and the contaminated versions of the SSS.

If $\mathcal{U}_r$ is an unauthorized set of users, then its members are not able to compute the secret S^i in the genuine scheme. We will next analyze two possible scenarios for the contaminated version.

In the first scenario, the attacker or at least one of his allies are not members of $\mathcal{U}_r$. Even if they cooperate, they miss at least one contaminated share and therefore cannot recover the actual secret (not even the adversary $\mathcal{A}$ can recover the secret within this scenario). Possible useful information to disclose the previous shared secret could have been the value of the attacker's share s_1^{i-1}, but this is secure under the assumption that H is a cryptographic strong hash function.

In the second scenario, the attacker and all of his allies are members of $\mathcal{U}_r$. Therefore, in addition to the information from the previous scenario, the users have access to all contaminated shares. However, this provides them no advantage under the assumption that the used cryptosystem is Ind-CPA secure because they do not know the secret key sk and cannot derive it from the information stored within the non-volatile memory.

□

3.2.3 SETUP Attack against Shamir's SSS

We exemplify the applicability of the proposed SETUP attack against Shamir's SSS [55]. We motivate our choice by the fact that Shamir's scheme is the most popular in the literature.

Shamir's scheme is ideal (Subsection 2.1.1), therefore the attacker cannot succeed by himself (Subsection 3.2.2). We give an optimal solution, which requires a single ally.

Fig.3.3 describes Shamir's contaminated sharing algorithm and Fig.3.4 explains the reconstruction process as performed by the attacker.

The public key encryption scheme must satisfy the requirements mentioned in the previous subsections for $u = 2$ and $Sp(S) = Sp(s) = \mathbb{Z}_q$. In the original paper, Rivest, Shamir and Adleman gave a solution to restrict RSA trapdoor permutations to $\mathbb{Z}_q$ [64]: a plaintext is repeatedly encrypted until the result lies in $\mathbb{Z}_q$; similarly, a ciphertext is repeatedly decrypted until the plaintext lies $\mathbb{Z}_q$. The least-significant bit is a hardcore hc of the RSA family of trapdoor permutations. Both these primitives are used to build the cryptosystem using the general method of constructing public key encryption schemes from trapdoor functions[1] with a minor modification: in case the cryptosystem outputs the ciphertext outside $\mathbb{Z}_q^2$, it restarts the encryption using a different random value.

3.2.4 Prevention Techniques

We have shown in the previous subsections that, under certain condition, a SETUP attack may succeed in SSS that use enough random values. A prevention technique is trivial: the dealer intercalates between any two valid secrets to be shared a dummy secret of a random value: $S^1, S'^1, S^2, S'^2, \ldots$. For the dummies executions, he does not distribute the shares to the participants, but only runs the Sharing Phase. As the attacker can never know the previous shared secret or the corresponding share, he has no advantage in computing the current secret. The drawback of this solution is clear: it doubles the computational costs.

[1] Please refer to Appendix A.1, Section A.1.4 for more details.

Input: m the number of participants, $S^1, S^2, \ldots$ the secrets to be shared, $q \geq m+1$ a prime;
Output: $s_1^i, s_2^i, \ldots, s_m^i$ the shares that correspond to the secret S^i, $i \geq 1$;

1: **if** i==1 **then**
2: $x_j \leftarrow^R \mathbb{Z}_q$ are chosen, $j = 1, \ldots, m$;
3: $s_1^1 \leftarrow^R \mathbb{Z}_q$ is chosen;
4: a $t-1$ degree polynomial is generated s.t. $f^1(x) = S^1 + a_1^1 x + \ldots + a_{t-1}^1 x^{t-1}$ and $f(x_1) = s_1^1$;
5: $s_j^1 = f^1(x_j)$, $j = 2, \ldots, m$;
6: $H(ID||s_1^1)$ is stored in the non-volatile memory;
7: **else**
8: $(s_1^i, s_2^i) = \mathsf{Enc}(pk, S^i \cdot \mathsf{H}(ID||s_1^{i-1}))$;
9: $\mathsf{H}(ID||s_1^{i-1})$ is replaced in the non-volatile memory by $\mathsf{H}(ID||s_1^i)$;
10: $a_j^i \leftarrow^R \mathbb{Z}_q$, $j = 3, \ldots, t-1$;
11: a_1^i and a_2^i are computed as the solutions of:

$$\begin{cases} s_1^i &= f^i(x_1) \\ s_2^i &= f^i(x_2) \end{cases}$$

where $f^i(x) = S^i + a_1^i x + \cdots + a_{t-1}^i x^{t-1}$ is a $t-1$ degree polynomial;
12: $s_j^i = f^i(x_j)$.
13: **end if**

Figure 3.3: SETUP Attack against Shamir's Secret Sharing Scheme [55]

Step 1. U_1 stores his share s_1^{i-1} from the previous execution;
Step 2. U_1 uses his own share s_1^i and s_2^i to compute the secret S^i:
$S^i = \mathsf{Dec}(sk, (s_1^i, s_2^i)) \cdot (\mathsf{H}(ID||s_1^{i-1}))^{-1}$.

Figure 3.4: Attacker's Advantage to Recover the Shared Secret Originated from a Contaminated Shamir's SSS [55]

We have introduced in Subsection 3.2.1 a restriction in the sense that the attacker should always be the first participant (U_1). The order of participants is not important, so we could have considered any other participant as being the attacker. The true restriction is given by the fact that the contaminated sharing mechanism must distribute the proper shares to the attacker and his allies: if the attacker does not receive the maliciously computed share that contains the encrypted secret leaked as a random value or he does not know the shares of his allies, the attack fails. The precise distribution of the contaminated shares to the corresponding participants is possible if the sharing device is in charge of the shares distribution as well. Therefore, an immediate protection against a SETUP attack is to use a sharing device that computes the shares, but does not distribute them to the participants.

The distribution remains the responsibility of the dealer.

Although the dealer is in charge with the distribution, his improper behavior may maintain the applicability of the attack. Consider for example that he inputs the secret into the contaminated sharing device, maps each output to a participant in the sense that the first output represents the share that will be given to a participant, the second output represents the share that will be given to another participant and so on and maintains the same mapping for several runs. The probability that the attack succeeds is given by the probability that the attacker manages to restore the valid ciphertext as a concatenation of his and his allies shares:

$$Pr_{\mathcal{A}} = \frac{1}{A_m^u} = \frac{(m-u)!}{m!}. \tag{3.1}$$

Of course, the success of the attack considerable diminishes in case the dealer performs the distribution as described before. For example, SETUP attack succeeds with probability 5% in Shamir's secret scheme with $m = 5$ participants.

However, a slightly modification of the attack raises the success probability in the proposed scenario by a factor of maximum $[m/u]$: as many groups of shares as possible are computed in the same way as the shares of the attacker and his allies. This way, no matter how the dealer performs the mapping, if the attacker and his allies receive as input such shares, then the adversary will be able to recover the secret. The attack remains secure against reverse engineering and maintains output indistinguishability.

The best prevention technique against the proposed SETUP attack is that the dealer randomly maps the shares to the participants for each run. The existence of a SETUP contaminated device that permits the attacker to recover the secret in the conditions of a random mapping remains an open problem.

3.3 Research Contributions

We showed that, under certain conditions, SETUP (Secretly Embedded Trapdoor with Universal Protection) attack can be embedded in secret sharing schemes that use random number generators in order to give the attacker an overwhelming advantage to access the shared secret [55] (Section 3.2): in case of ideal schemes the attack is performed by a coalition of a few participants (within at least one is the attacker), while in case of non-ideal schemes the attacker's knowledge may be enough to reveal the secret. We analyzed the properties of the general method of attack and considered some possible prevention techniques. In order to exemplify the applicability of our proposal, we successfully embedded SETUP in the most popular secret sharing scheme: Shamir's scheme becomes susceptible in case the attacker has only one ally.

Part II

Group Key Establishment Protocols based on Secret Sharing

Chapter 4

Introduction to GKE

4.1 Overview

In order to benefit of secure group-oriented applications, multiple users need to share a private key, which is obtained as the output of a *Group Key Establishment (GKE)* protocol.

Definition 4.1. ***(Group Key Establishment (GKE))*** [46]. A *Group Key Establishment (GKE)* is a process or a protocol whereby a shared secret becomes available to more parties, for subsequent cryptographic use.

GKE are also called *Conference Key Establishment* or *Multiparty Key Establishment* [46], [67].

The main goal of GKE is to establish a common key between the authorized members of a group, without disclosing it to other parties. The *authorized* participants to the protocol are also addressed as *qualified*, *legitimate* or *privileged*. A protocol runs for multiple times, named *sessions*. Each session is uniquely identified by a *session id*, which can be computed during the execution of the protocol or given in advance by the environment. We call *session key* the shared secret derived after one execution of the protocol. It only persists for a short period of time, a natural approach in cryptography (the probability to reveal a key increases with its period of usage). To become eligible to take part to protocol sessions, users must first register within the group. After registration, they acquire a *long-lived* or *long-term* secret, which they will later use to derive the session keys they are qualified for.

Menezes and al. motivate the importance of GKE [46] - in addition to its main target (to establish the group key that is necessary to implement cryptographic properties, like confidentiality or group authentication), it:

- limits the quantity of messages encrypted under the same key (by refreshing the group key for each session), which makes the system more powerful against cryptanalytic attacks;
- restricts information disclosure in time if the key is compromised (for one session);
- avoids the long-term storage of a large number of secret keys by creating keys at demand;

- permits independence between communication sessions and applications.

In general, GKE protocols present multiple phases:

1. *Initialization.* It defines the environment of the protocol: the parameters, the space of all possible keys and any other prerequisites.

2. *Users Registration.* It assigns group membership to users. Depending on the scenario, after registration, a user may for example share a secret key (or password) with a trusted group authority or may generate a certified long-lived public-private key pair for later signing purposes.

3. *Execution.* It describes the cryptographic algorithm, including the performed computations and the exchanged messages. It usually consists of multiple rounds of communication between principals.

4. *Key Computation.* It explicits the key computation formulas or algorithms performed by a party to derive the key from the knowledge he gained after the Execution Phase. It is sometimes integrated within a round of the execution phase.

5. *Key Confirmation.* It confirms that all the intended members actually own the key and no other except them does. Although it is an optional phase, it is usually performed for security reasons.

4.2 Classification

GKE protocols divide into two classes: *Group Key Transfer (GKT)* and *Group Key Agreement (GKA)*.

Definition 4.2. ***(Group Key Transfer (GKT))*** [46]. A *Group Key Transfer (GKT)* protocol or mechanisms is a Group Key Establishment (GKE) technique where one party creates or otherwise obtains a secret value, and securely transfers it to the others.

GKT are also called *Group Key Transport* or *Group Key Distribution* [44], [45], [46].

Definition 4.3. ***(Group Key Agreement (GKA))*** [46]. A *Group Key Agreement (GKA)* protocol or mechanism is a Group Key Establishment (GKE) technique in which a shared secret is derived by 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.

GKA are also called *Group Key Exchange* [44], [45].

The main difference between the two classes derives directly from their definitions: GKT requires the existence of a privileged party to select and distribute the key, while GKA does not, the key being computed as the result of the collaboration of legitimate participants via exchanged messages. Unlike GKA, in which the key is derived only by the cooperation of internal group members, GKT permits the entity that generates the key to be an outsider as well (i.e. not a group member). This entity has various names in the literature, such as:

Trusted Third Party (TTP), *Key Generation Center (KGC)*, *Key Distribution Center (KDC)* or *Group Controller* [2], [44], [45], [46]. The naming differs according to the precise function it fulfills. For example, it may exist an entity that generates the key (KGC) and an entity (distinct or not) that distributes it to the authorized members (KDC). For the rest of this work we will mainly refer to the KGC as a single party that performs both key generation and distribution.

The KGC must be trusted by all participants as honest in the sense that it selects a fresh key (a uniformly random value that has never been used before) and does not reveal it to unqualified parties. This trust assumption is not required for GKA protocols, which do not demand the existence of a privileged party to select the key, but compute it by equal contribution of the principals. However, regardless of the GKE type, a trust relation is mandatory: the qualified participants to a session trust each other that none of them discloses the shared key. Otherwise, the confidentiality of the protocol is violated by default. We remark that during the execution of a GKA protocol, participants do not trust each other and suspect their partners may intend to get control over the group key value - Definition 4.3 considers this behavior. Due to less trust assumptions, GKA usually satisfies stronger security.

GKT assumes (in general) the existence of secure communication channels between the KGC and each user in the Users Registration Phase: the long-lived key of a participant usually consists in a pre-shared secret (symmetric key or password) with the KGC. By contrast, GKA do not impose such an assumption: the long-lived keys of group members are usually public-private pairs uses for signing (or sometimes, for asymmetric encryption).

Regarding the contribution type of the participants to the GKA (a nonce or the long-lived key), GKA split into [67]:

- *Interactive GKA.* Group members contribute to the key generation with fresh values for each session (nonces). They require exchanged messages between the participants and therefore impose that all parties are online for the execution of the protocol.

- *Non-Interactive GKA.* Group members contribute to the key generation with their own public long-live keys. Examples include the original Diffie-Hellman protocol [23][1] and Joux tripartite protocol [35]. Unlike the Interactive GKA, their main advantage is that a user can determine the common key even if the others are offline.

GKT protocols are primary used in application with centralized control. Based on the particularity of the entity that generates and distributes the key, GKT can further divide into [2]:

- *Centralized GKT.* It involves a single entity that generates and distributes the key. Some of the drawbacks of this category include [2], [66] : (1) the KGC must be always online; (2) the KGC must maintain a secure communication channel with each group member; (3) the KGC may easily be the target of a DoS attack; (4) the computational power of the KGC limits the number of users he can handle.

[1] We mention Diffie-Hellman due to its popularity, although it applies for only two participants and therefore it cannot be considered a group protocol.

- Distributed GKT. It involves a single entity that generates the key, while the distribution is performed by one of the qualified members of the group, which is dynamically selected for each execution of the protocol. Although this category is more suitable (especially for unreliable networks), it preserves the first two drawbacks of the Centralized GKT and only diminishes the last two. As a disadvantage, we mention that the construction and maintenance of such structures becomes more complicated (especially in case of dynamic groups).

On the contrary, GKA protocols are not limited by the previously specified disadvantages of GKT: they do not provide a single point of trust, are more robust and the computational cost is in general balanced between all the participants to the protocol.

Despite all the mentioned advantages of GKA over GKT, one class or the other may suit best depending on the application needs or constraints (security requirements, computational resources and transmission costs). Due to the fact that parties do not necessary have to communicate between themselves (but only with the KGC, who performs most of the computation), the computational and transmission costs of GKT protocols are usually lower that those of GKA protocols. In addition, the design of GKT is in general less challenging.

Independent of the given classification, GKE may by considered in the context of *static* or *dynamic* groups: a *static GKE* does not provide special mechanisms for membership changes, while a *dynamic GKE* includes particular operations such as joining or leaving the group. In case that the authorized group of participants modifies, a static GKE must restart the process all over, while a dynamic GKE performs additional, but more efficient operations to update the group key and make it available and secure in the new settings. For the rest of our work we restrict to static GKE.

4.3 GKE based on Secret Sharing

A general mechanism for defining GKE protocols is immediate [46]: KGC generates a fresh group key and sends its encrypted value under the appropriate key to each legitimate participant. Hence, any authorized user decrypts and finds the key, while it remains secure against unauthorized parties. We have assumed that an authentication mechanism exists, such that the KGC or the users cannot be impersonated and the message cannot be modified during transmission. This trivial solution becomes inefficient for large groups: KGC must perform m encryptions and send m messages, where m is the number of qualified participants. In case a symmetric encryption scheme is used to decrease the computational costs (rather than an asymmetric encryption scheme), a supplementary assumption appears: each registered group member must previously share a secret with the KGC.

Secret sharing is used in GKE protocols to avoid such disadvantages, allowing efficient constructions: users may communicate through broadcast channels only, the computation of the key may consist in simple linear equations, the number of rounds remains constant regardless the group size. In addition, they introduce several benefits: a convenient way to differentiate between principals power within the group, delegation of duties by passing shares to other participants, group authentication instead of entity authentication, cheating detection and simple management of group sizing using the accepted threshold [63].

Various GKE constructions based on secret sharing schemes exist in the literature. Blom proposed an efficient GKT protocol in which every two users share a common private key that remains hidden when less than t users cooperate [8]. Blundo et al. generalized Blom's protocol by allowing m users to share a private key, while it remains secure for a coalition of up to t users [10]. Fiat and Naor improved the construction even more by permitting any subset of users to share a common key in the same conditions [26]. Pieprzyk and Li [43], [63], Bresson and Catalano [13], Cao et al. [20] , Harn and Lin [30], Yuan et al. [83] introduced GKE protocols based on Shamir's secret sharing scheme. Some other examples from literature include Sáez's protocol [65] (based on a family of vector space secret sharing schemes) and Hsu et al. [32], Sun et al. [73] and Olimid's [62] protocols (based on linear secret sharing schemes).

Chapter 5

Informally Secure GKE based on Secret Sharing

5.1 Informal Security Notions

The current section reviews the informal security goals GKE protocols must achieve and summarizes the adversarial and attack types GKE protocols should stand against [58].

5.1.1 Informal Security Requirements

A GKE protocol should satisfy a set of properties, which we informally recall next [44], [45].

Key confidentiality (also called *key privacy*, *key secrecy* or *non-disclosure*) [19], [23], [24], [34] guarantees that it is (computationally) infeasible for an adversary to compute the group key. The stronger notion of *known key security* [18], [77] assures that key confidentiality is maintained even if the attacker somehow manages to obtain group keys of previous sessions. *Backward secrecy* [17], [44] conserves the privacy of future keys regardless the adversary's actions in the past sessions. Correspondingly, *forward secrecy* [17], [24] imposes that the adversary actions in future runs of the protocol do not compromise the privacy of previous session keys (i.e. a key remains secure in the future).

Key selection must satisfy specific properties. *Key freshness* [46] requires that the group key has never been used before. The related concept of *key independence* [34], [72] imposes that no correlation exists between keys from different sessions; this means that (cooperation between) authorized participants to distinct sessions of the protocol cannot disclose session keys they are unauthorized for. In addition, *key randomness* warrants *key indistinguishability* from a random number and hence *key unpredictability*. Two other important security requirements regarding the key value exist: *key integrity* [34], which attests that no adversary can modify the group key and *key consistency*, which prevents different players to accept different keys.

Group member authentication represents a mandatory condition for group cryptographic protocols. *Entity authentication* [5] confirms the identity of a participant to the protocol to the others. Similarly, *unknown key share resilience* [24] restricts a user to believe that the key is shared with one party when in fact it is shared with another. *Key compromise impersonation (KCI) resilience* [6], [29] prevents an attacker who owns the long-lived key of

a participant to impersonate other parties to him. The stronger property named *ephemeral key leakage (EKL) resilience* (EKL) [85] avoids an adversary to recover the group key even if he discloses the long-lived keys and ephemeral keys of parties involved except both these values for participants in the test session[1].

(Implicit) Key authentication [46] limits the possible owners of the group key to the legitimate participants; this means that no other party except the qualified users is capable to compute the key, but it does not necessary mean that all legitimate principals actually own it. Another property, called *key confirmation* [12], [46] certifies that all authorized members actually have the key; however, it does not claim that no other party own the same key. *Explicit key authentication* (or *Mutual Authentication* (MA)) [4], [14], [17], [46] combines these notions and ensures that all qualified participants to the protocol have actually computed the group key and no one else except them have.

Unlike GKT, GKA protocols must satisfy additional properties. *Key contributiveness* assures that each party equally contributes to the key and hence guarantees its freshness [3]. *Key integrity* presumes, in addition to the previous definition, that the key is a function of only the contributions of the authorized members and no external contribution to the key establishment is tolerated (even if it brings no additional knowledge to the adversary) [3]. *Complete group key authentication* guarantees that the authorized participants compute the same key only if all the qualified parties have been contributed to its generation [3].

For more information on informal security requirements, we invite the reader to refer to [44] and [45].

5.1.2 Adversaries and Attacks Classification

A secure GKE protocol must stand against *passive* and *active adversaries* [44], [45], [46]. A passive adversary can only eavesdrop on the communication channel, while an active adversary has full control over the network (he can drop, modify or insert messages). A particular example of an active attack is *man-in-the-middle*, where the adversary interposes between the communication parties. It is immediate that an active attack is more powerful than a passive attack and therefore they should be considered while analyzing the security of a group key protocol. GKE protocols should also deal with *malicious* (or *dishonest*) users [38]: legitimate participants that do not behave accordingly to the protocol specifications. Subsection 5.2.2.2 reveals an attack mounted by an active malicious insider.

Regarding the appartenance to the group, attackers split into: *outsiders* and *insiders* [44], [46]. An outsider is a party that had not registered as a group member (he does not posses a valid long-lived key within the group) and hence never takes part to the key establishment as a legitimate participant. An outsider attack aims to reveal the established group key and therefore to break the security of the protocol, usually by impersonating authorized users. An insider is a valid group member, who has registered within the group at a given moment and therefore has the advantage to posses a long-lived key. He is qualified to compute session keys he is authorized for, but this should provide him no advantage in revealing other keys (of sessions he is unqualified for) or damage the protocol in any other way: find the long-term keys of other users, ruin key consistency or get control over the key value. Of course, insiders are more powerful than outsiders, because they have access to additional information.

[1]The reader may refer to Section 6.1 for more details.

Subsections 5.2.2.2, 5.2.3.2, 5.2.4.2 and 5.2.4.4 illustrate several insider attacks on different GKT protocols.

Impersonation attacks [19], [22] try to make messages originating from the adversary indistinguishable from the ones originating from legitimate users (the adversary pretends to be a qualified group member). This may result in computing a different key than the genuine one or in establishing a common key with an attacker instead of an authorized user. A successful impersonation attack can for example break entity authentication, unknown key share resilience or key compromise impersonation resilience. A special kind of impersonation attack is the *replay attack* that consists in injecting messages from previous executions of the protocol. This can turn into a *key replication attack*, where the same (corrupted) key is used for multiple runs of the protocol. It is immediate that a key replication attack violates key freshness. Subsections 5.2.1.2 and 5.2.4.2 present two replay attacks against recent GKT protocols.

Known key attacks [3] aim to disclose a session key when the adversary knows at least one key from a previous run of the protocol. All insiders satisfy the assumption by default, as they may legitimate initiate or take part to protocol executions. Subsection 5.2.3.3 exemplifies a known key attack.

Attacks can be classified based on the information the adversary has access to: the long-lived key of the registered users or the ephemeral secrets used during protocol execution. *Opening attacks* allow the attacker to learn the ephemeral secrets without revealing the long-lived secrets, while *weak corruption attacks* allow the attacker to learn the long-lived secrets without revealing ephemeral secrets [17]. *Strong corruption attacks* [15], [17] combine these two attacks and give the adversary tremendous power: he can corrupt a user and obtain both his long-lived secret and his ephemeral secret values.

DoS (Denial of Service) attacks [17], [71] lead to the futility of GKE protocols: they prevent legitimate users to establish common secret keys that they would have later use for application purposes. A DoS attack may inhibit users to compute any key at all or may force users to end up with different keys. Although the attack is discovered at the latest during the execution of the application (the group members realize that they cannot properly communicate between themselves) it represents an important aspect of network security. Subsection 5.2.2.2 describes a DoS attack mounted from inside.

5.2 GKE Protocols

The current section presents some recent GKE constructions that rely on incomplete and informal security arguments. Since they lack a formal security proof, the vulnerabilities may arise natural. We mention some attacks and countermeasures for each protocol. The list is not exhaustive and can be enriched; we consider the expansion of attacks and countermeasures chains a subject for future work.

We affirm that the improvements stand against the corresponding attacks, but do not claim that they represent secure versions of the protocols (as they also skip formal security proofs). We even clearly emphasize their weaknesses in some cases. The main goal of the current chapter is not to define secure GKE protocols, but to reveal vulnerabilities and emphasize the necessity of formal security proofs. Table 5.1 summarizes the work we will

Table 5.1: Attacks and countermeasures applied to Informal GKE Protocols

Original Protocol	Revealed Attacks & Countermeasures
Harn and Lin (Jun 2010) [30]	Nam et al. (Dec 2011) [50]
	Yuan et al. (Sep 2013) [84], Olimid [59]
Hsu et al. (Jan 2012) [32]	Olimid [61]
Sun et al. (Mar 2012) [73]	Olimid (Mar 2013) [60], Kim et al. (May 2013) [40]
Yuan et al. (Jan 2013) [83]	Olimid (Jul 2013) [56], [57]

refer to for the next sections.

For the rest of the chapter, U_1 denotes the initiator (without loss of generality, after a possible reordering) and U_a denotes the attacker. We do not remind the rest of the notations, since they are all explained in the List of Symbols and Notations. We emphasize that many operations are performed within cyclic groups but not explicit specify that[2] since it is clear from the context.

5.2.1 Harn and Lin (2010)

5.2.1.1 Protocol Description

Harn and Lin introduced in 2010 a GKT protocol based on Shamir's SSS [30]. Fig.5.1 describes the protocol in detail.

We mention the main idea: each user generates a uniformly random value R_i and sends it in clear, along with an authentication string, to the KGC (Round 3). The KGC selects a fresh session key k and generates a polynomial $f(x)$ by interpolating $t+1$ points: $(0, k)$ and $(x_i, y_i \oplus R_i)$, where (x_i, y_i) is the long-term secret shared with the user U_i, $i = 1, \ldots, t$ (we highlight that k represents the secret in a Shamir's SSS). Then, he generates other t points $P_1, \ldots, P_t$ and makes them public (Round 4). Since Shamir's scheme is perfect, no passive adversary can reveal k by only eavesdropping on these public values. On the other hand, each user U_i posses an extra point - due to his long-term secret shared with the KGC - and becomes able to determine the group secret key k by polynomial interpolation (Key Computation Phase).

In the original work, Harn and Lin suggest to add two extra rounds after the Key Computation Phase to obtain key confirmation: (1) each user U_i sends to the KGC a key confirmation message $h(x_i, y_i, k)$; (2) after the KGC receives the key confirmation messages from all valid group members that participate to the session, he sends each of them a corresponding key confirmation message $h(x_i, y_i, k, U_1, \ldots, U_t)$, $i = 1, \ldots, t$. We skip this enhancement for the rest of the subsection.

The protocol gives an elegant solution to build GKT protocols based on SSS, it is simple to expose and understand. However, the number of required rounds is high: it requires four rounds (without key confirmation), which considerable diminishes its applicability.

[2] By marking $\pmod p$ where p is the group order.

Initialization. The KGC selects 2 large safe primes p and q (i.e. $p' = \frac{p-1}{2}$ and $q' = \frac{q-1}{2}$ are also primes) and computes $n = pq$;

Users Registration. Each user $U_i, i = 1, \ldots, m$, shares a long-term secret $(x_i, y_i) \in \mathbb{Z}_n^* \times \mathbb{Z}_n^*$ with the KGC;

Round 1. User U_1:
1.1. sends a key generation request:
$$U_1 \rightarrow KGC : \{U_1, \ldots, U_t\}$$

Round 2. The KGC:
2.1. broadcasts:
$$KGC \rightarrow^* : \{U_1, \ldots, U_t\}$$

Round 3. Each user $U_i, i = 1, \ldots, t$:
3.1. chooses $R_i \leftarrow^R \mathbb{Z}_n^*$;
3.2. computes $Auth_i = h(x_i, y_i, R_i)$;
3.3. sends:
$$U_i \rightarrow KGC : (R_i, Auth_i)$$

Round 4. The KGC:
4.1. checks if $Auth_i = h(x_i, y_i, R_i)$, $i = 1, \ldots, t$;
If at least one equality does not hold, he quits;
4.2. selects a group key $k \leftarrow^R \mathbb{Z}_n^*$;
4.3. generates the polynomial $f(x)$ of degree t that passes through the $t+1$ points $(0, k)$, $(x_1, y_1 \oplus R_1)$, $\ldots$, $(x_t, y_t \oplus R_t)$;
4.4. computes t additional points $P_1, \ldots, P_t$ on $f(x)$;
4.5. computes the authentication message $Auth = h(k, U_1, \ldots, U_t, R_1, \ldots, R_t, P_1, \ldots, P_t)$;
4.6. broadcasts:
$$KGC \rightarrow^* : (P_1, \ldots, P_t, R_1, \ldots, R_t, Auth)$$

Key Computation. Each user U_i, $i = 1, \ldots, t$:
5.1. computes the group key $k = f(0)$ by interpolating the points $P_1, \ldots, P_t$ and $(x_i, y_i \oplus R_i)$;
5.2. checks if $Auth = h(k, U_1, \ldots, U_t, R_1, \ldots, R_t, P_1, \ldots, P_t)$;
If the equality does not hold, he quits.

Figure 5.1: Original Version of Harn and Lin's Group Key Transfer Protocol [30]

Step 1. U_a eavesdrops on a protocol session U_i participates to and gets $(R_i, Auth_i)$;
Step 2. U_a initiates (s_j), $j = 1, 2$ two legitimate sessions of the protocol with U_i and uses the same value $(R_a, Auth_a)$;
Step 3. U_a impersonate U_i in both sessions (s_1) and (s_2) by sending in Round 3 the message $(R_i, Auth_i)$ he had eavesdropped in Step 1;
Step 4. U_a is an authorized user for both sessions, so he recovers the polynomials:

$$f(x)_{(s_j)} = a_{(s_j)}x^2 + b_{(s_j)}x + c_{(s_j)}, j = 1, 2$$

Step 6. Since $(x_a, y_a \oplus R_a)$ and $(x_i, y_i \oplus R_i)$ are valid points on $f(x)_{(s_j)}$, $j = 1, 2$, U_a knows that $f(x_a)_{(s_1)} = f(x_a)_{(s_2)} = y_a \oplus R_a$ and $f(x_i)_{(s_1)} = f(x_i)_{(s_2)} = y_i \oplus R_i$; therefore both x_a and x_i are roots of:

$$(a_{(s_1)} - a_{(s_2)})x^2 + (b_{(s_1)} - b_{(s_2)})x + (c_{(s_1)} - c_{(s_2)}) = 0$$

Step 7. U_a reveals the long-term secret of U_i as:

$$x_i = x_j^{-1}(a_{(s_1)} - a_{(s_2)})^{-1}(c_{(s_1)} - c_{(s_2)}).$$

$$y_i = \begin{cases} f(x_i)_{(s_j)} \oplus R_i & if\, y_i \oplus R_i < n \\ (f(x_i)_{(s_j)} + n) \oplus R_i & otherwise \end{cases}$$

Figure 5.2: Replay Attack against the Original Version of Harn and Lin's Protocol [50]

5.2.1.2 Attack I

Nam et al. proposed a replay attack against the previously described protocol [50]. Fig.5.2 explains the attack in detail.

Let U_a to be an insider whose goal is to reveal the long-term password (x_i, y_i) of another user U_i, $i = 1, \ldots, m$, $i \neq a$. If so, he gains the same ability as U_i: he eavesdrops on the value R_i in step 3.3 of the protocol and becomes able to compute $(x_i, y_i \oplus R_i)$, a valid point on $f(x)$. Since $P_1, \ldots, P_t$ are public, he now owns $t + 1$ distinct points and hence he is capable to reconstruct the secret k by interpolation. As the adversary can obtain the session key of all sessions U_i is authorized for (even if U_a is unauthorized for), the confidentiality of the protocol is ruined.

5.2.1.3 Improvement I

The previous attack is possible because the KGC cannot detect replay messages. Nam et al. propose a countermeasure [50], described in detail in Fig.5.3.

We highlight the main idea: for each session, the KGC selects a uniformly random value R_0, which he then broadcasts to the group members (Round 2). Afterwards, each user uses the received value, along with the identities of the participants to the session, to compute the authentication string $Auth_i$ (Round 3). Since the value R_0 differs for distinct sessions, an eavesdropped message $(R_i, Auth_i)$ in one session becomes invalid for other sessions - the equality in step 4.1 fails and hence the KGC quits.

Initialization. The KGC selects 2 large safe primes p and q (i.e. $p' = \frac{p-1}{2}$ and $q' = \frac{q-1}{2}$ are also primes) and computes $n = pq$;

Users Registration. Each user $U_i, i = 1, \ldots, m$, shares a long-term secret $(x_i, y_i) \in \mathbb{Z}_n^* \times \mathbb{Z}_n^*$ with the KGC;

Round 1. User U_1:

1.1. sends a key generation request:

$$U_1 \rightarrow KGC : \{U_1, \ldots, U_t\}$$

Round 2. The KGC:

2.1. selects $R_0 \leftarrow^R \mathbb{Z}_n^*$;

2.2. broadcasts:

$$KGC \rightarrow^* : (\{U_1, \ldots, U_t\}, R_0)$$

Round 3. Each user $U_i, i = 1, \ldots, t$:

3.1. chooses $R_i \leftarrow^R \mathbb{Z}_n^*$;

3.2. computes $Auth_i = h(x_i, y_i, R_i, R_0, U_1, \ldots, U_t)$;

3.3. sends:

$$U_i \rightarrow KGC : (R_i, Auth_i)$$

Round 4. The KGC:

4.1. checks if $Auth_i = h(x_i, y_i, R_i, R_0, U_1, \ldots, U_t)$, $i = 1, \ldots, t$;
If at least one equality does not hold, he quits;

4.2. selects a group key $k \leftarrow^R \mathbb{Z}_n^*$;

4.3. generates the polynomial $f(x)$ of degree t that passes through the $t+1$ points $(0, k)$, $(x_1, y_1 \oplus R_1)$, $\ldots$, $(x_t, y_t \oplus R_t)$;

4.4. computes t additional points $P_1, \ldots, P_t$ on $f(x)$;

4.5. computes the authentication message $Auth = h(k, U_1, \ldots, U_t, R_1, \ldots, R_t, P_1, \ldots, P_t)$;

4.6. broadcasts:

$$KGC \rightarrow^* : (P_1, \ldots, P_t, R_1, \ldots, R_t, Auth)$$

Key Computation. Each user U_i, $i = 1, \ldots, t$:

5.1. computes the group key $k = f(0)$ by interpolating the points $P_1, \ldots, P_t$ and $(x_i, y_i \oplus R_i)$;

5.2. checks if $Auth = h(k, U_1, \ldots, U_t, R_1, \ldots, R_t, P_1, \ldots, P_t)$;
If the equality does not hold, he quits.

Figure 5.3: First Improved Version of Harn and Lin's Protocol [50]

Step 1. U_a intercepts the message $\{U_1, \ldots, U_t\}$ sent in Round 1 and prevents it from reaching the KGC;
Step 2. U_a replaces U_i by U_a in the message from Step 1 and sends:
$U_a \rightarrow KGC : \{U_1, \ldots, U_{i-1}, U_a, U_{i+1}, \ldots, U_t\}$
Step 3. U_a intercepts the broadcast message $\{U_1, \ldots, U_{i-1}, U_a, U_{i+1}, \ldots, U_t\}$ sent in Round 2 and prevents it from reaching the group members;
Step 4. U_a broadcasts the original list $\{U_1, \ldots, U_t\}$ to all the group members (as originating from the KGC):
$U_a(KGC) \rightarrow^*: \{U_1, \ldots, U_t\}$
Step 5. U_a intercepts the message R_i sent in Round 3 and prevents it from reaching the KGC;
Step 6. U_a allows the messages R_j, $j = 1, \ldots, t$, $j \neq i$ to reach the KGC, chooses $R_a \leftarrow^R \mathbb{Z}_n^*$ and sends it to the KGC (instead of R_i):
$U_a \rightarrow KGC : R_a$
Step 7. U_a intercepts the broadcast message $\{R_1, \ldots, R_{i-1}, R_a, R_{i+1}, \ldots, R_t, P_1, \ldots, P_t, Auth\}$ sent in Round 4 and prevents it from reaching the group members;
Step 8. U_a computes $k = f(0)$ by interpolating the public points $P_1, \ldots, P_t$ and $(x_a, y_a \oplus R_a)$, forges $Auth' = h(k, U_1, \ldots, U_t, R_1, \ldots, R_t, P_1, \ldots, P_t)$ and sends $(R_1, \ldots, R_t, P_1, \ldots, P_t, Auth')$ to all the group members except U_i (as originating from the KGC);
Step 9. For all $j = 1, \ldots, t$, $j \neq i$, U_j, recovers the group key $k = f(0)$ by interpolating the public points $P_1, \ldots, P_t$ and $(x_j, y_j \oplus R_j)$, checks that $Auth' = h(k, U_1, \ldots, U_t, R_1, \ldots, R_t, P_1, \ldots, P_t)$ holds and therefore accepts k as the correct group key.

Figure 5.4: Man-in-the-Middle Attack against the Original Version of Harn and Lin's Protocol [84]

5.2.1.4 Attack II

Yuan et al. consider the original proposal of Harn and Lin that skips R_i's authentication [84]. The only differences from the protocol described in Fig.5.1 appear in Rounds 3 and 4: the participants do not compute $Auth_i$ (step 3.2 vanishes) and hence they only send R_i to the KGC (step 3.3), respectively the KGC performs no verification (step 4.1 vanishes). We omit the full description to avoid repetition. Yuan et al. motivate their choice by the vulnerability of the authenticated version to a guessing attack[3]: an attacker can eavesdrop on $Auth_i$ and R_i and guess offline the pair (x_i, y_i) such that $Auth_i = h(x_i, y_i, R_i)$.

This version of Harn and Lin's protocol is vulnerable to a man-in-the-middle attack, revealed in Fig.5.4 [84].

The attacker's goal is to impersonate a user U_i in a session U_a is unauthorized for and share a common key with the rest of the participants to the session, while the victim U_i is expelled from the group. We highlight that the users believe to share a key within the group $\{U_1, \ldots, U_t\}$, while the KGC thinks he generates a key for $\{U_1, \ldots, U_{i-1}, U_a, U_{i+1}, \ldots, U_t\}$.

[3]Please refer to Subsection 5.2.3.5 for more details.

The attack in Fig.5.4 is self-contained, so we skip other comments. We only mention that a simpler version is possible and revise the differences as follows: (1) Step 4 - U_a broadcast the original list to all group members, except the victim U_i; (2) Step 5 - is no longer required; (3) Step 7 - U_a allows the message to reach all group members, except U_i; (4) Step 8 - U_a only computes the session group key k by interpolation.

The modified version of the attack presents two main advantages. First, it is easier to mount, as U_a must intercept and prevent fewer messages from reaching their destination. Second, it prevents detection: in the original version of the attack, U_i may suspect a malicious action since he is invited to take part to a protocol session that never finalizes; in the modified version of the attack, U_i is not even aware that the session is taking place.

5.2.1.5 Improvement II

The previous attack is possible because the first two rounds of the protocol lack authentication; hence, any insider or outsider may impersonate a valid user or the KGC and send a list of participants in their behalf.

Yuan et al. propose a countermeasure that they claim to stand against the man-in-the-middle attack [84]. Fig.5.5 describes it in detail.

We highlight the modifications from the original construction. The KGC generates a RSA public-private key pair in the Initialization Phase. Let n be the RSA modulus, $\Phi(n) = (p-1)(q-1)$ the Euler's totient function, e the signature verification exponent and d the corresponding signature generation exponent. We skip more details of RSA here, but invite the reader to address the original paper [64]. The KGC makes his public RSA key (e, n) available to users during the Registration Phase, as they will later need it for signature verification purposes. In Round 2, the KGC signs $h(U_1, \ldots, U_t)$ and broadcast the signature v along with the list of participants. This aims to authenticate the message origin such it becomes impossible for an attacker to send a different message on his behalf. In Round 3, each user that identifies himself in the list verifies the origin of the message by using the public key of the KGC.

We remark that the proposed solution does not eliminate a replay attack: an attacker U_a may eavesdrop on a message $(\{U_1, \ldots, U_t\}, v)$ in one session of the protocol and reuse it to impersonate the KGC in another session that intends to establish a common key for the same set of users. This is possible because the signature only relies on the list of participants to a given session and the signing exponent remains unchanged for a long period of time (it is generated during the Initialization Phase and maintained for multiple sessions).

5.2.1.6 Attack III

Unlike Yuan et al.'s claim that in their improved version no intermediate entity can play the role of a participant to a session without being detected [83], their proposal remains vulnerable to a man-in-the-middle attack [59]. Fig.5.6 presents the attack in detail.

U_a maintains the same abilities (he is an active insider with full control over the communication channel that shares a long-term key (x_a, y_a) with the KGC) and has the same goal (to impersonate a victim U_i and share a common key within a set of participants $\{U_1, \ldots, U_{i-1}, U_a, U_{i+1}, \ldots, U_t\}$, while the other users believe to share a common key within $\{U_1, \ldots, U_t\}$).

Initialization. The KGC selects 2 large safe primes p and q (i.e. $p' = \frac{p-1}{2}$ and $q' = \frac{q-1}{2}$ are also primes) and computes $n = pq$;

Then, it selects $e \in \mathbb{Z}_n^*$ s.t. $(e, \Phi(n)) = 1$ and computes $d \in \mathbb{Z}_n^*$ s.t. $ed = 1 \pmod{\Phi(n)}$;

Users Registration. Each user $U_i, i = 1, \ldots, m$, shares a long-term secret $(x_i, y_i) \in \mathbb{Z}_n^* \times \mathbb{Z}_n^*$ and the public key (e, n) with the KGC;

Round 1. User U_1:

1.1. sends a key generation request:

$$U_1 \rightarrow KGC : \{U_1, \ldots, U_t\}$$

Round 2. The KGC:

2.1. computes $v = h(U_1, \ldots, U_t)^d \pmod{n}$;

2.2. broadcasts:

$$KGC \rightarrow^*: (\{U_1, \ldots, U_t\}, v)$$

Round 3. Each user $U_i, i = 1, \ldots, t$:

3.1. checks if $h(U_1, \ldots, U_t) = v^e \pmod{n}$;

If the equality does not hold, he quits.

3.2. chooses $R_i \leftarrow^R \mathbb{Z}_n^*$;

3.3. sends:

$$U_i \rightarrow KGC : R_i$$

Round 4. The KGC:

4.1. selects a group key $k \leftarrow^R \mathbb{Z}_n^*$;

4.2. generates the polynomial $f(x)$ of degree t that passes through the $t+1$ points $(0, k)$, $(x_1, y_1 \oplus R_1), \ldots, (x_t, y_t \oplus R_t)$;

4.3. computes t additional points $P_1, \ldots, P_t$ on $f(x)$;

4.4. computes the authentication message $Auth = h(k, U_1, \ldots, U_t, R_1, \ldots, R_t, P_1, \ldots, P_t)$;

4.5. broadcasts:

$$KGC \rightarrow^*: (R_1, \ldots, R_t, P_1, \ldots, P_t, Auth)$$

Key Computation. Each user U_i, $i = 1, \ldots, t$:

5.1. computes the group key $k = f(0)$ by interpolating the points $P_1, \ldots, P_t$ and $(x_i, y_i \oplus R_i)$;

5.2. checks if $Auth = h(k, U_1, \ldots, U_t, R_1, \ldots, R_t, P_1, \ldots, P_t)$;

If the equality does not hold, he quits.

Figure 5.5: Second Improved Version of Harn and Lin's Protocol [84]

Step 1. U_a does not interfere in Round 1 of a genuine session, but he intercepts the response $(\{U_1, \ldots, U_t\}, v)$ sent in Round 2 and prevents it from reaching the group members.

Step 2. U_a impersonate all participants to the session by sending R'_j on behalf of U_j, $j = 1, \ldots, t$:

$$U_a(U_j) \rightarrow KGC : R'_j$$

Step 3. U_a intercepts the message $(R'_1, \ldots, R'_t, P'_1, \ldots, P'_t, Auth')$ sent in Round 4 and prevents it from reaching the group members.

Step 4. U_a initiates a new protocol session:

$$U_a \rightarrow KGC : \{U_1, \ldots, U_{i-1}, U_a, U_{i+1}, \ldots, U_t\}$$

Step 5. U_a intercepts the broadcast message $(\{U_1, \ldots, U_{i-1}, U_a, U_{i+1}, \ldots, U_t\}, v')$ sent in Round 2 of the new session and prevents it from reaching the group members;

Step 6. U_a forwards the response in Step 1 to all group members except U_i;
For all $j = 1, \ldots, t$, $j \neq i$, U_j checks that $h(U_1, \ldots, U_t) = v^e \pmod n$ holds and therefore cannot detect the attack;

Step 7. U_a allows the messages R_j, $j = 1, \ldots, t$, $j \neq i$ to reach the KGC, chooses $R_a \leftarrow^R \mathbb{Z}_n^*$ and sends it to the KGC :

$$U_a \rightarrow KGC : R_a$$

Step 8. U_a intercepts the message $(R_1, \ldots, R_{i-1}, R_a, R_{i+1}, \ldots, R_t, P_1, \ldots, P_t, Auth)$ in Round 4, computes the group key $k = f(0)$ by interpolating the public points $P_1, \ldots, P_t$ and $(x_a, y_a \oplus R_a)$ then forwards the message to the group members (as originating from the KGC).

Step 9. For all $j = 1, \ldots, t$, $j \neq i$, U_j, recovers the group key $k = f(0)$ by interpolating the public points $P_1, \ldots, P_t$ and $(x_j, y_j \oplus R_j)$, checks that $Auth = h(k, U_1, \ldots, U_t, R_1, \ldots, R_{i-1}, R_a, R_{i+1}, \ldots, R_t, P_1, \ldots, P_t)$ holds and therefore accepts k as the correct group key.

Figure 5.6: Man-in-the-Middle Attack against the Second Improved Version of Harn and Lin's Protocol [59]

The attack in Fig.5.6 is self-contained. We only emphasize the main idea: U_a makes the KGC believe that the genuine session has successfully finished and then initiate a new protocol session that the participants consider to be the genuine one.

In addition to the first attack, U_a must impersonate all users $U_1, \ldots, U_t$ (in Step 2), but this is trivial to achieve because Round 3 skips authentication.

We remark that neither the victim U_i nor the KGC can suspect a malicious behavior: U_i is not aware that a protocol session is taking place and the KGC considers that both sessions have successfully finished.

5.2.2 Hsu et al. (2012)

5.2.2.1 Protocol Description

Hsu et al. introduced in 2012 a GKT protocol based on a perfect linear SSS (LSSS) - the perspective of secret m-sharing described in Subsection 2.2 [32]. Fig 5.7 describes the protocol in detail.

We remark a main difference from the rest of the protocols described in this section: the construction does not inquire an external KGC and therefore no previous registration to the KGC is required. The initiator U_1 performs the role of the KGC and establishes a common secret key with each other participant at runtime, which corresponds to the long-term key, but has the advantage to be fresh for each session. However, each participant U_i, $i = 1, \ldots, m$, must own a public-secret key pair (pk_i, sk_i) authenticated by a trusted authority with a certificate.

The protocol assumes that the DLP is hard (i.e. given $pk_i = g^{sk_i}$ it is computationally infeasible to compute sk_i, $i = 1, \ldots, m$) and that CDH holds (i.e. given pk_i and pk_j it is computationally infeasible to compute $g^{sk_i sk_j}$, $i, j = 1, \ldots, m$, $i \neq j$)[4] in G.

5.2.2.2 Attack I

Hsu et al.'s protocol is vulnerable to an active attack mounted from inside [61]. Fig.5.8 reveals the attack.

The adversary's goal is to break key consistency and make different authorized users accept distinct keys: any qualified user $U_i \in \mathcal{U} \setminus \{U_a, U_1\}$ (except the initiator) computes a key k_i selected by the adversary. The attack assumes that U_a is able to intercept exchanged messages and to prevent them from reaching their destination, as well as injecting messages of his choice.

We skip the details as Fig.5.8 is self-contained. We only emphasize that the user U_i is unable to discover the attack during the protocol execution: $U_i \in \mathcal{U} \setminus \{U_a, U_1\}$ recovers the correct value T_i, but then computes the group key as $k_i = T_i + K'_i$. The verification holds in the Key Computation Phase, because the attacker was able to compute $Auth_i$ based on his own choice of k_i. Therefore, U_i considers k_i to be the correct group key.

We remark that U_a may mount the same attack simultaneously against multiple users in order to induce each one a different key. More, the adversary's identity remains hidden, which allows him to mount the attack several times, without being detected and excluded from the group.

5.2.2.3 Improvement I

The previous attack is caused by an authentication flaw: the group key k is not properly authenticated as originating from the initiator U_1. This allows the adversary to impersonate U_1 and send a modified but valid authentication string that helps him to achieve his goal.

A trivial way to reveal the attack is immediate: a Key Confirmation Phase assures that all users posses the correct key [12]. Fig.5.9 describes this enhancement: each user signs the group key he obtained and broadcast it to the other members. In order to maintain the

[4] Please refer to Appendix A.2, Definitions A.11 and A.13 for more details.

Initialization. Let G be a multiplicative cyclic group of order p, with g as generator, where p is a large safe prime (i.e. $p' = \frac{p-1}{2}$ is also prime);

Users Registration. Each user U_i, $i = 1, \ldots, m$ owns a public-private key pair (pk_i, sk_i) s.t. $pk_i = g^{sk_i}$ in G.

Round 1. User U_1:

1.1. chooses $r_1 \leftarrow^R \mathbb{Z}_p^*$;

1.2. sends a key generation request:

$$U_1 \rightarrow^*: (\{U_1, \ldots, U_t\}, r_1, pk_1)$$

Round 2. Each user $U_i, i = 2, \ldots, t$:

2.1. chooses $r_i \leftarrow^R \mathbb{Z}_p^*$;

2.2. computes $S_i = pk_1^{sk_i r_i r_1}$ a shared secret with U_1 and $Auth_i = h(S_i, r_1)$;

2.3. broadcasts:

$$U_i \rightarrow^*: (r_i, pk_i, Auth_i)$$

Round 3. User U_1:

3.1. computes $S_i = pk_i^{sk_1 r_i r_1}$, $i = 2, \ldots, t$;

3.2. checks if $Auth_i = h(S_i, r_1)$, $i = 2, \ldots, t$;
If at least one equality does not hold, he quits;

3.3. chooses the group key $k \leftarrow^R \mathbb{Z}_p^*$, splits each secret S_i into two parts $S_i = x_i || y_i$ and computes $t-1$ values $K_i = k - T_i$, where $T_i = (y_i v(x_i), r)$ is the inner product of the vectors $y_i v(x_i) = y_i \sum_{j=1}^{t} x_i^{j-1} e_j$ ($e_j = (0, \ldots, 1, \ldots, 0)$ with 1 on position j), $i = 2, \ldots, t$ and $r = (r_1, \ldots, r_t)$;

3.4. computes $Auth = h(k, U_1, \ldots, U_t, r_1, \ldots, r_t, K_2, \ldots, K_t)$;

3.5. broadcasts:

$$U_1 \rightarrow^*: (Auth, K_2, \ldots, K_t)$$

Key Computation. Each user U_i, $i = 2, \ldots, t$:

4.1. computes the inner product $T_i = (y_i v(x_i), r)$, recovers the group key $k = T_i + K_i$;

4.2. checks if $Auth = h(k, U_1, \ldots, U_t, r_1, \ldots, r_t, K_2, \ldots, K_t)$;
If the equality does not hold, he quits.

Figure 5.7: Original Version of Hsu et al.'s Group Key Transfer Protocol [32]

Step 1. U_a is qualified to participate to a protocol session and hence he finds the key k:
$k = T_a + K_a$
Step 2. U_a intercepts the broadcast message sent in Round 3 of the protocol and prevents it from reaching U_i;
Step 3. U_a eavesdrops on K_i and therefore computes:
$T_i = k - K_i$
Step 4. U_a chooses a key k_i, computes the corresponding value $K'_i = k_i - T_i$ and authenticates his selection as:
$Auth_i = h(k_i, U_1, \ldots, U_t, r_1, \ldots, r_t, K_2, \ldots, K'_i, \ldots K_t)$
Step 5. U_a sends:
$U_a \rightarrow U_i : (Auth_i, K_2, \ldots, K'_i, \ldots, K_t)$
Step 6. U_i recovers the correct value T_i, computes the group key $k_i = T_i + K'_i$ and checks that
$Auth = h(k_i, U_1, \ldots, U_t, r_1, \ldots, r_t, K_2, \ldots, K'_i, \ldots, , K_t)$
holds. U_i accepts k_i as the correct group key.

Figure 5.8: Insider Attack against the Original Version of Hsu et al.'s Protocol [61]

confidentiality of its value, the key is first hashed, along with some of the public values used during the protocol execution. The drawback of this solution is clear - the computational and transmission costs significantly increase: in addition to the original protocol, each user generates one signature and verifies $t - 1$ others, respectively an additional round of communication is necessary and t more broadcast messages circulate over the network. More, this approach does not eliminate the attack, but only reveals it during the key generation process, in advance to the execution of the application. We remark that such attacks are usually disclosed at runtime, since the users realize that they cannot properly achieve their tasks (for example they are not able to communicate between themselves).

5.2.2.4 Improvement II

We propose a different improvement: the initiator signs the value $Auth$ so that the attacker cannot forge it. Fig.5.10 describes the countermeasure in detail.

Unlike the first solution, the second discards the authentication flaw (U_a cannot impersonate U_1 anymore) and decreases the overall cost (in addition to the original protocol, the initiator generates one signature and the rest of participants verify it).

5.2.3 Sun et al. (2012)

5.2.3.1 Protocol Description

Sun et al. introduced in 2012 a GKT protocol, which is based on the untraditional perspective on secret m-sharing introduced in Subsection 2.2 [73]. Fig.5.11 describes the protocol in detail.

We stress a small modification: in the original version of the protocol, U_i sends the value r_i only to the KGC (Round 3); instead, we consider a broadcast message in order to avoid

Initialization. Let G be a multiplicative cyclic group of order p, with g as generator, where p is a large safe prime (i.e. $p' = \frac{p-1}{2}$ is also prime);

Users Registration. Each user U_i, $i = 1, \ldots, m$ owns a public-private key pair (pk_i, sk_i) s.t. $pk_i = g^{sk_i}$ in G.

Round 1. User U_1:
1.1. chooses $r_1 \leftarrow^R \mathbb{Z}_p^*$;
1.2. sends a key generation request:
$U_1 \rightarrow^*: (\{U_1, \ldots, U_t\}, r_1, pk_1)$

Round 2. Each user $U_i, i = 2, \ldots, t$:
2.1. chooses $r_i \leftarrow^R \mathbb{Z}_p^*$;
2.2. computes $S_i = pk_1^{sk_i r_i r_1}$ a shared secret with U_1 and $Auth_i = h(S_i, r_1)$;
2.3. broadcasts:
$U_i \rightarrow^*: (r_i, pk_i, Auth_i)$

Round 3. User U_1:
3.1. computes $S_i = pk_i^{sk_1 r_i r_1}$, $i = 2, \ldots, t$;
3.2. checks if $Auth_i = h(S_i, r_1)$, $i = 2, \ldots, t$;
If at least one equality does not hold, he quits;
3.3. chooses the group key $k \leftarrow^R \mathbb{Z}_p^*$, splits each secret S_i into two parts $S_i = x_i || y_i$ and computes $t-1$ values $K_i = k - T_i$, where $T_i = (y_i v(x_i), r)$ is the inner product of the vectors $y_i v(x_i) = y_i \sum_{j=1}^{t} x_i^{j-1} e_j$ ($e_j = (0, \ldots, 1, \ldots, 0)$ with 1 on position j), $i = 2, \ldots, t$ and $r = (r_1, \ldots, r_t)$;
3.4. computes $Auth = h(k, U_1, \ldots, U_t, r_1, \ldots, r_t, K_2, \ldots, K_t)$;
3.5. broadcasts:
$U_1 \rightarrow^*: (Auth, K_2, \ldots, K_t)$

Key Computation. Each user U_i, $i = 2, \ldots, t$:
4.1. computes the inner product $T_i = (y_i v(x_i), r)$, recovers the group key $k = T_i + K_i$;
4.2. checks if $Auth = h(k, U_1, \ldots, U_t, r_1, \ldots, r_t, K_2, \ldots, K_t)$;
If the equality does not hold, he quits.

Key Confirmation. Each user U_i, $i = 1, \ldots, t$:
5.1. computes $V_i = \Sigma.\mathsf{Sign}_{U_i}(h(k, U_1, \ldots, U_t, r_1, \ldots, r_t, K_2, \ldots, K_t))$;
5.2. broadcasts:
$U_i \rightarrow^*: V_i$
5.3. checks if $\Sigma.\mathsf{Verify}_{U_j}(h(k, U_1, \ldots, U_t, r_1, \ldots, r_t, K_2, \ldots, K_t), V_j) = 1$, $j = 1, \ldots, t$, $j \neq i$.
If at least one equality does not hold, he quits.

Figure 5.9: First Improved Version of Hsu et al.'s Group Key Transfer Protocol [61]

Initialization. Let G be a multiplicative cyclic group of order p, with g as generator, where p is a large safe prime (i.e. $p' = \frac{p-1}{2}$ is also prime);

Users Registration. Each user U_i, $i = 1, \ldots, m$ owns a public-private key pair (pk_i, sk_i) s.t. $pk_i = g^{sk_i}$ in G.

Round 1. User U_1:

1.1. chooses $r_1 \leftarrow^R \mathbb{Z}_p^*$;

1.2. sends a key generation request:

$$U_1 \rightarrow^*: (\{U_1, \ldots, U_t\}, r_1, pk_1)$$

Round 2. Each user $U_i, i = 2, \ldots, t$:

2.1. chooses $r_i \leftarrow^R \mathbb{Z}_p^*$;

2.2. computes $S_i = pk_1^{sk_i r_i r_1}$ a shared secret with U_1 and $Auth_i = h(S_i, r_1)$;

2.3. broadcasts:

$$U_i \rightarrow^*: (r_i, pk_i, Auth_i)$$

Round 3. User U_1:

3.1. computes $S_i = pk_i^{sk_1 r_i r_1}$, $i = 2, \ldots, t$;

3.2. checks if $Auth_i = h(S_i, r_1)$, $i = 2, \ldots, t$;
If at least one equality does not hold, he quits;

3.3. chooses the group key $k \leftarrow^R \mathbb{Z}_p^*$, splits each secret S_i into two parts $S_i = x_i || y_i$ and computes $t-1$ values $K_i = k - T_i$, where $T_i = (y_i v(x_i), r)$ is the inner product of the vectors $y_i v(x_i) = y_i \sum_{j=1}^{t} x_i^{j-1} e_j$ ($e_j = (0, \ldots, 1, \ldots, 0)$ with 1 on position j), $i = 2, \ldots, t$ and $r = (r_1, \ldots, r_t)$;

3.4. computes $Auth = \Sigma.\mathsf{Sign}_{U_1}(h(k, U_1, \ldots, U_t, r_1, \ldots, r_t, K_2, \ldots, K_t))$;

3.5. broadcasts:

$$U_1 \rightarrow^*: (Auth, K_2, \ldots, K_t)$$

Key Computation. Each user U_i, $i = 2, \ldots, t$:

4.1. computes the inner product $T_i = (y_i v(x_i), r)$, recovers the group key $k = T_i + K_i$;

4.2. checks if $\Sigma.\mathsf{Verify}_{U_1}(h(k, U_1, \ldots, U_t, r_1, \ldots, r_t, K_2, \ldots, K_t), Auth) = 1$.
If the equality does not hold, he quits.

Figure 5.10: Second Improved Version of Hsu et al.'s Group Key Transfer Protocol [61]

qualified participants to eavesdrop during protocol execution (the users need all values r_i in the Key Computation Phase).

The protocol assumes that DLP is hard in G[5].

5.2.3.2 Attack I

Sun et al.'s protocol is susceptible to an insider attack [60]. Fig.5.12 exposes the attack.

As U_a is an insider, he may legitimate take part to protocol sessions. Let (s_1) be one of these sessions. Suppose U_a is unauthorized to recover (s_2) session key, $(s_2) \neq (s_1)$. We assume that there exists a participant $U_b \in \mathcal{U}_{(s_1)} \bigcap \mathcal{U}_{(s_2)}$ that is qualified for both (s_1) and (s_2). Then, the attacker U_a can find the key $k_{(s_2)}$. In conclusion, an insider can determine any session key under the assumption that at least one mutual authorized participant for both sessions exists, which is very likely to happen.

The attack also stands if there is no common qualified user for the two sessions, but there exists a third one (s_3) that has a mutual authorized party with each of the former sessions. The extension is straightforward: let $U_{1,3}$, $U_{2,3}$ be the common qualified parties for sessions (s_1) and (s_3), respectively (s_2) and (s_3). U_a computes the key $k_{(s_3)}$ as in the proposed attack due to the common authorized participant $U_{1,3}$. Once he obtains the key $k_{(s_3)}$, he mounts the attack again for sessions (s_3) and (s_2) based on the common party $U_{2,3}$ and gets $k_{(s_2)}$.

The attack extends in chain: the insider U_a reveals a session key $k_{(s_x)}$ if he is able to build a chain of sessions $(s_1) \ldots (s_x)$, where (s_i) and (s_{i+1}) have at least one common qualified member $U_{i,i+1}$, $i = 1, \ldots, x-1$ and U_a is authorized to recover the key $k_{(s_1)}$.

5.2.3.3 Attack II

Sun et al.'s protocol is vulnerable to a known key attack [60]. Fig.5.13 exposes the attack.

We remark the similarity with the previous insider attack. However, in case of the known key attack, the adversary may be an insider or an outsider that somehow manages to obtain a session key. The attack may also be mount in chain, similar to the insider attack. We omit the details in order to avoid repetition.

For both attacks, the adversary computes the session key as the product of two values: g^{s_b} (disclosed only by eavesdropping) and $g^{s'_b}$ (revealed by eavesdropping when a session key is known). We remark that the attacker is unable to determine the long-term secret s'_b if the discrete logarithm assumption holds, but we emphasize this does not imply that the protocol is secure, since the adversary's main objective is to find the session key, which can be disclosed without the knowledge of s'_b.

5.2.3.4 Improvement I

Sun et al.'s GKA protocol fails because the values $g^{s'_i}$, $i = 1, \ldots, m$ are maintained during multiple sessions. We highlight that the underlying SSS, described in Subsection 2.2, suffers from a similar limitation, caused by the usage of the long-term secrets s'_i, $i = 1, \ldots, m$ during multiple sessions: any entity that discloses a secret S determines the values $s'_i = S - s_i$ by eavesdropping on s_i, $i = 1, \ldots, m$ and uses them to reveal other shared secrets.

[5] Please refer to Appendix A.2, Definition A.13 for more details.

Initialization. The KGC selects a multiplicative cyclic group G of prime order p with g as generator;

Users Registration. Each user $U_i, i = 1, \ldots, m$, shares a long-term secret $s'_i \in G$ with the KGC;

Round 1. User U_1:

1.1. sends a key generation request:

$$U_1 \rightarrow KGC : \{U_1, \ldots, U_t\}$$

Round 2. The KGC:

2.1. broadcasts:

$$KGC \rightarrow^*: \{U_1, \ldots, U_t\}$$

Round 3. Each user $U_i, i = 1, \ldots, t$:

3.1. chooses $r_i \leftarrow^R \mathbb{Z}_p^*$;

3.2. broadcasts:

$$U_i \rightarrow^*: r_i$$

Round 4. The KGC:

4.1. chooses $S \leftarrow^R G$;

4.2. invokes the secret m-sharing scheme to split S into 2 shares t times s.t. $S = s_i + s'_i$, $i = 1, \ldots, t$;

4.3. computes $k = g^S$, $M_i = (g^{s_i+r_i}, U_i, h(U_i, g^{s_i+r_i}, s'_i, r_i))$, $i = 1, \ldots, t$ and $Auth = h(k, g^{s_1+r_1}, \ldots, g^{s_t+r_t}, U_1, \ldots, U_t, r_1, \ldots, r_t)$;

4.4. broadcasts:

$$KGC \rightarrow^*: (M_1, \ldots, M_t, Auth)$$

Key Computation. Each user U_i, $i = 1, \ldots, t$:

5.1. checks if $h(U_i, g^{s_i+r_i}, s'_i, r_i)$ equals the corresponding value in M_i;
If the equality does not hold, he quits;

5.2. computes the group key $k = g^{s'_i} g^{s_i+r_i} (g^{r_i})^{-1}$;

5.3. checks if $Auth = h(k, g^{s_1+r_1}, \ldots, g^{s_t+r_t}, U_1, \ldots, U_t, r_1, \ldots, r_t)$;
If the equality does not hold, he quits;

5.4. computes $h_i = h(s'_i, k, U_1, \ldots, U_t, r_1, \ldots, r_t)$

5.4. sends:

$$U_i \rightarrow KGC : h_i$$

Key Confirmation. The KGC:

6.1. checks if $h_i = h(s'_i, k, U_1, \ldots, U_t, r_1, \ldots, r_t)$, $i = 1, \ldots, t$ certifying that all users posses the same key.

Figure 5.11: Original Version of Sun et al.'s Group Key Transfer Protocol [73]

Step 1. U_a is qualified to participate to a session (s_1) and hence he finds the key $k_{(s_1)}$:

$$k_{(s_1)} = \frac{g^{s'_a} \cdot g^{sa_{(s_1)}+ra_{(s_1)}}}{g^{ra_{(s_1)}}}$$

Step 2. Since $r_{i_{(s_1)}}$ and $g^{s_{i(s_1)}+r_{i(s_1)}}$ are send in clear in Rounds 3 and 4 of the protocol, he is able to compute $g^{s'_i}$ for all $U_i \in \mathcal{U}_{(s_1)}$:

$$g^{s'_i} = \frac{k_{(s_1)} \cdot g^{r_{i(s_1)}}}{g^{s_{i(s_1)}+r_{i(s_1)}}}$$

Step 3. U_a eavesdrops on $r_{j_{(s_2)}}$ and $g^{s_{j(s_2)}+r_{j(s_2)}}$ in a session (s_2) s.t. $U_a \notin \mathcal{U}_{(s_2)}$ and determines, for all $U_j \in \mathcal{U}_{(s_2)}$:

$$g^{s_{j(s_2)}} = \frac{g^{s_{j(s_2)}+r_{j(s_2)}}}{g^{r_{j(s_2)}}}$$

Step 4. If $U_b \in \mathcal{U}_{(s_1)} \bigcap \mathcal{U}_{(s_2)}$ exists, then U_a reveals the key $k_{(s_2)}$ of the session (s_2) as:

$$k_{(s_2)} = g^{s'_b} \cdot g^{s_{b(s_2)}} = g^{s'_b + s_{b(s_2)}}.$$

Figure 5.12: Insider Attack against the Original Version of Sun et al.'s Protocol [60]

Step 1. Since $r_{i_{(s_1)}}$ and $g^{s_{i(s_1)}+r_{i(s_1)}}$ are send in clear in Rounds 3 and 4 of the protocol, he is able to compute $g^{s'_i}$ for all $U_i \in \mathcal{U}_{(s_1)}$ under the assumption that he had somehow managed to disclose the session key $k_{(s_1)}$:

$$g^{s'_i} = \frac{k_{(s_1)} \cdot g^{r_{i(s_1)}}}{g^{s_{i(s_1)}+r_{i(s_1)}}}$$

Step 2. U_a eavesdrops on $r_{j_{(s_2)}}$ and $g^{s_{j(s_2)}+r_{j(s_2)}}$ in a session (s_2), s.t. $(s_2) \neq (s_1)$, $U_a \notin \mathcal{U}_{(s_2)}$ and determines, for all $U_j \in \mathcal{U}_{(s_2)}$:

$$g^{s_{j(s_2)}} = \frac{g^{s_{j(s_2)}+r_{j(s_2)}}}{g^{r_{j(s_2)}}}$$

Step 3. If $U_b \in \mathcal{U}_{(s_1)} \bigcap \mathcal{U}_{(s_2)}$ exists, then U_a reveals the key $k_{(s_2)}$ of the session (s_2) as:

$$k_{(s_2)} = g^{s'_b} \cdot g^{s_{b(s_2)}} = g^{s'_b + s_{b(s_2)}}.$$

Figure 5.13: Known Key Attack against the Original Version of Sun et al.'s Protocol [60]

A trivial modification prevents the proposed attacks: KGC replaces the values s'_i at the beginning of each session. This way, even if the attacker determines $g^{s'_i}$ in one session, the value becomes unusable for other sessions. However, this introduces a major drawback: KGC must share a secure channel with each user for any round. If this were the case, KGC could have just sent the secret session key via the secure channel and no other protocol would have been necessary. In conclusion, the GKT protocol would have become useless.

Another similar solution exists: during the User Registration Phase an ordered set of secrets is shared between KGC and each user. For each session, the corresponding secret in the set is used as a share in the underlying SSS. Although this solution has the benefit that it only requires the existence of secure channels at registration, it introduces other disadvantages: each user must store a considerable larger quantity of secret information,

Initialization. The KGC selects a multiplicative cyclic group G of prime order p with g as generator;

Users Registration. Each user $U_i, i = 1, \ldots, m$, shares a long-term secret $s'_i \in G$ with the KGC;

Round 1. User U_1:

1.1. sends a key generation request:

$$U_1 \rightarrow KGC : \{U_1, \ldots, U_t\}$$

Round 2. The KGC:

2.1. broadcasts:

$$KGC \rightarrow^*: \{U_1, \ldots, U_t\}$$

Round 3. Each user $U_i, i = 1, \ldots, t$:

3.1. chooses $r_i \leftarrow^R \mathbb{Z}_p^*$;

3.2. broadcasts:

$$U_i \rightarrow^*: r_i$$

Round 4. The KGC:

4.1. chooses $S \leftarrow^R G$ and $\alpha \leftarrow^R G$;

4.2. invokes the secret m-sharing scheme to split S into 2 shares t times s.t. $S = s_i + s'_i$, $i = 1, \ldots, t$;

4.3. computes $k = \alpha^S$, $M_i = (\alpha^{s_i+r_i}, U_i, h(U_i, \alpha^{s_i+r_i}, s'_i, r_i, \alpha))$, $i = 1, \ldots, t$ and $Auth = h(k, \alpha^{s_1+r_1}, \ldots, \alpha^{s_t+r_t}, U_1, \ldots, U_t, r_1, \ldots, r_t, \alpha)$;

4.4. broadcasts:

$$KGC \rightarrow^*: (M_1, \ldots, M_t, Auth, \alpha)$$

Key Computation. Each user U_i, $i = 1, \ldots, t$:

5.1. checks if $h(U_i, g^{s_i+r_i}, s'_i, r_i, \alpha)$ equals the corresponding value in M_i;
If the equality does not hold, he quits;

5.2. computes the group key $k = \alpha^{s'_i}\alpha^{s_i+r_i}(\alpha^{r_i})^{-1}$;

5.3. checks if $Auth = h(k, \alpha^{s_1+r_1}, \ldots, \alpha^{s_t+r_t}, U_1, \ldots, U_t, r_1, \ldots, r_t, \alpha)$;
If the equality does not hold, he quits;

5.4. computes $h_i = h(s'_i, k, U_1, \ldots, U_t, r_1, \ldots, r_t, \alpha)$

5.4. sends:

$$U_i \rightarrow KGC : h_i$$

Key Confirmation. The KGC:

6.1. checks if $h_i = h(s'_i, k, U_1, \ldots, U_t, r_1, \ldots, r_t, \alpha)$, $i = 1, \ldots, t$ certifying that all users posses the same key.

Figure 5.14: Improved Version of Sun et al.'s Group Key Transfer Protocol [60]

Step 1. U_a eavesdrops on U_i, r_i, $g^{s_i+r_i}$ $i = 1, \ldots t$ and $Auth$ in Rounds 2, 3 and 4 of the protocol;
Step 2. U_a obtains the session key k by launching a guessing attack on $Auth = h(k, g^{s_1+r_1}, \ldots, g^{s_t+r_t}, U_1, \ldots, U_t, r_1, \ldots, r_t)$

Figure 5.15: Guessing Attack against the Original Version of Sun et al.'s Protocol [40]

the protocol can run for at most a number of times equal to the set cardinality, KGC must broadcast the round number so that participants remain synchronized.

We propose next a countermeasure [60] inspired by the work of Pieprzyk and Li [63]. Fig.5.14 describes it is detail. The main idea consist in using for each session a different public value $\alpha \in G$ to compute the session key $k = \alpha^S$.

The countermeasure eliminates both attacks. Under the discrete logarithm assumption and uniformly random selection of $\alpha_{(s_1)}$ and $\alpha_{(s_2)}$, a value ${\alpha_{(s_1)}}^{s'_i}$ from a session (s_1) can no longer be used to compute a session key $k_{(s_2)} = \alpha_{(s_2)}^{s'_i+s_{i_{(s_2)}}}$ with $(s_2) \neq (s_1)$. The values α are authenticated to originate from KGC so that an attacker cannot impersonate the KGC and use a suitable value (for example $\alpha_{(s_2)} = \alpha_{(s_1)}^a$ with a known a).

We remark that the modified version of the protocol maintains all the benefits of the original construction and preserves the computational cost, while the transmission cost increases negligible. However, we admit that it conserves a weakness of the original protocol: it cannot achieve forward secrecy. Any attacker that obtains a long-term secret becomes able to compute previous keys of sessions he had eavesdropped before. The limitation is introduced by construction because the long-term secret is directly used to compute the session key.

5.2.3.5 Attack III

Independently of our work, Kim et al. introduce a different type of attack against Sun et al.'s protocol [40]. Fig.5.15 describes the attack.

The protocol becomes susceptible to Kim et al.'s attack under the assumption that the hash function h is vulnerable to a guessing attack: given $g^{s_1+r_1}, \ldots, g^{s_t+r_t}, U_1, \ldots, U_t, r_1, \ldots, r_t$ (available by eavesdropping), it is possible to compute k such that $Auth = h(k, g^{s_1+r_1}, \ldots, g^{s_t+r_t}, U_1, \ldots, U_t, r_1, \ldots, r_t)$.

We skip other details here and invite the reader to address the original paper for two similar versions of the attack and further details [40].

We only remark that the attack is available both for insiders and outsiders and it remains valid for the improved version we have proposed in the previous subsection. However, we emphasize that the proposed attack is limited by the success of the guessing attack against the hash function. Similar attacks can be mounted against all protocols described within the current chapter, but we omit to describe them to avoid repetition. We only mentioned its existence to highlight that the security analysis of such protocols raised other authors interest as well.

5.2.4 Yuan et al. (2013)

5.2.4.1 Protocol Description

Yuan et al. introduced in 2013 a password-based GKT protocol that uses Shamir's SSS [83]. Fig.5.16 describes the protocol in detail.

The protocol is remarkable similar to the one of Harn and Lin [30], described in Subsection 5.2.1, except some differences that we underline next.

An obvious difference between the two constructions consists in the long-term secrets that are shared between the users and the KGC: Yuan et al.'s protocol requires a password $pw_i = pw_{ix}|| \ pw_{iy}$, while Harn and Lin's protocol needs a secret pair (x_i, y_i) (Users Registration Phase). The computational relation between the long-term secret and the random chosen numbers also differs: pw_{ix} tries to mask k_i (steps 1.2 and 3.2), while R_i is computational unrelated to x_i or y_i and sent in clear (step 3.3). We will show later that the relation between pw_{ix}, pw_{iy} and k_i reveals a new attack.

A second difference is that in Yuan et al.'s protocol the values M_i used to authenticate the random numbers k_i depend on the group members of the current session (steps 1.2 and 3.2), while in Harn and Lin's protocol $Auth_i$ is independent of the current session group (step 3.2).

A last notable difference appears in Round 4: the KGC sends t messages in case of Yuan et al.'s protocol (step 4.7), while a single broadcast message is enough in case of Harn and Lin's protocol (step 4.6). The former construction is therefore less efficient than the latter in this stage.

5.2.4.2 Attack I

We have mentioned in the previous subsection that Yuan et al.'s construction is very much alike to the protocol that Harn and Lin had been published three years before [30]. This similarity preserves a vulnerability [56], [57]: the protocol is susceptible to a replay attack analogous to the one that Nam et al. mounted against Harn and Lin's proposal [50].

As before, we consider U_a to be an insider whose goal is to reveal the long-term password of another user U_i, $i = 1, \ldots, m$, $i \neq a$. This gives the attacker the ability to obtain the session key of all sessions U_i is authorized for (even if U_a is unauthorized for) and therefore break the confidentiality of the protocol. We explain the attack in detail in Fig.5.17.

Compared to the protocol of Harn and Lin, U_a cannot replay an eavesdropped message from any previous session U_i participated to. The message must originate from a session between the adversary U_a and the victim U_i, otherwise the KGC quits in step 4.2 (the authorization string M_i depends on the current group members). However, U_a is an insider and hence he can initiate two sessions by himself: in the first session, he permits U_i to send his own message, while in the second session he impersonate U_i by sending the message eavesdropped in the first session. We mention that the same approach is possible for Harn and Lin's protocol also, but considered in Subsection 5.2.1.2 the original version of the attack.

Initialization. The KGC selects 2 large primes p and q and computes $n = pq$;

Users Registration. Each user $U_i, i = 1, \ldots, m$, shares a long-term secret password $pw_i = pw_{ix}||pw_{iy}$ with the KGC;

Round 1. User U_1:
1.1. chooses $k_1 \leftarrow^R \mathbb{Z}_n$;
1.2. computes $K_1 = pw_{1x} + k_1$ and $M_1 = h_1(U_1, \ldots, U_t, k_1)$;
1.3. sends a key generation request:
$$U_1 \rightarrow KGC : (U_1, \{U_1, \ldots, U_t\}, K_1, M_1)$$

Round 2. The KGC:
2.1. computes $k_1 = K_1 - pw_{1x}$;
2.2. checks if $M_1 = h_1(U_1, \ldots, U_t, k_1)$;
If the equality does not hold, he quits;
2.3. broadcasts:
$$KGC \rightarrow^*: \{U_1, \ldots, U_t\}$$

Round 3. Each user $U_i, i = 2, \ldots, t$:
3.1. chooses $k_i \leftarrow^R \mathbb{Z}_n$;
3.2. computes $K_i = pw_{ix} + k_i$ and $M_i = h_1(U_1, \ldots, U_t, k_i)$;
3.3. sends:
$$U_i \rightarrow KGC : (U_i, \{U_1, \ldots, U_t\}, K_i, M_i)$$

Round 4. The KGC:
4.1. computes $k_i = K_i - pw_{ix}$, i=2,. . . ,t;
4.2. checks if $M_i = h_1(U_1, \ldots, U_t, k_i)$, i=2,. . . ,t;
If at least one equality does not hold, he quits;
4.3. selects 2 random numbers x_{ta} and y_{ta} of lengths equal to pw_{ix} and pw_{iy};
4.4. generates the polynomial $f(x)$ of degree t that passes through the $t + 1$ points (x_{ta}, y_{ta}), $(pw_{1x}, pw_{1y} + k_1)$, . . . , $(pw_{tx}, pw_{ty} + k_t)$;
4.5. computes t additional points $P_1, \ldots, P_t$ on $f(x)$;
4.6. computes the verification messages $V_i = h_2(U_1, \ldots, U_t, P_1, \ldots, P_t, k_i)$, $i = 1, \ldots, t$;
4.7. sends, $i = 1, \ldots, t$:
$$KGC \rightarrow U_i : (P_1, \ldots, P_t, V_i)$$

Key Computation. Each user U_i, $i = 1, \ldots, t$:
5.1. checks if $V_i = h_2(U_1, \ldots, U_t, P_1, \ldots, P_t, k_i)$;
If the equality does not hold, he quits;
5.2. computes the group key $k = f(0)$ by interpolating the points $P_1, \ldots, P_t$ and $(pw_{ix}, pw_{iy} + k_i)$.

Figure 5.16: Original Version of Yuan et al.'s Group Key Transfer Protocol [83]

Step 1. U_a initiates a legitimate session of the protocol (s_1) with U_i;
Step 2. U_a eavesdrops on $(U_i, \{U_i, U_a\}, K_i, M_i)$ in Round 3 of the protocol;
Step 3. U_a initiates another legitimate session of the protocol (s_2) with U_i and uses the same value k_a for both session (s_1) and (s_2);
Step 4. U_a impersonate U_i in session (s_2) by sending in Round 3 the message $(U_i, \{U_i, U_a\}, K_i, M_i)$ he had eavesdropped in Step 2;
Step 5. U_a is an authorized user for both sessions, so he recovers the polynomials:

$$f(x)_{(s_j)} = a_{(s_j)}x^2 + b_{(s_j)}x + c_{(s_j)}, j = 1, 2$$

Step 6. Since $(pw_{ax}, pw_{ay} + k_a)$ and $(pw_{ix}, pw_{iy} + k_i)$ are valid points on $f(x)_{(s_j)}$, U_a knows that $f(pw_{ax})_{(s_1)} = f(pw_{ax})_{(s_2)} = pw_{ay} + k_a$ and $f(pw_{ix})_{(s_1)} = f(pw_{ix})_{(s_2)} = pw_{iy} + k_i$; therefore both pw_{ax} and pw_{ix} are roots of:

$$(a_{(s_1)} - a_{(s_2)})x^2 + (b_{(s_1)} - b_{(s_2)})x + (c_{(s_1)} - c_{(s_2)}) = 0$$

Step 7. U_a reveals the long-term password of U_i as:

$$pw_{ix} = pw_{ax}^{-1}(a_{(s_1)} - a_{(s_2)})^{-1}(c_{(s_1)} - c_{(s_2)}).$$
$$pw_{iy} = f(pw_{ix})_{(s_j)} - K_{i(s_j)} + pw_{ix}, \text{ for any } j = 1, 2$$

Figure 5.17: Replay Attack against the Original Version of Yuan et al.'s Protocol [56]

5.2.4.3 Improvement I

The attack revealed in the previous subsection is possible because the KGC cannot detect replay messages. We give next a countermeasure analogous to the one that Nam et al. proposed against Harn and Lin's protocol [56], [57]. Fig.5.18 exposes it in detail.

We highlight the main idea: for each session, the KGC selects a uniformly random value k_0, which he broadcasts to the participants (Round 2); then, the principals use it to compute the hash value M_i (Round 3). Since the value k_0 differs for distinct sessions, an eavesdropped value M_i in one session becomes invalid for other sessions - the equality in step 4.2 fails and hence the KGC quits.

We mention a slight modification in the protocol definition: Round 1 restricts to the key generation request, while U_1 performs the other steps in Round 3 (i.e. except the initiation request, U_1 behaves similar to the rest of the users). This approach is required in the improved version (because U_1 uses the value k_0 to compute M_i), but it could have been also adopted in the original version for simplicity of exposure and efficiency.

5.2.4.4 Attack II

Although the first improved version of Yuan et al.'s protocol stands against the replay attack, it remains vulnerable to an insider attack [57]. Fig.5.19 reveals the details.

The proposed attack differs from the replay attack in the sense that U_a does not rely on a previous eavesdropped message originated from U_i; hence, U_i is genuine (it is not impersonated anymore). On the other hand, it requires four sessions between the adversary and the victim. It is a natural assumption to consider that the protocol allows multiple sessions between the same parties. However, if it is considered suspicious that a single user initiates

Initialization. The KGC selects 2 large primes p and q and computes $n = pq$;

Users Registration. Each user $U_i, i = 1, \ldots, m$, shares a long-term secret password $pw_i = pw_{ix}||pw_{iy}$ with the KGC;

Round 1. User U_1:
1.1. sends a key generation request:
$$U_1 \rightarrow KGC : \{U_1, \ldots, U_t\}$$

Round 2. The KGC:
2.1. chooses $k_0 \leftarrow^R \mathbb{Z}_n$;
2.2. broadcasts:
$$KGC \rightarrow^*: (\{U_1, \ldots, U_t\}, k_0)$$

Round 3. Each user $U_i, i = 1, \ldots, t$:
3.1. chooses $k_i \leftarrow^R \mathbb{Z}_n$;
3.2. computes $K_i = pw_{ix} + k_i$ and $M_i = h_1(U_1, \ldots, U_t, k_i, k_0)$;
3.3. sends:
$$U_i \rightarrow KGC : (U_i, \{U_1, \ldots, U_t\}, K_i, M_i)$$

Round 4. The KGC:
4.1. computes $k_i = K_i - pw_{ix}$, i=1,. . . ,t;
4.2. checks if $M_i = h_1(U_1, \ldots, U_t, k_i, k_0)$, i=1,. . . ,t;
If at least one equality does not hold, he quits;
4.3. selects 2 random numbers x_{ta} and y_{ta} of lengths equal to pw_{ix} and pw_{iy};
4.4. generates the polynomial $f(x)$ of degree t that passes through the $t + 1$ points (x_{ta}, y_{ta}), $(pw_{1x}, pw_{1y} + k_1)$, . . . , $(pw_{tx}, pw_{ty} + k_t)$;
4.5. computes t additional points $P_1, \ldots, P_t$ on $f(x)$;
4.6. computes the verification messages $V_i = h_2(U_1, \ldots, U_t, P_1, \ldots, P_t, k_i, k_0)$, $i = 1, \ldots, t$;
4.7. sends, $i = 1, \ldots, t$:
$$KGC \rightarrow U_i : (P_1, \ldots, P_t, V_i)$$

Key Computation. Each user U_i, $i = 1, \ldots, t$:
5.1. checks if $V_i = h_2(U_1, \ldots, U_t, P_1, \ldots, P_t, k_i, k_0)$;
If the equality does not hold, he quits;
5.2. computes the group key $k = f(0)$ by interpolating the points $P_1, \ldots, P_t$ and $(pw_{ix}, pw_{iy} + k_i)$.

Figure 5.18: First Improved Version of Yuan et al.'s Protocol [56]

Step 1. U_a initiates (s_j), $j = 1, \ldots, 4$ four legitimate sessions of the protocol with U_i;
Step 2. U_a is an authorized user for all sessions, so he recovers the polynomials:
$$f(x)_{(s_j)} = a_{(s_j)}x^2 + b_{(s_j)}x + c_{(s_j)}, j = 1, \ldots, 4$$
Step 3. Since $(pw_{ix}, pw_{iy} + k_{i(s_j)})$ are valid points on $f(x)_{(s_j)}$, U_a obtains:
$$pw_{iy} + k_{i(s_j)} = a_{(s_j)}pw_{ix}^2 + b_{(s_j)}pw_{ix} + c_{(s_j)},\ j = 1, \ldots, 4$$
Step 4. U_a eavesdrops on $K_{i(s_j)}$, knows that $k_{i(s_j)} = K_{i(s_j)} - pw_{ix}$ and acquires:
$$pw_{iy} = a_{(s_j)}pw_{ix}^2 + (b_{(s_j)} + 1)pw_{ix} + c_{(s_j)} - K_{i(s_j)}, j = 1, \ldots, 4$$
Step 5. U_a eliminates pw_{iy} from the first two equalities $(j = 1, 2)$, respectively from the last two equalities $(j = 3, 4)$ and gets:
$$A_{(s_1s_2)}pw_{ix}^2 + B_{(s_1s_2)}pw_{ix} + C_{(s_1s_2)} = 0$$
$$A_{(s_3s_4)}pw_{ix}^2 + B_{(s_3s_4)}pw_{ix} + C_{(s_3s_4)} = 0$$
where:
$$A_{(s_1s_2)} = a_{(s_1)} - a_{(s_2)} \qquad A_{(s_3s_4)} = a_{(s_3)} - a_{(s_4)}$$
$$B_{(s_1s_2)} = b_{(s_1)} - b_{(s_2)} \qquad B_{(s_3s_4)} = b_{(s_3)} - b_{(s_4)}$$
$$C_{(s_1s_2)} = c_{(s_1)} - c_{(s_2)} - (K_{i(s_1)} - K_{i(s_2)}) \qquad C_{(s_3s_4)} = c_{(s_3)} - c_{(s_4)} - (K_{i(s_3)} - K_{i(s_4)})$$
Step 6. U_a reveals the long-term password of U_i as:
$$pw_{ix} = (A_{(s_1s_2)}C_{(s_3s_4)} - A_{(s_3s_4)}C_{(s_1s_2)})(A_{(s_3s_4)}B_{(s_1s_2)} - A_{(s_1s_2)}B_{(s_3s_4)})^{-1}$$
$$pw_{iy} = f(pw_{ix})_{(s_j)} - K_{i(s_j)} + pw_{ix}, \text{ for any } j = 1, \ldots, 4$$

Figure 5.19: Insider Attack against the First Improved Version of Yuan et al.'s Protocol [57]

the protocol multiple times with the same other participant, a coalition of insiders may mount the attack: each attacker initializes a different session with the victim U_i and finally they cooperate to disclose the long-term key password $pw_{ix}||pw_{iy}$.

We remark that after the log-term secret password is revealed, an impersonation attack is immediate: the adversary U_a uses (pw_{ix}, pw_{iy}) and pretends his identity is P_i.

5.2.4.5 Improvement II

The insider attack becomes possible because pw_{iy} can be expressed as the value of a polynomial with known coefficients in pw_{ix}. This permits the attacker to replace pw_{iy} and obtain a system of equations with the single unknown pw_{ix}. Fig.5.20 introduces a countermeasure [56].

We emphasize the main idea: the KGC generates a polynomial $f(x)$ that passes through $(pw_{ix}, h_3(U_1, \ldots, U_t, pw_{iy}, k_i, k_0))$ instead of $(pw_{ix}, pw_{iy} + k_i)$, $i = 1, \ldots, t$ (Round 4). This leads to the futility of the attack, since the argue fails due to the new form of the equations in Step 3:

$$h_3(U_1, \ldots, U_t, pw_{iy}, k_{i(s_j)}, k_0)) = a_{(s_j)}pw_{ix}^2 + b_{(s_j)}pw_{ix} + c_{(s_j)},\ j = 1, \ldots, 4.$$

Initialization. The KGC selects 2 large primes p and q and computes $n = pq$;

Users Registration. Each user $U_i, i = 1, \ldots, m$, shares a long-term secret password $pw_i = pw_{ix}||pw_{iy}$ with the KGC;

Round 1. User U_1:
1.1. sends a key generation request:
$$U_1 \rightarrow KGC : \{U_1, \ldots, U_t\}$$

Round 2. The KGC:
2.1. chooses $k_0 \leftarrow^R \mathbb{Z}_n$;
2.2. broadcasts:
$$KGC \rightarrow^*: (\{U_1, \ldots, U_t\}, k_0)$$

Round 3. Each user $U_i, i = 1, \ldots, t$:
3.1. chooses $k_i \leftarrow^R \mathbb{Z}_n$;
3.2. computes $K_i = pw_{ix} + k_i$ and $M_i = h_1(U_1, \ldots, U_t, k_i, k_0)$;
3.3. sends:
$$U_i \rightarrow KGC : (U_i, \{U_1, \ldots, U_t\}, K_i, M_i)$$

Round 4. The KGC:
4.1. computes $k_i = K_i - pw_{ix}$, i=1,. . . ,t;
4.2. checks if $M_i = h_1(U_1, \ldots, U_t, k_i, k_0)$, i=1,. . . ,t;
If at least one equality does not hold, he quits;
4.3. selects 2 random numbers x_{ta} and y_{ta} of lengths equal to pw_{ix} and pw_{iy};
4.4. generates the polynomial $f(x)$ of degree t that passes through the $t + 1$ points (x_{ta}, y_{ta}), $(pw_{1x}, h_3(U_1, \ldots, U_t, pw_{1y}, k_1, k_0))$, . . . , $(pw_{tx}, h_3(U_1, \ldots, U_t, pw_{ty}, k_t, k_0))$;
4.5. computes t additional points $P_1, \ldots, P_t$ on $f(x)$;
4.6. computes the verification messages $V_i = h_2(U_1, \ldots, U_t, P_1, \ldots, P_t, k_i, k_0)$, $i = 1, \ldots, t$;
4.7. sends, $i = 1, \ldots, t$:
$$KGC \rightarrow U_i : (P_1, \ldots, P_t, V_i)$$

Key Computation. Each user U_i, $i = 1, \ldots, t$:
5.1. checks if $V_i = h_2(U_1, \ldots, U_t, P_1, \ldots, P_t, k_i, k_0)$;
If the equality does not hold, he quits;
5.2. computes the group key $k = f(0)$ by interpolating the points $P_1, \ldots, P_t$ and $(pw_{ix}, h_3(U_1, \ldots, U_t, pw_{iy}, k_i, k_0))$.

Figure 5.20: Second Improved Version of Yuan et al.'s Protocol [56]

5.3 Research Contributions

First, we briefly reviewed the informal security notions that GKE protocols should satisfy and reminded the adversarial models and types of attacks that a GKE protocol must stand against [58] (Subsections 5.1.1 and 5.1.2). Second, we introduced a multitude of attacks against the very recent protocols described throughout this chapter and propose possible countermeasures: (1) a man-in-the-middle attack against the Yuan et al.'s improved version of Harn and Lin's protocol [59] (Subsection 5.2.1.6); (2) an insider attack and the corresponding countermeasures applied to Hsu et al.'s protocol [61] (Subsections 5.2.2.2, 5.2.2.3 and 5.2.2.4); (3) an insider attack, a known key attack and the corresponding countermeasure applied to Sun et al.'s protocol [60] (Subsections 5.2.3.2, 5.2.3.3 and 5.2.3.4); (4) a replay attack, an insider attack and the corresponding countermeasures applied to Yuan et al.'s protocol [56], [57] (Subsections 5.2.4.2, 5.2.4.3, 5.2.4.4 and 5.2.4.5).

Chapter 6

Formally Secure GKE based on Secret Sharing

6.1 Formal Security Models

The current section briefly reviews the history of formal security models designed for GKE protocols. It then focuses on the *sequence of games* technique used to organize security proofs and on the description of the GBG model, which we will both use in the security arguing of our proposed protocol in the next section. Finally, we mention some limitations of current security models.

6.1.1 Overview

The multitude of attacks revealed in the previous chapter clearly highlights the limitation of informal arguing and the necessity of stronger tools to prove the security of GKE protocols. Formal models consider the security requirements mentioned in Section 5.1.1 within a precise environment, specifying the trust assumptions, the relations between participants, the adversarial power, the communication medium and others relevant aspects.

Bresson et al. introduced the first security model (BCPQ model) for GKE protocols in 2001 [16], as a generalization of existing security models designed for two or three party protocols [4], [5], [6]. Their model was rapidly extended to allow dynamic groups (BCP model), meaning that the group membership may change during the protocol execution [14]. One year later, the same authors improved their work to stand against strong corruptions (BCP+ model), which permit an attacker to reveal the ephemeral internal state information of the user instances [15]. In 2005, Katz and Shin introduced security against insider attacks [38] (KS model). They considered the existence of malicious users[1] (by introducing a model within the Universally Composability (UC) framework), in the sense that even if a registered party always knows the key of sessions he is qualified for, he must be restricted to perform malicious actions; for example, he should not disclose session keys he is unauthorized for, modify the value of the common key as he desires or make honest users compute different

[1] We emphasize that within the definition of formal security models, an adversary is not considered to be a malicious user, but an external attacker who has access to the long-lived keys and/or ephemeral values used during the run of the protocol by making queries [17].

keys. Further, Bohli et al. introduced the notion of key contributiveness [11] and Bresson and Manulis considered strong corruptions [17]. At PKC'09, Gorantla, Boyd and González Nieto described a stronger security model (GBG model), which stands against key compromise impersonation (KCI) [29]. Two years later, Zhao et al. extended their work (eGBG model) and considered the leakage of the ephemeral keys (EKL) of the parties, which are used to derive the session key [85]. For a more detailed survey on group key security models, the reader may refer to [44] and [45].

For the rest of the chapter, we will restrict to the review of GBG model. We motivate our choice by the introduction of a new protocol that we prove secure in the GBG model (Section 6.2.2).

6.1.2 Game-based Proofs

Three different approaches exist for provable security [44]:

- *Computational Security.* It limits the computational power and the resources of the adversary, who is modeled as a PPT algorithm; it usually relies on the computational infeasibility of cryptographic problems or assumptions (for example CDH or DLP)[2];
- *Information-Theoretic Security.* Unlike the computational security, it does not limit the power nor the resources of the adversary, being the strongest notion of security; however, it may limit the access of the adversary to some knowledge that the legitimate participants have access to;
- *Symbolic Security.* It considers *black boxes* to simulate real constructions and it analyzes the security goals by using formal methods techniques and deductions based on a corresponding set of formal rules.

For the rest of our work, we restrict to computational security, which, besides its previously mentioned limitation, provides the advantage of allowing more practical constructions.

Computational security defines the security goals as interactions (*games*) between an *adversary* (a PPT algorithm that simulates the attacker) and a *challenger* (an algorithm that *simulates* the actions of set of legitimate participants). As both the adversary and the challenger are probabilistic algorithms, the game is modeled as a probability space. The attack is considered successful if an event Win (that models the attack) occurs with a probability that non-negligibly differs from a specified *target probability.* Typically, the target probability is either 0, either 1/2 (to specify the indistinguishability from a similar game in which the adversary interacts with a different challenger) [44], [69].

The *sequence of games* represents a technique used to organize a security proof. We review it next, based on the description given by Shoup [69].

The standard procedure within a game-based proof is to construct a sequence of games $\mathsf{Game\ i}, \mathsf{i} = 0, \ldots, n$, where $\mathsf{Game\ 0}$ is the original game played between the adversary $\mathcal{A}$ and the challenger under the appropriate settings of the cryptographic protocol. For each game within the sequence, $\mathsf{Win_i}$ denotes the event that the adversary $\mathcal{A}$ wins $\mathsf{Game\ i}, \mathsf{i} = 0, \ldots, n$. Therefore, $\mathsf{Win_0}$ specifies a successful attack.

[2]Please refer to Appendix A.2 for more details.

The proof shows that the probability to win two consecutive games is negligible in the security parameter $\mathcal{K}$:

$$|Pr[\mathsf{Win}_{\mathsf{i}+1}] - Pr[\mathsf{Win}_{\mathsf{i}}]| \leq negl(\mathcal{K}). \tag{6.1}$$

The sequence of games stops when the winning probability $Pr[\mathsf{Win}_{\mathsf{n}}]$ is negligibly close to the target probability. If this is the case, then the security of the cryptographic construction is proved.

For ease of analysis, the games are chosen such that each game in the sequence brings only small changes with respect to the previous game. Usually, four types of transitions between consecutive games are used [44], [69]:

1. *Transitions based on indistinguishability.* Let P_1 and P_2 be two computationally indistinguishable distributions. $\mathsf{Game\ i}$ and $\mathsf{Game\ i+1}$ are build such that it exists a PPT algorithm (*distinguisher*) $\mathcal{D}$ that given as input an element from P_1 outputs 1 with probability $\mathsf{Win}_{\mathsf{i}}$ and that given as input an element from P_2 outputs 1 with probability $\mathsf{Win}_{\mathsf{i+1}}$. As P_1 and P_2 are indistinguishable, $|Pr[\mathsf{Win}_{\mathsf{i+1}}] - Pr[\mathsf{Win}_{\mathsf{i}}]|$ is negligible. Usually, the two games are unified within one that is defined as $\mathsf{Game\ i}$ if the input is an element from P_1 and $\mathsf{Game\ i+1}$ if the input is an element from P_2;

2. *Transitions based on failure events.* $\mathsf{Game\ i}$ and $\mathsf{Game\ i+1}$ are build such that they are identical, unless a *failure event* F occurs, i.e. $\mathsf{Win}_{\mathsf{i}} \wedge \neg\mathsf{F}$ and $\mathsf{Win}_{\mathsf{i+1}} \wedge \neg\mathsf{F}$ are equivalent. From the Difference Lemma I[3] it follows that $|Pr[\mathsf{Win}_{\mathsf{i+1}}] - Pr[\mathsf{Win}_{\mathsf{i}}]| \leq Pr[\mathsf{F}]$, which means it becomes negligible if $Pr[\mathsf{F}]$ is negligible. Typically, the event F is chosen such that when it occurs a security assumption does not hold (e.g.: a collision is found for a hash function, a signature is forged, the DLP problem is broken).

3. *Transitions based on condition events.* $\mathsf{Game\ i+1}$ is build such that equals $\mathsf{Game\ i}$ when a given *condition* C occurs, i.e. $Pr[\mathsf{Win}_{\mathsf{i+1}}] = Pr[\mathsf{Win}_{\mathsf{i}}|\mathsf{C}]$. From the Difference Lemma II[4] it follows that $|Pr[\mathsf{Win}_{\mathsf{i+1}}] - Pr[\mathsf{Win}_{\mathsf{i}}]| \leq Pr[\neg\mathsf{C}]$, which means it becomes negligible if $Pr[\neg\mathsf{C}]$ is negligible. Transitions based on condition events mainly differs from the transitions based on failure events by the fact that it is not necessary to detect whether the event C occurs or not.

4. *Transitions based on bridging steps.* $\mathsf{Game\ i+1}$ is just a rewriting of $\mathsf{Game\ i}$ (i.e. $Pr[\mathsf{Win}_{\mathsf{i+1}}] = Pr[\mathsf{Win}_{\mathsf{i}}]$) which helps to keep the proof simple by preparing the requirements for the previous type of transitions. Trivially, $|Pr[\mathsf{Win}_{\mathsf{i+1}}] - Pr[\mathsf{Win}_{\mathsf{i}}]| = 0$ and hence negligible.

6.1.3 GBG Security Model

GBG is a computational security model that uses reductionist techniques (such as the sequence of games described before) to bring the adversary's probability to win a security game to a computationally infeasible cryptographic assumption.

[3]Please refer to Appendix B.2, Lemma B.1 for more details.
[4]Please refer to Appendix B.2, Lemma B.2 for more details.

Not all computational security models use the reductionist technique. For example, Katz and Shin's model [38] is defined within the UC framework, which provides indistinguishability proofs that rely on the computational indistinguishability between the execution of the security game in the ideal world and the real world. In the ideal world, a trusted party receives all the inputs and returns the corresponding outputs to the participants (process modeled by an *ideal functionality*), while in the real world the users exchange messages via communication channels and compute the outputs as specified by the construction of the protocol. A cryptographic protocol is proved secure in the UC model if an adversary in the real world can be simulated in the ideal world and the two executions cannot be distinguished by an observer [38], [44].

We overview next the GBG security model defined by Gorantla, Colin and González Nieto for GKE protocols [29].

Let $\mathcal{U} = \{U_1, \ldots, U_m\}$ be the set of m users that may take part to the GKE protocol. Each user can participate in concurrent executions (sessions), having different protocol partners within $\mathcal{U}$. In order to model this, each user $U \in \mathcal{U}$ has multiple instances, called *oracles* and denoted by Π_U^s, where U participates in the s-th session of the protocol. q_s denotes the upper bound for the number of (concurrent) sessions.

The protocol instance in which Π_U^s participate is uniquely identified by a *session id* sid_U^s known to all oracles participating in the same session. The session id may be generated in advance by the environment or it may be computed during the execution of the protocol.

For each session, after the run of the protocol, every instance Π_U^s either computes a session key k_U^s and enters an *accepted* state, either terminates without computing a session key and entering an accepted state. It is publicly available if an instance terminates with or without acceptance.

The *partner id* pid_U^s of an instance Π_U^s represents the set of identities of the parties to whom U wishes to establish a common session key, including himself.

Definition 6.1. ***(Partner)*** [29]. Two instances $\Pi_{U_i}^{s_i}$ and $\Pi_{U_j}^{s_j}$ are *partnered* if the following conditions hold:

1. both have accepted (i.e. have computed a session key);
2. $\mathsf{sid}_{U_i}^{s_i} = \mathsf{sid}_{U_j}^{s_j}$ (i.e. session ids are the same);
3. $\mathsf{pid}_{U_i}^{s_i} = \mathsf{pid}_{U_j}^{s_j}$ (i.e. partner ids are the same).

Each party U_i owns a public-private *long-term key* pair (pk_i, sk_i) generated in advance and therefore known to all instances $\Pi_{U_i}^{s_i}$, $i = 1, \ldots, m$. The long-term keys are certified by a trusted authority and usually generated within a Registration Phase.

Every instance Π_U^s maintains an internal state that contains all private ephemeral information used during the session. This may include for examples nonces used during the protocol execution. Once the oracle has accepted, the internal state information is erased. We highlight that the internal state does not contain the long-term key or any public information.

Definition 6.2. ***(Correctness)*** [29]. A GKE protocol is *correct* if the following conditions holds:

1. all instances have accepted;

2. all instances are partnered;

3. all instances have computed the same session group key.

An adversary $\mathcal{A}$ (which may for example ruin the correctness of the protocol) is modeled as a PPT algorithm that is assumed to have full control over the communication channel: he can modify, delete or insert messages (he is therefore an active adversary). $\mathcal{A}$ interacts with the group members by asking the following queries [29]:

- $\mathsf{Execute}(\mathsf{sid}, \mathsf{pid})$: prompts a complete execution of the protocol among the partners in pid using the session id sid and returns the transcript of the protocol (i.e. all messages exchanged during the execution) to $\mathcal{A}$. This query models passive attacks;

- $\mathsf{Send}(\Pi_U^s, M)$: sends the message M to the instance Π_U^s and returns the response generated by Π_U^s to $\mathcal{A}$. In the particular case of $M = (\mathsf{sid}, \mathsf{pid})$, the oracle Π_U^s is initiated with the session id sid and the partner id pid;

- $\mathsf{RevealKey}(\Pi_U^s)$: returns the group session key k_U^s to $\mathcal{A}$, if Π_U^s has accepted;

- $\mathsf{RevealState}(\Pi_U^s)$: returns the internal state of Π_U^s to $\mathcal{A}$. If the instance Π_U^s has accepted, the internal state is erased and hence the query returns nothing;

- $\mathsf{Corrupt}(U)$: returns the long-term private key sk of U to $\mathcal{A}$;

- $\mathsf{Test}(\Pi_U^s)$: generates a random bit b and returns the session key k_U^s of Π_U^s if $b = 1$ or a random value if $b = 0$ to $\mathcal{A}$. This query is allowed only once, anytime during the protocol and it is answered only if the oracle Π_U^s has previously accepted.

We highlight the difference between a *corrupted user* - $\mathsf{Corrupt}(U)$ reveals the long-term private key of U- and a *corrupted instance* - $\mathsf{RevealState}(\Pi_U^s)$ reveals the internal ephemeral private state information of an instance of U.

Definition 6.3. ***(Freshness)*** [29]. An instance Π_U^s is *fresh* if the following conditions hold:

1. Π_U^s and his partners have not been asked $\mathsf{RevealKey}$ after their acceptance;

2. Π_U^s and his partners have not been asked $\mathsf{RevealState}$ before their acceptance;

3. if $\Pi_{U_i}^{s_i}$ is a partner of $\Pi_{U_j}^{s_j}$ and U_i has been asked $\mathsf{Corrupt}$, then any message that $\mathcal{A}$ sends to $\Pi_{U_j}^{s_j}$ on behalf of $\Pi_{U_i}^{s_i}$ must come from $\Pi_{U_i}^{s_i}$ intended to $\Pi_{U_j}^{s_j}$.

Informally, *freshness* implies that the instance has not been trivially compromised by exposing the secret key and that the adversary is an outsider ($\mathcal{A}$ cannot impersonate users of whom he knows their secret key) [44].

We give next the security definitions, as described by the GBG model [29] and specify the informal security notions they imply. We specify that the AKE and MA security are stronger definitions of the ones introduced by Bresson and Manulis [17] (in order to model KCI attacks), while the contributiveness brings nothing new.

Definition 6.4. ***(AKE Security)*** [29]. Let $\mathcal{A}$ be an active adversary against the AKE security of a GKE protocol and $b \leftarrow^R \{0,1\}$. $\mathsf{Game}^{\mathsf{AKE}}$ is defined as follows:

- in Stage 1, $\mathcal{A}$ is allowed to make Execute, Send, RevealKey, RevealState and Corrupt queries. At the end of the stage, $\mathcal{A}$ asks a Test query to a fresh instance Π_U^s which has accepted and receives $k_1 = k_U^s$ if $b = 1$ or $k_0 \leftarrow^R Sp(k)$ if $b = 0$;
- in Stage 2, the adversary $\mathcal{A}$ continues to make queries. At the end of the stage, $\mathcal{A}$ outputs a bit b'.

$\mathcal{A}$ wins $\mathsf{Game}^{\mathsf{AKE}}$ if: (1) $b' = b$ and (2) the instance Π_U^s remained fresh until the end of $\mathcal{A}$'s execution. Let $\mathsf{Win}^{\mathsf{AKE}}$ be the winning probability of $\mathcal{A}$ in the AKE security game. Then the advantage of $\mathcal{A}$ in winning the game is:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{AKE}} = |2Pr[\mathsf{Win}^{\mathsf{AKE}}] - 1|.$$

A protocol is called AKE secure if $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{AKE}}$ is negligible in the security parameter $\mathcal{K}$ for any PPT adversary $\mathcal{A}$.

Informally, AKE (Authenticated Key Exchange) security implies: *key confidentiality* (unauthorized parties cannot recover the key), *forward secrecy* ($\mathcal{A}$ may obtain the long-term private keys of the users, but this has no impact on the confidentiality of the keys established in previous sessions of the protocol), *known key security* ($\mathcal{A}$ may obtain keys from previous sessions, but this has no impact on the confidentiality of the current session key), *KCI resilience* ($\mathcal{A}$ can corrupt the owner of the Test query, but it is not able to impersonate any of his partners to him; otherwise Π_U^s is not fresh) [29], [44].

Definition 6.5. ***(MA Security)*** [29]. Let $\mathcal{A}$ be an active adversary against the MA security of a GKE protocol that is allowed to make Execute, Send, RevealKey, RevealState and Corrupt queries. $\mathcal{A}$ wins $\mathsf{Game}^{\mathsf{MA}}$ if at some point during the interaction there exists an uncorrupted instance $\Pi_{U_i}^{s_i}$ (although the user U_i may be corrupted) that has accepted with a key $k_{U_i}^{s_i}$ and another user $U_j \in \mathsf{pid}_{U_i}^{s_i}$ that is uncorrupted at the time $\Pi_{U_i}^{s_i}$ accepts such that:

1. there exists no instance $\Pi_{U_j}^{s_j}$ with $(\mathsf{pid}_{U_j}^{s_j}, \mathsf{sid}_{U_j}^{s_j}) = (\mathsf{pid}_{U_i}^{s_i}, \mathsf{sid}_{U_i}^{s_i})$ or
2. there exists an instance $\Pi_{U_j}^{s_j}$ with $(\mathsf{pid}_{U_j}^{s_j}, \mathsf{sid}_{U_j}^{s_j}) = (\mathsf{pid}_{U_i}^{s_i}, \mathsf{sid}_{U_i}^{s_i})$ and has accepted with $k_{U_j}^{s_j} \neq k_{U_i}^{s_i}$.

Let $\mathsf{Win}^{\mathsf{MA}}$ be the winning probability of $\mathcal{A}$ in the MA security game. Then the advantage of $\mathcal{A}$ in winning the game is:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MA}} = Pr[\mathsf{Win}^{\mathsf{MA}}].$$

A protocol is called MA secure if $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MA}}$ is negligible in the security parameter $\mathcal{K}$ for any PPT adversary $\mathcal{A}$.

Informally, MA (Mutual Authentication) security implies: *mutual authentication* of the parties involved in the GKE protocol, *unknown key share* resilience ($\mathcal{A}$ cannot make a participant believe that the key is shared with one party when in fact it is shared with another), *key confirmation* (each user is assured that every other protocol participant owns the session group key), *KCI* resilience ($\mathcal{A}$ may obtain the long-term private key of a user U_i, but he cannot execute the protocol in his behalf; otherwise $\Pi_{U_i}^{s_i}$ is corrupted) [29],[44].

Definition 6.6. ***(Contributiveness)*** [17], [29]. Let $\mathcal{A}$ be an active adversary against the contributiveness of a GKE protocol that is allowed to make $\mathsf{Execute}$, Send, $\mathsf{RevealKey}$, $\mathsf{RevealState}$ and $\mathsf{Corrupt}$ queries. $\mathsf{Game}^{\mathsf{Con}}$ is defined as follows:

- in $\mathsf{Prepare}$ stage, $\mathcal{A}$ queries the instances and outputs some state information ζ and a key k'. At the end of the stage a set Π is build such that it consists of uncorrupted instances which have been asked either $\mathsf{Execute}$ or Send queries;
- in Attack stage, on input (ζ, Π), $\mathcal{A}$ interacts with the instances as in the $\mathsf{Prepare}$ stage. At the end of the stage he outputs (U, s).

$\mathcal{A}$ wins $\mathsf{Game}^{\mathsf{Con}}$ if an instance Π_U^s of an uncorrupted party U has terminated accepting k' with $\Pi_U^s \notin \Pi$. Let $\mathsf{Win}^{\mathsf{Con}}$ be the winning probability of $\mathcal{A}$ in the contributiveness game. Then the advantage of $\mathcal{A}$ in winning the game is:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Con}} = Pr[\mathsf{Win}^{\mathsf{Con}}].$$

A protocol achieves contributiveness if $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Con}}$ is negligible in the security parameter $\mathcal{K}$ for any PPT adversary $\mathcal{A}$.

Informally, the contributiveness implies (in case of GKA protocols) equal contribution of the parties to the established key and thus *key freshness*, *key randomness* and *key unpredictability* [29], [44].

GBG model does not consider EKL: the adversary is not allowed to make $\mathsf{RevealState}$ queries during the test session (otherwise the session is not fresh). This means that a protocol may become susceptible to a EKL attack if the ephemeral information of any of the users has been disclosed. Zhao et al. extend GBG model into eGBG model that overcomes this disadvantage. They modify the definition of freshness such that it permits the adversary to find the long-lived keys and ephemeral keys of parties involved except both these values for the participants in the test session [85]. More, they remove the limitation of the adversary to determine the internal state after the instance has accepted by maintaining the information available after the acceptance (i.e. the internal state is not erased any more). We skip more details here, but suggest the reader to refer to the original paper [85] if interested.

6.1.4 Limitations of the Current Models

Formal security models appeared as a natural way to overcome the inappropriate informal arguing. We have seen in Chapter 5 that the insecurity of a cryptographic protocol is ruined when it is shown vulnerable to at least one attack. However, even if no attack is revealed, that does not certify that the construction is secure. Security models demonstrate the impossibility

of successful attacks under specific circumstances: whenever a protocol is (correctly) proved secure within a security model, then no adversary defined within the corresponding settings may successfully mount an attack considered by the model's specifications. Despite their great importance in analyzing secure cryptographic protocols, we must admit some limitations.

First, there are no explicit criteria a good security model must fulfill [44]. Manulis considered that a good security model must be (at least) *abstract* (i.e. independent of the implementation or assumptions), *self-contained* (i.e. not based on any external assumptions or unspecified parameters), *precise* (i.e. rigorous, without any ambiguous interpretations) and *modular* (i.e. allows security proofs for only certain specific goals).

Second, current group key security models were designed to deal with the theoretical aspect of cryptographic protocols. They cannot provide security against improper implementation or usage. Different models are considered to analyze the physical aspects of cryptographic protocols. Although they do not particularly consider GKE, we mention as examples the model of a physical computer as a combination of a theoretical machine and a leakage function introduced by Micali and Reyzin [47] and the formal model to analyze side-channel attacks given by Standaert et al. [70].

Third, some limitations exist concerning the adversarial capabilities: all the security models mentioned throughout this chapter ignore DoS or fault-tolerance [17]. Therefore, protocols remain vulnerable to this kind of attacks, even if they are formally proved secure. The attacks restrict the applicability of a protocol and may ruin it in the sense that participants are not capable to compute a common secret key or end up with different keys, which leads to the futility of the protocol.

Last, we remark that it may be a difficult task to find the correct tradeoff between the required level of security and the efficiency of a protocol: usually cryptographic constructions proved to satisfy more security properties require more computational power or transmission overhead, which may turn them inappropriate for particular applications. This is not an actual limitation of the security models themselves, but we consider it important to mention.

6.2 GKE Protocols

The current section presents two GKE constructions based on secret sharing that are proved secure in formal security models. Although much research has been done in the field of secret sharing based GKE, very few constructions benefit of a formal security analysis. This motivates us to introduce a new GKE protocol that we prove secure in the GBG model. The second part of this section describes our proposal.

As usual, we consider the notations from the List of Symbols and Notations and assume the reader is familiar with the notions reviewed in Appendix A. For ease of exposure and without loss of generality, we adopt the same perspective from the original works and consider that the initiator intend to establish a key with all group members (i.e. $\{U_1, \ldots, U_m\}$, including himself).

6.2.1 Existing work

6.2.1.1 Bresson and Catalano (2004)

Bresson and Catalano introduced in 2004 a GKA protocol that they proved secure in the BCP model [13]. Fig.6.1 describes the construction in detail. Contrary to the rest of the protocols, we specify each time if the computation is performed in G or Q to avoid confusion.

The construction uses Shamir's secret sharing, being somehow similar to, but more efficient than a previous GKA protocol introduced by Li and Pieprzyk five years before [43]. Besides secret sharing techniques, the protocol also uses ElGamal encryption.

The authors proved their work secure under the DDH assumption[5] within the ROM[6]. The random oracle assumption is required such that no information is leaked in the Key Confirmation Phase, where the hash function H has the role to *hide* the value of the group session key. However, we mention that the ROM is only used for efficiency and the proof also stands in the standard model, by replacing the random oracle with a one-way function [13].

The formal security is argued within the BCP model [14], which by definition does not consider strong corruptions (i.e. the adversary is cannot obtain private ephemeral information of participants). The authors themselves admit that they deliberately bypass the case when the players behave malicious in the sense that they do not act accordingly to the protocol execution. They only allow some of the participants to posses a *rushing* behavior, which permits them to wait the reception of the messages originating from the others parties and then act accordingly [13].

As a drawback of the protocol, Kim et al. remark its inefficiency in computation: each party should perform more than m modular exponentiations, m signature generations and $m-1$ signature verifications [39]. Dutta and Barua highlighted the communication overload: more than m^2 messages sent during the protocol execution and a large number of bits transmitted over the communication channel [25]. In addition, we notice that broadcast communication is not sufficient for the protocol execution; it requires unicast messaging. Last, we emphasize that even if the protocol apparently runs for two rounds, the key confirmation requires an extra round.

6.2.1.2 Cao et al. (2008)

Cao et al. defined in 2008 a GKA protocol based on secret sharing which they proved secure within the UC framework [20]. Fig.6.2 describes the construction in detail. Unlike the rest of the protocols, which are given in the multiplicative notation, we stick to the additive notation used in the original paper.

The construction uses Shamir's secret sharing scheme, but unlike all the other protocols presented throughout this work, it performs polynomial interpolation twice (in Round 2 and the Key Computation Phase). Besides secret sharing techniques, the protocol uses bilinear mapping and it is identity based. We therefore introduce a new notation: ID_i as the identity of the user U_i, $i = 1, \ldots, m$.

The authors proved their work secure under the CDH assumption[7] (i.e. in additive

[5] Please refer to Appendix A.2, Definition A.12 for more details.
[6] Please refer to Appendix A.1, Section A.1.6 for more details.
[7] Please refer to Appendix A.2, Definition A.11 for more details.

Initialization. Let G be a multiplicative group of prime order p, $Q \subset G$ a subgroup of prime order q (i.e. $q|p-1$) with g as generator and sid the session id, which is generated in advance to the protocol execution.

Users Registration. Let $\Sigma.\mathsf{Sig}_{U_i}$ be the signing algorithm of U_i and $\Sigma.\mathsf{Verify}_{U_i}$ the corresponding verification algorithm, $i = 1, \ldots, m$. In addition, each user U_i owns a public-private key pair (pk_i, sk_i) s.t. $pk_i = g^{sk_i}$ in G.

Round 1. Each user U_i, $i = 1, \ldots, m$:

1.1. chooses $a_i \leftarrow^R Q$ and $r_i, b_{i,1}, \ldots, b_{i,m-1} \leftarrow^R \mathbb{Z}_q$;

1.2. generates the polynomial $f_i(x) = r_i + b_{i,1}x + \cdots + b_{i,m-1}x^{m-1} \pmod q$;

1.3. chooses $k_i \leftarrow^R \mathbb{Z}_q$;

1.4. computes $C_{i,j} = (A_{i,j}, B_{i,j}) = (g^{k_i} \pmod p, a_i pk_j^{k_i} \pmod p)$, $j = 1, \ldots, m$, $j \neq i$;

1.5. sends:

$$U_i \rightarrow U_j : (C_{i,j}, f_i(j), \sigma_{i,j} = \Sigma.\mathsf{Sign}_{U_i}(C_{i,j}||f_i(j)||\mathsf{sid}))$$

Round 2. Each user U_i, $i = 1, \ldots, m$:

2.1. checks if $\mathsf{Verify}_{U_j}(C_{j,i}||f_j(i)||\mathsf{sid}, \sigma_{j,i}) = 1$, $j = 1, \ldots, m$, $j \neq i$;
If at least one equality does not hold, he quits;

2.2. computes $A_i = \prod_{j \neq i} A_{j,i} \pmod p$ and $B_i = a_i \prod_{j \neq i} B_{j,i} \pmod p$;

2.3. computes $a = B_i A_i^{-sk_i} = \prod_{j=1}^{m} a_i$;

2.4. computes $f_i = f_i(i) + \sum_{j \neq i} f_j(i) \pmod q$;

2.5. broadcasts:

$$U_i \rightarrow^*: (f_i, \Sigma.\mathsf{Sign}_{U_i}(f_i||\mathsf{sid}))$$

Key Computation. Each user U_i, $i = 1, \ldots, m$:

3.1. generates the polynomial $f(x)$ of degree $m-1$ that passes through the m points (j, f_j), $j = 1, \ldots, m$;

3.2. computes $r = f(0)$, $k' = ag^r \pmod p$ and the secret key $k = \mathsf{H}(k')$;

Key Confirmation. Each user U_i, $i = 1, \ldots, m$:

4.1. computes $s_i = \mathsf{H}(k'||\mathsf{sid})$;

4.2. broadcasts:

$$U_i \rightarrow^*: (s_i, \Sigma.\mathsf{Sign}_{U_i}(s_i||\mathsf{sid}))$$

4.3. checks if all the m broadcast messages are equal.
If at least one message differs, he quits. Otherwise, he accepts with the key k.

Figure 6.1: Bresson and Catalano's Group Key Agreement Protocol [13]

Initialization. Let G_1 and G_2 be 2 groups of prime order p, P a generator of G_1, $e : G_1 \to G_2$ a bilinear map, (P_{Pub}, s) a master public-secret key pair s.t. $P_{Pub} = sP$ in G_1, $v_0, v_1 \in \{0,1\}^p$ 2 random values and sid the session id, which is generated in advance to the protocol execution.

Users Registration. Each user U_i, $i = 1, \ldots, m$, identified by his identity ID_i, owns a public-private key pair (Q_i, S_i), where $Q_i = \mathsf{H}_1(ID_i)$ and $S_i = sQ_i \in G_1$.

Round 1. Each user U_i, $i = 1, \ldots, m$:

1.1. chooses $r_i, r'_i \leftarrow^R \mathbb{Z}_p^*$;

1.2. computes $O_i = r_i P$ and $O'_i = r'_i P$;

1.3. broadcasts:

$$U_i \to^*: (\mathsf{sid}, O_i, O'_i)$$

Round 2. Each user U_i, $i = 1, \ldots, m$:

2.1. chooses $K_i \leftarrow^R \mathbb{Z}_p$;

2.2. generates the polynomial $f_i(x) = K_i + a_{i_1}x + \cdots + a_{i_{m-1}}x^{m-1}$ of degree $m-1$ that passes through the m points $(0, K_i)$ and $(j, \mathsf{H}_3(r_i O_j))$, $j = 1, \ldots, m$, $j \neq i$;

2.3. computes $P_{ij} = f_i(m+j)$, $j = 1, \ldots, m$, $j \neq i$, $P_i = P_{i1}||\ldots||P_{im}$, $O = O_1||\ldots||O_m$, $O' = O'_1||\ldots||O'_m$, $h_i = \mathsf{H}_2(P_i||O||O'||K_i||\mathsf{sid})$ and $\delta_i = r_i P_{Pub} + h_i S_i$;

2.4. broadcasts:

$$U_i \to^*: (\mathsf{sid}, P_i, \delta_i)$$

Key Computation. Each user U_i, $i = 1, \ldots, m$:

3.1. generates the polynomial $f_j(x)$, $j = 1, \ldots m$, $j \neq i$ of degree $m-1$ that passes through the m points $(i, \mathsf{H}_3(r_i O_j)))$ and $(m+l, P_{jl})$, $l = 1, \ldots, m$, $l \neq j$;

3.2. computes $K_j = f_j(0)$;

3.3. checks if $e(\sum_{j=1}^{m} \delta_j, P) = e(\sum_{j=1}^{m} (O_j + h_j Q_j), P_{Pub})$.
If the verification does not hold, he quits.

3.4. computes $k' = \mathsf{H}_4(K_1 + \cdots + K_n)$, $h' = \mathsf{H}_2(\mathsf{H}_4(k', v_0), \mathsf{sid})$ and $\delta'_i = r'_i P_{pub} + h' S_i$;

3.5. computes the key $k = \mathsf{H}_4(k', v_1)$;

Key Confirmation. Each user U_i, $i = 1, \ldots, m$:

4.1. broadcasts:

$$U_i \to^*: (\mathsf{sid}, ID_i, \delta'_i)$$

4.2. checks if $e(\sum_{j=1}^{m} \delta'_j, P) = e(\sum_{j=1}^{m} (O'_j + h' Q_j), P_{Pub})$.
If at least one equality does not hold, he quits. Otherwise, he accepts with the key k.

Figure 6.2: Cao et al.'s Group Key Agreement Protocol [20]

notation, given $P_{Pub} = sP$ and P it is computationally infeasible to compute s) within the ROM[8] (i.e. the hash functions $\mathsf{H_i}$, $i = 1, \ldots, 4$ are modeled as random oracles).

The formal security is argued within the original UC framework model proposed by Katz and Shin [38], whose limitations were highlighted by Hofheinz et al. [31] and Bohli et al. [11]. We only remind here two important drawbacks: (1) it skips the exact settings for key contributiveness: even if a construction is proved secure within the model settings, it is still possible for a malicious participant to obtain key control over the session group key [31], [44]; (2) it assumes that the session id is always available before the protocol's execution [11].

Similar to the Bresson and Catalano's protocol, we remark that even it may seem that the protocol runs for only two rounds, in fact it requires an additional round for key confirmation. Another similar aspect is that both protocols restrict their applicability to static groups. As an advantages to the previous mentioned construction, Cao et al.'s protocol uses only broadcast messaging and the communication overload is lower: $3m$ messages sent during the protocol execution.

6.2.2 A Novel GKA Protocol

6.2.2.1 Description

We introduced a new GKA protocol based on perfect secret m-sharing [62]. Fig. 6.3 describes the protocol in detail.

The main idea is that each user U_i uniformly selects a random r_i (later a share in a $(2, 2)$ all-or-nothing scheme) and broadcasts its secured value through a trapdoor function F (Rounds 1 and 2). The initiator (U_1, without loss of generality) computes the session key k based on the received values and the session id, invokes secret m-sharing on inputs k and $r_2, \ldots, r_m$ and extracts the second shares $r'_2, \ldots, r'_m$, which he then broadcasts (Round 3). Each user U_i can recover the agreed session key k from only his own shares r_i and r'_i (Key Computation Phase).

Key confirmation is achieved by default due to secret m-sharing: U_i knows r'_j (step 3.3), computes the corresponding r_j such that $\{r_j, r'_j\}$ represents a valid set of shares for k (step 5.1) and verifies that r_j is the genuine value chosen by U_j (step 5.2). Hence, U_i is sure that U_j uses the correct values $\{r_j, r'_j\}$ as inputs for the reconstruction algorithm and obtains the same key. However, an adversary may prevent the last broadcast message (step 3.3) to arrive to one or more users, who become unable to recover the key. This represents a DoS attack, which is always detected, but leads to the futility of the protocol. We do not consider this as a weakness of our proposal, since GKE security models ignore DoS scenarios (as we have already mentioned in Subsection 6.1.4) and therefore all existing provable secure GKA protocols are susceptible to such attacks.

The construction requires a secret m-sharing scheme that given as input a secret k and the shares $r_2, \ldots, r_m$ permits to compute the second shares $r'_2, \ldots, r'_m$ such that $\mathsf{Share}(k, 2) = \{r_i, r'_i\}$ (or, equivalent $\mathsf{Rec}(r_i, r'_i) = k$), $i = 2, \ldots, m$. We emphasize that this does not restrict the applicability, since efficient schemes with such property exist - for example the secret m-sharing scheme in Fig.2.3.

[8] Please refer to Appendix A.1, Section A.1.6 for more details.

Initialization. Let $\mathcal{F} = (\mathsf{Gen}, \mathsf{F}, \mathsf{F}^{-1})$ be a trapdoor function and SS a secret m-sharing scheme.

Users Registration. Let $\Sigma.\mathsf{Sig}_{U_i}$ be the signing algorithm of U_i and $\Sigma.\mathsf{Verify}_{U_i}$ the corresponding verification algorithm, $i = 1, \ldots, m$.

Round 1. User U_1:
1.1. runs $\mathcal{F}.\mathsf{Gen}$ to obtain a public-private key pair (pk, sk);
1.2. chooses $r_1 \leftarrow^R \{0,1\}^{\mathcal{K}}$;
1.3. broadcasts:
$$U_1 \rightarrow^*: (\mathcal{U}, pk, \mathsf{F}(pk, r_1), \sigma = \Sigma.\mathsf{Sign}_{U_1}(\mathcal{U}||pk||\mathsf{F}(pk, r_1)));$$

Round 2. Each user U_i, $i = 2, \ldots, m$:
2.1. checks if $\Sigma.\mathsf{Verify}_{U_1}(\mathcal{U}||pk||\mathsf{F}(pk, r_1), \sigma) = 1$.
If the equality does not hold, he quits;
2.2. chooses $r_i \leftarrow^R \{0,1\}^{\mathcal{K}}$;
2.3. broadcasts:
$$U_i \rightarrow^*: (\mathsf{F}(pk, r_i),\ \sigma_i = \Sigma.\mathsf{Sign}_{U_i}(\mathcal{U}||pk||\mathsf{F}(pk, r_i)));$$

Round 3. User U_1:
3.1. checks if $\Sigma.\mathsf{Verify}_{U_i}(\mathcal{U}||pk||\mathsf{F}(pk, r_i), \sigma_i) = 1$, $i = 2, \ldots, m$.
If at least one equality does not hold, he restarts the protocol;
3.2. computes $\mathsf{sid}_{U_1} = \mathsf{F}(pk, r_1)|| \ldots ||\mathsf{F}(pk, r_m)$, $r_i = \mathsf{F}^{-1}(sk, \mathsf{F}(pk, r_i))$, the session key $k = \mathsf{H}(\mathsf{sid}_{U_1}||r_1||r_2|| \ldots ||r_m)$ and r'_i s.t. $\mathcal{SS}.\mathsf{Share}(k, 2) = \{r_i, r'_i\}$, $i = 2, \ldots, m$;
3.3. broadcasts:
$$U_1 \rightarrow^*: (r'_2, \ldots r'_m, \sigma' = \Sigma.\mathsf{Sign}_{U_1}(\mathcal{U}||pk||r'_2|| \ldots ||r'_m||\mathsf{sid}_{U_1}));$$

Key Computation. Each user U_i, $i = 2, \ldots, m$:
4.1. checks if $\Sigma.\mathsf{Verify}_{U_j}(\mathcal{U}||pk||\mathsf{F}(pk, r_j), \sigma_j) = 1$, $j = 2, \ldots m$, $j \neq i$.
If at least one equality does not hold, he quits;
4.2. computes $\mathsf{sid}_{U_i} = \mathsf{F}(pk, r_1)|| \ldots ||\mathsf{F}(pk, r_m)$ and $k = \mathcal{SS}.\mathsf{Rec}(r_i, r'_i)$;
4.3. checks if $\Sigma.\mathsf{Verify}_{U_1}(\mathcal{U}||pk||r'_2|| \ldots ||r'_m||\mathsf{sid}_{U_i}, \sigma') = 1$.
If the equality holds, he accepts the key k; otherwise he quits;

Key Confirmation. Each user U_i, $i = 2, \ldots, m$:
5.1. computes r_j s.t. $\mathcal{SS}.\mathsf{Share}(k, 2) = \{r_j, r'_j\}$, $j = 2, \ldots m$, $j \neq i$;
5.2. checks if $\mathsf{F}(pk, r_j)$ equals the one sent in step 2.3.
If at least one equality does not hold, he quits.

Figure 6.3: Olimid's Group Key Agreement Protocol [62]

6.2.2.2 Formal Proofs of Security

We next prove our protocol security in the GBG model [62].

Theorem 6.1. ***(AKE Security)*** [62]. If the signature scheme Σ is UF-CMA, the trapdoor function $\mathcal{F}$ is secure, the hash function H is a random oracle and the secret m-sharing scheme $\mathcal{SS}$ is perfect then our protocol is AKE secure and

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{AKE}} \leq 2m^2\mathsf{Adv}_{\mathcal{A},\Sigma}^{\mathsf{UF-CMA}} + \frac{(q_s+q_r)^2}{2^{\mathcal{K}-1}} + \frac{(m+1){q_s}^2}{2^{\mathcal{K}-1}} + 2\mathsf{Adv}_{\mathcal{A},\mathcal{F}}.$$

Proof. We prove by a sequence of games. Let $\mathsf{Win}_{\mathsf{i}}^{\mathsf{AKE}}$ be the event that the adversary $\mathcal{A}$ wins $\mathsf{Game\ i}, \mathsf{i} = 0, \ldots, 4$.

Let **Game 0** be the original AKE security game. By definition, we have:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{AKE}} = |2Pr[\mathsf{Win}_0^{\mathsf{AKE}}] - 1|. \tag{6.2}$$

Let **Game 1** be the same as $\mathsf{Game\ 0}$, except that the simulation fails if an event Forge occurs. Hence:

$$|Pr[\mathsf{Win}_1^{\mathsf{AKE}}] - Pr[\mathsf{Win}_0^{\mathsf{AKE}}]| \leq Pr[\mathsf{Forge}]. \tag{6.3}$$

The event Forge simulates a successfully forgery on an honest user signature: $\mathcal{A}$ issues a Send query on some message (M, σ), where σ is a valid signature on M that has not been previously output by an oracle Π_U^s before querying $\mathsf{Corrupt}(U)$.

We follow the idea from [29] to estimate $Pr[\mathsf{Forge}]$ and construct an UF-CMA forger $\mathcal{F}_\Sigma$ against the signature scheme Σ using the adversary $\mathcal{A}$. $\mathcal{F}_\Sigma$ is given a public key pk^*, which he assigns uniformly random to one of the m parties. The public-private key pairs for all the other $m-1$ participants are generated honestly by using $\Sigma.\mathsf{Gen}$. $\mathcal{A}$ may corrupt $\mathcal{U} \setminus \{U^*\}$ through a KCI attack, but must remain passive on their behalf. The forger simulates all queries of $\mathcal{A}$: he executes all operations by himself for the corrupted parties or he obtains the signatures from his signing oracle for the uncorrupted party.

Assuming Forge occurs, $\mathcal{A}$ outputs a valid message-signature pair under a public key pk_i that matches pk^* with probability $1/m$. The probability that $\mathcal{A}$ did not corrupt the user that he had previously assigned pk^* is $1/m$. Therefore:

$$Pr[\mathsf{Forge}] \leq m^2\mathsf{Adv}_{\mathcal{A},\Sigma}^{\mathsf{UF-CMA}}. \tag{6.4}$$

Let **Game 2** be the same as $\mathsf{Game\ 1}$, except that the simulation fails if an event $\mathsf{Collision}$ occurs. Hence:

$$|Pr[\mathsf{Win}_2^{\mathsf{AKE}}] - Pr[\mathsf{Win}_1^{\mathsf{AKE}}]| \leq Pr[\mathsf{Collision}]. \tag{6.5}$$

The event $\mathsf{Collision}$ occurs when the random oracle H produces a collision for any of its inputs. Since the total number of random oracle queries is bounded by $q_s + q_r$:

$$Pr[\mathsf{Collision}] \leq \frac{(q_s+q_r)^2}{2^{\mathcal{K}}}. \tag{6.6}$$

Let **Game 3** be the same as Game 2, except that the simulation fails if an event Repeat occurs. Hence:

$$|Pr[\mathsf{Win}_3^{\mathsf{AKE}}] - Pr[\mathsf{Win}_2^{\mathsf{AKE}}]| \leq Pr[\mathsf{Repeat}]. \tag{6.7}$$

The event Repeat simulates a replay attack: it occurs when the same public-private key pair (pk, sk) is used in different sessions (event bounded by $q_s^2/2^{\mathcal{K}}$) or when a party uses the same value r_i in different sessions (event bounded by $mq_s^2/2^{\mathcal{K}}$).

Hence, we get:

$$Pr[\mathsf{Repeat}] \leq \frac{(m+1){q_s}^2}{2^{\mathcal{K}}}. \tag{6.8}$$

Note that this last game eliminates forgeries and replay attacks.

Let **Game 4** be the same as Game 3, except that the simulation fails if $\mathcal{A}$ is able to compute a value r_i, $i = 1, \ldots, n$. Hence:

$$|Pr[\mathsf{Win}_4^{\mathsf{AKE}}] - Pr[\mathsf{Win}_3^{\mathsf{AKE}}]| \leq \mathsf{Adv}_{\mathcal{A},\mathcal{F}}. \tag{6.9}$$

Since only r'_i, $i = 2, \ldots, m$ are available for the adversary in this last game and $\mathcal{SS}$ is a perfect secret m-sharing scheme, $\mathcal{A}$ has no advantage in finding the secret:

$$|Pr[\mathsf{Win}_4^{\mathsf{AKE}}]| = 0. \tag{6.10}$$

By combining (6.2) - (6.10) we get:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{AKE}} \leq 2m^2\mathsf{Adv}_{\mathcal{A},\Sigma}^{\mathsf{UF-CMA}} + \frac{(q_s+q_r)^2}{2^{\mathcal{K}-1}} + \frac{(m+1){q_s}^2}{2^{\mathcal{K}-1}} + 2\mathsf{Adv}_{\mathcal{A},\mathcal{F}}. \tag{6.11}$$

□

Theorem 6.2. ***(MA Security)*** [62]. If the signature scheme Σ is UF-CMA and the hash function H is a random oracle then our protocol is MA secure and

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MA}} \leq m^2\mathsf{Adv}_{\mathcal{A},\Sigma}^{\mathsf{UF-CMA}} + \frac{(q_s+q_r)^2}{2^{\mathcal{K}}} + \frac{(m+1){q_s}^2}{2^{\mathcal{K}}}.$$

Proof. We prove by a sequence of games. Let $\mathsf{Win}_{\mathsf{i}}^{\mathsf{MA}}$ be the event that the adversary $\mathcal{A}$ wins Game i, $\mathsf{i} = 0, \ldots, 3$.

Let **Game 0** be the original MA security game. By definition, we have:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MA}} = Pr[\mathsf{Win}_0^{\mathsf{MA}}]. \tag{6.12}$$

Let **Game 1** be the same as Game 0, except that the simulation fails if the event Forge defined in Game 1 of Theorem 6.1 occurs:

$$|Pr[\mathsf{Win}_1^{\mathsf{MA}}] - Pr[\mathsf{Win}_0^{\mathsf{MA}}]| \leq Pr[\mathsf{Forge}] \leq m^2\mathsf{Adv}_{\mathcal{A},\Sigma}^{\mathsf{UF-CMA}}. \tag{6.13}$$

Let **Game 2** be the same as Game 1, except that the simulation fails if the event Collision defined in Game 2 of Theorem 6.1 occurs:

$$|Pr[\mathsf{Win}_2^{\mathsf{MA}}] - Pr[\mathsf{Win}_1^{\mathsf{MA}}]| \leq Pr[\mathsf{Collision}] \leq \frac{(q_s + q_r)^2}{2^{\mathcal{K}}}. \tag{6.14}$$

Let **Game 3** be the same as Game 2, except that the simulation fails if the event Repeat defined in Game 3 of Theorem 6.1 occurs:

$$|\mathsf{Pr}[\mathsf{Win}_3^{\mathsf{MA}}] - Pr[\mathsf{Win}_2^{\mathsf{MA}}]| \leq Pr[\mathsf{Repeat}] \leq \frac{(m+1){q_s}^2}{2^{\mathcal{K}}}. \tag{6.15}$$

This last game excludes both forgeries and replay attacks. Therefore, if Game 3 does not aboard, each user U_j, uncorrupted at the time when $\Pi_{U_i}^{s_i}$ accepts, has a corresponding instance $\Pi_{U_j}^{s_j}$ such that $(\mathsf{pid}_{U_j}^{s_j}, \mathsf{sid}_{U_j}^{s_j}) = (\mathsf{pid}_{U_i}^{s_i}, \mathsf{sid}_{U_i}^{s_i})$. According to the MA security definition, $\mathcal{A}$ may win the game only if $\Pi_{U_i}^{s_i}$ and $\Pi_{U_j}^{s_j}$ accept with different keys $k_{U_i}^{s_i} \neq k_{U_j}^{s_j}$, which is impossible. Hence:

$$Pr[\mathsf{Win}_3^{\mathsf{MA}}] = 0 \tag{6.16}$$

By combining (6.12) - (6.16) we get:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MA}} \leq m^2 \mathsf{Adv}_{\mathcal{A},\Sigma}^{\mathsf{UF-CMA}} + \frac{(q_s + q_r)^2}{2^{\mathcal{K}}} + \frac{(m+1){q_s}^2}{2^{\mathcal{K}}}. \tag{6.17}$$

□

Theorem 6.3. ***(Contributiveness)*** [62]. If the trapdoor function $\mathcal{F}$ is secure and the hash function H is a random oracle then our protocol is contributive and

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Con}} \leq \frac{(m+1){q_s}^2}{2^{\mathcal{K}}} + \frac{q_r}{2^{\mathcal{K}}}.$$

Proof. We prove by a sequence of games. Let $\mathsf{Win}_{\mathsf{i}}^{\mathsf{Con}}$ be the event that the adversary $\mathcal{A}$ wins Game $\mathsf{i}, \mathsf{i} = 0 \ldots 2$.

Let **Game 0** be the original contributiveness game. By definition, we have:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Con}} = Pr[\mathsf{Win}_0^{\mathsf{Con}}]. \tag{6.18}$$

Let **Game 1** be the same as Game 0, except that the simulation fails if the event Repeat defined in Game 3 of Theorem 6.1 occurs:

$$|Pr[\mathsf{Win}_1^{\mathsf{Con}}] - Pr[\mathsf{Win}_0^{\mathsf{Con}}]| \leq Pr[\mathsf{Repeat}] \leq \frac{(m+1){q_s}^2}{2^{\mathcal{K}}}. \tag{6.19}$$

Let **Game 2** be the same as Game 1, except that the simulation fails if $\mathcal{A}$ can find a collision for $k = \mathsf{H}(\mathsf{sid}_{U_1}||r_1||r_2||\ldots||r_m)$ on an input r_i, $i = 1, \ldots, m$:

$$|Pr[\mathsf{Win}_2^{\mathsf{Con}}] - Pr[\mathsf{Win}_1^{\mathsf{Con}}]| \leq \frac{q_r}{2^{\mathcal{K}}}. \tag{6.20}$$

If Game 2 does not abort, the output of the random oracle is uniformly distributed. Hence:

$$Pr[\mathsf{Win}_2^{\mathsf{Con}}] = 0. \tag{6.21}$$

By combining (6.18) - (6.21) we get:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Con}} \leq \frac{(m+1){q_s}^2}{2^{\mathcal{K}}} + \frac{q_r}{2^{\mathcal{K}}}. \tag{6.22}$$

□

6.2.2.3 Analysis

Table 6.1 analyses the complexity of the proposed protocol from three different perspectives: storage, computational cost and overall transmission cost. Let l_x be the length (in bits) of x and c_y be the cost to execute y.

First, our proposal is storage efficient: each user maintains his long-lived secret key and a session ephemeral value r_i; in addition, the initiator keeps secret a session trapdoor key. Second, the computational cost is acceptable for a regular party. In case Key Confirmation Phase is performed, extra cost is required. However, we stress that $c_{\mathcal{SS}.\mathsf{Share}}$ and $c_{\mathcal{SS}.\mathsf{Rec}}$ can be neglected as they reduce to a sum modulo a prime when the scheme in Fig.2.3 is used. We also remark that the untraditional perspective on secret m-sharing schemes permits the parties to efficiently recover the key by themselves (with no need to interact with the other parties in Key Computation Phase). Third, the number of exchanged messages during one session of the protocol is significantly lower: $m+1$ broadcast messages, in comparison to almost double (ignoring the high number of unicast messages) for Bresson and Catalano's protocol and almost triple for Cao et al.'s construction.

Our construction introduces different storage and computational costs for the initiator and the rest of the users - a property of GKT rather than GKA protocols. This suggests to introduce an online high performance entity that runs the computation instead of the initiator, while the initiator plays the role of a regular party (after he requests the key establishment). Therefore, the costs of the initiator become lower, while key contributiveness is maintained.

Table 6.2 compares our proposal with the existing work.

First, it achieves better security: as we have already mentioned in the descriptions of the protocols (Subsections 6.2.1.1 and 6.2.1.2), Bresson and Catalano miss the strong corruption (the adversary is not allowed to reveal private internal state information of participants) and Cao et al. miss the contributiveness proof. Our proposal is secure in the GBG model and hence it stands against KCI attacks. We admit that the security proof is performed in the ROM rather than the standard model.

Second, our proposal maintains a constant number of rounds regardless the group size, similar to the existing protocols. We remind that Bresson and Catalano's and Cao et al.'s constructions use the last round for key confirmation, while we obtain this property only by additional computation.

Last, we emphasize two more advantages of our work: the session id is computed at runtime (the environment of the protocol does not generate it in advance) and participants communicate through broadcast channels only (highly appreciated due to the increase of concurrency and the reduction of overhead transmission).

Table 6.1: Complexity Analysis of the Proposed Protocol [62]

	Storage	Computation	Transmission
U_1	$l_{sk} + l_{sk_1} + \mathcal{K}$	$c_{\mathcal{F}.\mathsf{Gen}} + c_{\mathsf{F}} + (m-1)c_{\mathsf{F}^{-1}} + c_{\mathsf{H}} +$ $2c_{\Sigma.\mathsf{Sign}} + (m-1)c_{\Sigma.\mathsf{Verify}} + (m-1)c_{\mathcal{SS}.\mathsf{Share}}$	$l_{\mathcal{U}} + l_{pk} + ml_{\mathsf{F}} +$ $+(m+1)l_{\Sigma.\mathsf{Sign}} +$ $(m-1)\mathcal{K}$
U_i $(i \neq 1)$	$l_{sk_i} + \mathcal{K}$	$c_{\mathsf{F}} + c_{\Sigma.\mathsf{Sign}} + mc_{\Sigma.\mathsf{Verify}} + c_{\mathcal{SS}.\mathsf{Rec}} +$ $(+(m-2)c_{\mathcal{SS}.\mathsf{Share}})$	

Table 6.2: Comparison of the Proposed Protocol to the Existing Work [62]

	No. of Rounds	Transmission Type	Group Type	sid Generation	Security Model
Our Protocol	3	broadcast	static	at runtime	GBG / ROM
Bresson-Catalano [13]	3	unicast, broadcast	static	in advance	BCP / ROM
Cao et al. [20]	3	broadcast	static	in advance	UC / ROM

6.3 Research Contributions

We gave a secure method to build GKA protocols based on secret m-sharing in the GBG model [62] (Subsection 6.2.2). Our protocol achieves better security than the existing work, while it maintains the same number of communication rounds (regardless the number of participants). In addition: broadcast messaging is sufficient, key computation is efficient (in the sense that users recover the key from their own shares) and key confirmation is achieved by default (without the necessity of an extra round of communication).

Conclusions and Further Research

The current thesis focuses on GKE protocols based on secret sharing and the underlying schemes that are used for their construction. We briefly review next our results and point out possible directions of research. Throughout the thesis, **Research Contributions** sections summarize the personal contribution of each chapter and indicate the original papers that introduce the results. We highlight that except the introductory chapters (Part I, Chapter 1 and Part II, Chapter 4), all the others contain personal results. We divide our work into two streams:

- **Cryptanalysis (Chapters 3 and 5)**.

  First, we mount SETUP in SSS that employ enough randomness in order to give the attacker an overwhelming advantage to access the shared secret [55]. Even though similar malicious implementations were introduced for multiple cryptographic primitives (e.g.: key generation, signatures, e-voting schemes), we are the first to consider them for SSS. The technique we propose may successfully embed into classical schemes like Shamir's, which is used as primitive in the design of multiple GKE protocols. Therefore, analyzing the security of these protocols with respect to SETUP against the underlying schemes represent an immediate research topic.

  Second, motivated by the multitude of secret sharing-based GKE protocols with informal security proofs introduced in the last years, we analyze their security [56], [57], [58], [59], [60], [61]. Since no formal security proof is given, vulnerabilities tend to arise natural. We describe six such attacks against four very recent protocols (or their improved version) (2012-2013) and show how to address them. However, the improved versions built to stand against these attacks also lack a formal security proof and hence remain susceptible to similar vulnerabilities. Further research may extend the chains of attacks and countermeasures applied to GKE protocols or reveal vulnerabilities against similar GKE protocols that exist in the literature.

- **Design (Chapters 2 and 6)**.

  First, we define a new VSSS that allows to split a black-and-white secret image into color shares [52], [53]. The proposal reveals no information for at most two participants and permits perfect reconstruction when all parties collaborate. A GKE construction that uses VSSS may be an interesting approach for further research. To the best of our knowledge, none currently exists in the literature. We are aware of the practical limitations of such a proposal, but we can indicate an immediate usage: to set up a visual private common key that may subsequently be used for visual encryption.

Second, we review the notion of *secret m-sharing* and introduce the concept of *perfect secret m-sharing* as a natural extension of perfect sharing [62]. The very few GKE protocols based on secret sharing with formal security proofs that currently exist in the literature motivate our work: we use secret m-sharing (due to the advantages it provides) as primitive to build a secure GKA protocol [62]. The protocol is proved secure in the GBG model, while it maintains the same efficiency in the number of communication rounds as the existing work. To the best of our knowledge, no other secret sharing-based protocol is proved secure in a similar or stronger security model. We consider as topic for further research the improvement of the proposed GKE protocol to stand against EKL, a notion that is not modeled within GBG. Forthcoming work may also enhance the protocol to admit dynamic groups, anonymity, robustness or better efficiency. Defining efficient GKE protocols based on secret sharing that are proved secure in strong security models represents a challenging task.

Other possible research topics include:

- *defining Flexible GKE protocols based on secret sharing.* Flexible GKE protocols were introduced in 2010 by Abdalla et al. [1]: when a group of users compute a common private key, each member also obtains additional information that can later use to derive on-demand independent subgroup secret session keys (shared between subsets of members of the original group) without the necessity of initiating a new session. Although the proposal of Abdalla et al. was proved vulnerable and further improved to stand against insider attacks [21], their idea is of great interest, due to the efficiency it provides.
- *defining Asymmetric GKE protocols (ASGKE) based on secret sharing.* Asymmetric GKE protocols were introduced in 2009 by Wu et al. [76]: a group of users share a common public key, but each member owns a different secret key (which permits him to decrypt ciphertexts encrypted using the common public key or provide a signature verifiable under the public group key). These systems present remarkable properties: escrow monitoring, delegation of duties, backup of cryptographic keys and group signatures. We mention our work on ASGKE, but the construction does not rely on SSS and hence is not strictly related to the current thesis [54].
- *analyzing and improving the existing GKE protocols based on modern primitives or concepts.* Our current work restricts to GKE based on classical secret sharing. Further research may be done in the field of GKE that use underlying lattice-based, bilinear-pairings or even quantum SSS.

Appendix A.
Background on Cryptography

Although we assume that the reader is familiar with the basic cryptographic concepts, we briefly review next some of the notions and assumptions we have used throughout the thesis.

For more details, we invite the reader to address [37] and [46].

A.1 Cryptographic Notions

A.1.1 Negligible Functions

Definition A.1. ***(Negligible Function)***.

A function F *is negligible if for every positive polynomial* f *there exists an* $N \in \mathbb{N}$ *such that for all* $n > N$*, the following holds:*

$$\mathsf{F}(n) < \frac{1}{f(n)}.$$

Informally, a function F is negligible if for all sufficiently large n, F is smaller than any inverse polynomial in n .

A.1.2 Trapdoor Functions and Permutations

Definition A.2. ***(One-Way Function)***.

A function $\mathsf{F} : \{0,1\}^{\mathcal{K}_1} \rightarrow \{0,1\}^{\mathcal{K}_2}$ *is called one-way function if the following conditions hold:*

- *it is easy to compute: there exists an efficient deterministic algorithm s.t. on input* x *outputs* $y = \mathsf{F}(x)$*;*
- *it is hard to invert: for every PPT adversary* $\mathcal{A}$*, the probability to invert* F *is negligible (i.e. the probability that given* y*,* $\mathcal{A}$ *outputs* x *s.t.* $\mathsf{F}(x) = y$ *is negligible).*

Definition A.3. ***(Secure Trapdoor Function)***.

A trapdoor function is a triplet of algorithms $\mathcal{F} = (\mathsf{Gen}, \mathsf{F}, \mathsf{F}^{-1})$*, where:*

1. $\mathsf{Gen}(1^{\mathcal{K}})$ *is a randomized algorithm that on input a security parameter* $\mathcal{K}$ *outputs a public-private key pair* (pk, sk)*;*

2. *without the knowledge of the trapdoor sk,* $\mathsf{F}(pk, \cdot)$ *is a one-way function;*

3. *with the knowledge of the trapdoor sk,* $\mathsf{F}^{-1}(sk, \cdot)$ *is a deterministic function that efficiently inverts* $\mathsf{F}(pk, \cdot)$.

Let $\mathsf{Adv}_{\mathcal{A},\mathcal{F}}$ be the advantage of an adversary $\mathcal{A}$ to invert $\mathsf{F}(pk, \cdot)$ without the knowledge of sk. Then $\mathcal{F}$ is secure if $\mathsf{Adv}_{\mathcal{A},\mathcal{F}}$ is negligible in the security parameter $\mathcal{K}$ for any PPT adversary $\mathcal{A}$.

The similar notions of *one-way permutation* and *secure trapdoor permutation* require that the domain of F equals its codomain. We skip the definitions to avoid repetition.

A.1.3 Public Key Encryption Schemes

Definition A.4. ***(Public Key Encryption Scheme)***.
A public key encryption scheme is a triplet of algorithms $\mathcal{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$*, where:*

1. $\mathsf{Gen}(1^{\mathcal{K}})$ *is a randomized algorithm that on input a security parameter* $\mathcal{K}$ *outputs a public-private key pair* (pk, sk)*;*

2. $\mathsf{Enc}(pk, M)$ *is a randomized encryption algorithm under the public key pk that on input a message* $M \in Sp(M)$ *outputs a ciphertext* $C \in Sp(C)^u$*,* $u \in \mathbb{Z}^*$ *fixed;*

3. $\mathsf{Dec}(sk, C)$ *is a deterministic decryption algorithm under the secret key sk that on input a ciphertext* $C \in Sp(C)^u$ *outputs the plaintext* $M \in Sp(M)$ *or failure.*

The encryption scheme is consistent if $\mathsf{Dec}(sk, \mathsf{Enc}(pk, M)) = M$*, for all public-private key pairs* (pk, sk) *generated by* Gen *and* $M \in Sp(M)$.

We denote by $Sp(M)$ the plaintext space (which may depend on pk), respectively by $Sp(C)^u$ the ciphertext space.

Definition A.5. ***(Ind-CPA Security)***. *Let* $\mathcal{A}$ *be an adversary against the Ind-CPA security of a public key encryption scheme* $\mathcal{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. $\mathsf{Game}^{\mathsf{Ind-CPA}}$ *is defined as follows:*

- *in* Stage 1*, a public-private key pair* (pk, sk) *is generated by running* $\mathsf{Gen}(1^{\mathcal{K}})$ *;*

- *in* Stage 2*,* $\mathcal{A}$ *is given pk and access to* $\mathsf{Enc}(pk, \cdot)$*. At the end of the stage,* $\mathcal{A}$ *outputs two messages of the same length* M_1 *and* M_2*;*

- *in* Stage 3*, a random bit b is chosen, the challenge ciphertext* $C = \mathsf{Enc}(pk, M_b)$ *is computed and given to* $\mathcal{A}$*;*

- *in* Stage 4*,* $\mathcal{A}$ *continues to have access to* $\mathsf{Enc}(pk, \cdot)$*. At the end of the stage,* $\mathcal{A}$ *outputs a bit* b'.

$\mathcal{A}$ *wins* $\mathsf{Game}^{\mathsf{Ind-CPA}}$ *if* $b = b'$*. Let* $\mathsf{Win}^{\mathsf{Ind-CPA}}$ *be the winning probability of* $\mathcal{A}$ *in the Ind-CPA security game. Then the advantage of* $\mathcal{A}$ *in winning the game is:*

$$\mathsf{Adv}^{\mathsf{Ind-CPA}}_{\mathcal{A},\mathcal{PKE}} = |2Pr[\mathsf{Win}^{\mathsf{Ind-CPA}}] - 1|.$$

A public key encryption scheme is called Indistinguishable under a Chosen Plaintext Attack (Ind-CPA) if $\mathsf{Adv}^{\mathsf{Ind-CPA}}_{\mathcal{A},\mathcal{PKE}}$ *is negligible in the security parameter* $\mathcal{K}$ *for any PPT adversary* $\mathcal{A}$.

In case of public key encryption schemes, both the encryption algorithm Enc and the public key pk are public. This implies that: (1) Ind-CPA security is equivalent to *semantic security* and (2) public key encryption must be randomized (i.e. a plaintext encrypts to multiple ciphertexts). As a consequence of randomization, the ciphertext space is always larger than the plaintext space.

A.1.4 Public Key Encryption Schemes from Trapdoor Permutations

Definition A.6. ***(Hard Core Predicate)***.
Let $\mathcal{F} = (\mathsf{Gen}, \mathsf{F}, \mathsf{F}^{-1})$ *be a secure trapdoor permutation. A deterministic PPT algorithm* hc*, which on inputs pk and x, outputs a single bit* $\mathsf{hc}(pk, x)$ *is called hard-core predicate if* $|2Pr[\mathcal{A}(pk, \mathsf{F}(pk, x)) = \mathsf{hc}(pk, x)] - 1|$ *is negligible in the security parameter* $\mathcal{K}$ *for any PPT adversary* $\mathcal{A}$.

Let $\mathcal{F} = (\mathsf{Gen}, \mathsf{F}, \mathsf{F}^{-1})$ be a secure trapdoor permutation and hc an associated hard-core predicate as described before. A public key encryption scheme $\mathcal{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ can be constructed as:

- $\mathsf{Gen}(1^{\mathcal{K}})$ is the key generation algorithm, which runs $\mathcal{F}.\mathsf{Gen}(1^{\mathcal{K}})$ to obtain a public-private key pair (pk, sk);
- $\mathsf{Enc}(pk, M)$ is a randomized encryption algorithm under the public key pk that on input a message $M = (m_1, ..., m_l) \in \{0, 1\}^l$, randomly chooses x_1 in the domain of F, computes $x_i = \mathsf{F}(pk, x_{i-1})$, for all $i = 2, \ldots, l$ and outputs the ciphertext $C = (c_0, c_1, ..., c_l) = (x_{l+1}, \mathsf{hc}(pk, x_1) \oplus m_1, ..., \mathsf{hc}(pk, x_l) \oplus m_l)$;
- $\mathsf{Dec}(sk, C)$ is a deterministic decryption algorithm under the secret key sk that on input a ciphertext C outputs the plaintext $M = (\mathsf{hc}(pk, x_1) \oplus c_1, ..., \mathsf{hc}(pk, x_l) \oplus c_l)$, where $x_i = \mathsf{F}^{-1}(sk, x_{i+1})$, $i = 1, \ldots, l$.

Theorem A.1. *If* $\mathcal{F} = (\mathsf{Gen}, \mathsf{F}, \mathsf{F}^{-1})$ *is a secure trapdoor permutation and* hc *is a corresponding hard-core predicate, then the public key cryptosystem* $\mathcal{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is semantically secure.*

The proof is beyond the purpose of this work. The reader may refer to the bibliography for more details [9], [37], [78].

A.1.5 Digital Signatures

Definition A.7. ***(Digital Signature Scheme)***.
A digital signature scheme is a triplet of algorithms $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$*, where:*

1. $\mathsf{Gen}(1^{\mathcal{K}})$ *is a randomized algorithm that on input a security parameter* $\mathcal{K}$ *outputs a public-private key pair* (pk, sk)*;*

2. $\mathsf{Sign}(sk, M)$ *is a randomized signing algorithm under the private key sk that on input a message* $M \in \{0,1\}^*$ *outputs a valid signature* σ*;*

3. $\mathsf{Verify}(pk, M, \sigma)$ *is a deterministic algorithm that outputs 1 for a valid message-signature pair* (M, σ) *and 0 otherwise.*

For simplicity of exposure, we have used throughout the thesis a slightly different notation: $\Sigma.\mathsf{Sign}_{U_i}(M)$ denotes the signing algorithm performed by the user U_i on the message M and $\Sigma.\mathsf{Verify}_{U_i}(M, \sigma)$ denotes the verification algorithm on a message M and a signature σ that is supposed to originate from U_i, $i = 1, \ldots m$. The signature is certified such that no other party can impersonate U_i by generating a new signature scheme on his behalf and making the others believe that it corresponds to U_i.

Definition A.8. ***(UF-CMA Security)****. Let* $\mathcal{A}$ *be an adversary against the UF-CMA security of a digital signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$*.* $\mathsf{Game}^{\mathsf{UF-CMA}}$ *is defined as follows:*

- *in* Stage 1*, a public-private key pair* (pk, sk) *is generated by running* $\mathsf{Gen}(1^{\mathcal{K}})$ *;*
- *in* Stage 2*,* $\mathcal{A}$ *is given pk and access to* $\mathsf{Sign}(sk, \cdot)$*. At the end of the stage,* $\mathcal{A}$ *outputs a pair* (M, σ)*.*

$\mathcal{A}$ *wins* $\mathsf{Game}^{\mathsf{UF-CMA}}$ *if: (1) no signature on message* M *was requested during the execution and (2)* (M, σ) *represents a valid signature, i.e.* $\mathsf{Verify}(pk, M, \sigma) = 1$*. Let* $\mathsf{Win}^{\mathsf{UF-CMA}}$ *be the winning probability of* $\mathcal{A}$ *in the UF-CMA security game. Then the advantage of* $\mathcal{A}$ *in winning the game is:*

$$\mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathcal{A},\Sigma} = Pr[\mathsf{Win}^{\mathsf{UF-CMA}}].$$

A signature scheme is called UnForgeable under a Chosen Message Attack (UF-CMA) if $\mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathcal{A},\Sigma}$ *is negligible in the security parameter* $\mathcal{K}$ *for any PPT adversary* $\mathcal{A}$*.*

A.1.6 Hash Functions and ROM

Definition A.9. ***(Hash Function)****.*
A function $\mathsf{H} : \{0,1\}^* \leftarrow \{0,1\}^{l(\mathcal{K})}$ *is called hash function if the following conditions hold:*

1. *it compresses: it maps an arbitrary length input* $x \in \{0,1\}^*$ *to an output* $\mathsf{H}(x)$ *of fixed length in the security parameter* $\mathcal{K}$*;*

2. *it is a one-way function.*

Definition A.10. ***(Collision Resistant Hash Function)****.*
Let $\mathcal{A}$ *be an adversary against the collision resistant property of a hash function* H*.* $\mathsf{Game}^{\mathsf{Coll-Res}}$ *is defined as follows:*

- *in* Stage 1*,* $\mathcal{A}$ *is given access to* H*;*
- *in* Stage 2*,* $\mathcal{A}$ *outputs two values* x_1 *and* x_2*.*

$\mathcal{A}$ wins $\mathsf{Game}^{\mathsf{Coll-Res}}$ *if: (1)* $x_1 \neq x_2$ *and (2)* $\mathsf{H}(x_1) = \mathsf{H}(x_2)$. *Let* $\mathsf{Win}^{\mathsf{Coll-Res}}$ *be the winning probability of $\mathcal{A}$ in the Coll-Res game. Then the advantage of $\mathcal{A}$ in winning the game is:*

$$\mathsf{Adv}_{\mathcal{A},\mathsf{H}}^{\mathsf{Coll-Res}} = Pr[\mathsf{Win}^{\mathsf{Coll-Res}}].$$

A hash function is called collision resistant if $\mathsf{Adv}_{\mathcal{A},\mathsf{H}}^{\mathsf{Coll-Res}}$ *is negligible in the security parameter $\mathcal{K}$ for any PPT adversary $\mathcal{A}$.*

Collision-resistance is a strong security concept that implies the weaker notions of *pre-image resistance*[9] (i.e. given $y \in \{0,1\}^{l(\mathcal{K})}$ it is infeasible for a PPT adversary to find $x \in \{0,1\}^*$ s.t. $\mathsf{H}(x) = y$) and *second pre-image resistance* (i.e. given $x \in \{0,1\}^*$ it is infeasible for a PPT adversary to find $x' \in \{0,1\}^*$, $x' \neq x$ s.t. $\mathsf{H}(x) = \mathsf{H}(x')$).

A *random oracle* simulates ideal random functions as black boxes to which all parties (including the adversary) have access by asking *queries*: they give a binary string as input and receive a binary string as output. The box is *consistent* in the sense that for a given input it always return the same output.

A random oracle behaves like a collision resistant hash functions. We skip the details here but invite the reader to address [37], if interested. This approach is used to prove the security of a cryptographic construction within an ideal model (*Random Oracle Model (ROM)*) by replacing the real hash functions with random oracles.

A.2 Cryptographic Assumptions

Definition A.11. ***(Computational Diffie-Hellman(CDH) Assumption)***. *Let G be a multiplicative cyclic group of prime order p, with g as generator. The Computational Diffie-Hellman (CDH) assumption holds if given g, g^a and g^b, any PPT adversary $\mathcal{A}$ has a negligible probability in computing g^{ab}:*

$$Adv_{\mathcal{A}}^{CDH} = Pr[\mathcal{A}(p, g, g^a, g^b) = g^{ab}] \leq negl(\mathcal{K})$$

where $a, b \in \mathbb{Z}_p^*$ are random and $\mathcal{K}$ is the security parameter.

Definition A.12. ***(Decisional Diffie-Hellman(DDH) Assumption)***. *Let G be a multiplicative cyclic group of prime order p, with g as generator. The Decisional Diffie-Hellman (DDH) assumption holds if given g, g^a, g^b and g^c any PPT adversary $\mathcal{A}$ has a negligible probability in distinguishing g^{ab} from g^c:*

$$Adv_{\mathcal{A}}^{DDH} = Pr[\mathcal{A}(p, g, g^a, g^b, g^c) = 1] - Pr[\mathcal{A}(p, g, g^a, g^b, g^{ab}) = 1] \leq negl(\mathcal{K})$$

where $a, b, c \in \mathbb{Z}_p^*$ are random and $\mathcal{K}$ is the security parameter.

Definition A.13. ***(Discrete Logarithm Problem (DLP))***.

Let G be a multiplicative cyclic group of prime order p, with g as generator. The Discrete Logarithm (DLP) is hard in G if given g^a, any PPT adversary $\mathcal{A}$ has a negligible probability in computing a:

[9] A hash function should achieve *pre-image resistance* by definition, since it is a one-way function.

$$Adv_{\mathcal{A}}^{DLP} = Pr[\mathcal{A}(p, g, g^a) = a] \leq negl(\mathcal{K})$$

where $a \in \mathbb{Z}_p^*$ *is random and* $\mathcal{K}$ *is the security parameter.*

Appendix B.
Background on Mathematics

B.1 Entropy

Let X be a random variable with the probability distribution $Pr[X = x_i] = p_i$, $i = 1, \ldots, m$. By definition, X takes values in the finite set $\{x_1, \ldots, x_m\}$ with the corresponding probabilities $\{p_1, \ldots, p_m\}$ such that $0 \leq p_i \leq 1$ and $\sum_{i=1}^{m} p_i = 1$. Let Y and Z be two other random variables set in a similar way.

Definition B.1. ***(Entropy)*** [46]. The *entropy* of X is defined as

$$H[\mathsf{X}] = -\sum_{i=1}^{m} p_i \lg p_i,$$

where $p_i \lg p_i = 0$ if $p_i = 0$.

Informally, the entropy measures the quantity of information revealed by an obervation on a random variable: when entropy is high, few information is revealed, while when entropy is low a lot of information is disclosed; in particular, for $H[X] = 0$, the value of X is known with probability 1.

Definition B.2. ***(Conditional Entropy)*** [46]. The *conditional entropy* of X given by $\mathsf{Y} = y$ is defined as

$$H[\mathsf{X}|\mathsf{Y} = \mathsf{y}] = -\sum_{x} Pr[\mathsf{X} = x|\mathsf{Y} = \mathsf{y}] \lg(Pr[\mathsf{X} = x|\mathsf{Y} = y]),$$

where x ranges over all values of X.
The *conditional entropy* of X given by Y is defined as

$$H[\mathsf{X}|\mathsf{Y}] = \sum_{y} Pr[\mathsf{Y} = y] H[\mathsf{X}|\mathsf{Y} = y],$$

where y ranges over all values of Y.

Informally, the conditional entropy $H[\mathsf{X}|\mathsf{Y}]$ measures the amount of uncertinty about X after Y has been observed. In particular, for $H[\mathsf{X}|\mathsf{Y}] = H[\mathsf{X}]$, an observation on Y reveals nothing about X.

B.2 Difference Lemmas

Lemma B.1. ***(Difference Lemma I)*** [69]. Let X, Y, Z be events defined in some probability distribution s.t. $\mathsf{X} \wedge \neg\mathsf{Z}$ and $\mathsf{Y} \wedge \neg\mathsf{Z}$ are equivalent. Then:

$$|Pr[\mathsf{X}] - Pr[\mathsf{Y}]| \leq Pr[\mathsf{Z}].$$

Proof.

$$|Pr[\mathsf{X}] - Pr[\mathsf{Y}]| = |Pr[\mathsf{X} \wedge \mathsf{Z}] + Pr[\mathsf{X} \wedge \neg\mathsf{Z}] - |Pr[\mathsf{Y} \wedge \mathsf{Z}] - Pr[\mathsf{Y} \wedge \neg\mathsf{Z}]. \tag{B.1}$$

Since $\mathsf{X} \wedge \neg\mathsf{Z}$ and $\mathsf{Y} \wedge \neg\mathsf{Z}$ are equivalent, then $Pr[\mathsf{X} \wedge \neg\mathsf{Z}] = Pr[\mathsf{Y} \wedge \neg\mathsf{Z}]$ and:

$$|Pr[\mathsf{X}] - Pr[\mathsf{Y}]| = |Pr[\mathsf{X} \wedge \mathsf{Z}] - Pr[\mathsf{Y} \wedge \mathsf{Z}]|. \tag{B.2}$$

As both $Pr[\mathsf{X} \wedge \mathsf{Z}]$ and $Pr[\mathsf{Y} \wedge \mathsf{Z}]$ are positive numbers less than or equal to $Pr[\mathsf{Z}]$:

$$|Pr[\mathsf{X}] - Pr[\mathsf{Y}]| \leq Pr[\mathsf{Z}]. \tag{B.3}$$

□

Lemma B.2. ***(Difference Lemma II)*** [44]. Let X, Y, Z be events defined in some probability distribution s.t. $Pr[\mathsf{Y}] = Pr[\mathsf{X}|\mathsf{Z}]$. Then:

$$Pr[\mathsf{X}] - Pr[\mathsf{Y}] \leq Pr[\neg\mathsf{Z}].$$

Proof.

$$Pr[\mathsf{X}] = Pr[\mathsf{X}|\mathsf{Z}]Pr[\mathsf{Z}] + Pr[\mathsf{X}|\neg\mathsf{Z}]Pr[\neg\mathsf{Z}]. \tag{B.4}$$

As $Pr[\mathsf{Y}] = Pr[\mathsf{X}|\mathsf{Z}]$:

$$Pr[\mathsf{X}] = Pr[\mathsf{Y}]Pr[\mathsf{Z}] + Pr[\mathsf{X}|\neg\mathsf{Z}]Pr[\neg\mathsf{Z}]. \tag{B.5}$$

Since all probabilities are less than or equal to 1:

$$Pr[\mathsf{X}] \leq Pr[\mathsf{Y}] + Pr[\neg\mathsf{Z}] \tag{B.6}$$

□

References

[1] Michel Abdalla, Céline Chevalier, Mark Manulis, and David Pointcheval. Flexible group key exchange with on-demand computation of subgroup keys. In *Proceedings of the Third international conference on Cryptology in Africa*, AFRICACRYPT'10, pages 351–368, Berlin, Heidelberg, 2010. Springer-Verlag. pages 98

[2] Yair Amir, Yongdae Kim, Cristina Nita-Rotaru, and Gene Tsudik. On the performance of group key agreement protocols. *ACM Trans. Inf. Syst. Secur.*, 7(3):457–488, August 2004. pages 47

[3] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. Authenticated group key agreement and friends. In *Proceedings of the 5th ACM conference on Computer and communications security*, CCS '98, pages 17–26, New York, USA, 1998. ACM. pages 52, 53

[4] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, CCS '93, pages 62–73, New York, USA, 1993. ACM. pages 52, 79

[5] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, pages 232–249, London, UK, 1994. Springer-Verlag. pages 51, 79

[6] Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In *Proceedings of the 6th IMA International Conference on Cryptography and Coding*, pages 30–45, London, UK, 1997. Springer-Verlag. pages 51, 79

[7] George Blakley. Safeguarding cryptographic keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, pages 313–317, Monval, NJ, USA, 1979. AFIPS Press. pages 21

[8] Rolf Blom. An optimal class of symmetric key generation systems. In *Proc. of the EUROCRYPT'84 workshop on Advances in cryptology: theory and application of cryptographic techniques*, pages 335–338, New York, USA, 1985. Springer-Verlag New York, Inc. pages 49

[9] Manuel Blum and Shafi Goldwasser. An efficient probabilistic public key encryption scheme which hides all partial information. In *Proceedings of CRYPTO'84 on Advances in cryptology*, pages 289–302, New York, USA, 1985. Springer-Verlag New York, Inc. pages 101

[10] Carlo Blundo, Alfredo De Santis, Amir Herzberg, Shay Kutten, Ugo Vaccaro, and Moti Yung. Perfectly-secure key distribution for dynamic conferences. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 471–486, London, UK, 1993. Springer-Verlag. pages 49

[11] Jens-Matthias Bohli, María Isabel González Vasco, and Rainer Steinwandt. Secure group key establishment revisited. *Int. J. Inf. Secur.*, 6(4):243–254, June 2007. pages 80, 90

[12] Colin Boyd. On key agreement and conference key agreement. In *Proceedings of the Second Australasian Conference on Information Security and Privacy*, ACISP '97, pages 294–302, London, UK, 1997. Springer-Verlag. pages 52, 62

[13] Emmanuel Bresson and Dario Catalano. Constant round authenticated group key agreement via distributed computation. In *Proceedings of the 7th International Workshop on Theory and Practice in Public Key Cryptography*, PKC'2004, pages 115–129. Springer, 2004. pages 14, 49, 87, 88, 96

[14] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably authenticated group Diffie-Hellman key exchange - The dynamic case. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '01, pages 290–309, London, UK, 2001. Springer-Verlag. pages 52, 79, 87

[15] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '02, pages 321–336, London, UK, 2002. Springer-Verlag. pages 53, 79

[16] Emmanuel Bresson, Olivier Chevassut, David Pointcheval, and Jean-Jacques Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, CCS '01, pages 255–264, New York, USA, 2001. ACM. pages 79

[17] Emmanuel Bresson and Mark Manulis. Securing group key exchange against strong corruptions. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, ASIACCS '08, pages 249–260, New York, USA, 2008. ACM. pages 51, 52, 53, 79, 80, 83, 85, 86

[18] Mike Burmester. On the risk of opening distributed keys. In *Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '94, pages 308–317, London, UK, 1994. Springer-Verlag. pages 51

[19] Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system (extended abstract). In *Proceedings of the 13th annual international conference on theory and application of cryptographic techniques*, EUROCRYPT'94, pages 275–286. Springer, 1994. pages 51, 53

[20] Chunjie Cao, Chao Yang, Jianfeng Ma, and Sangjae Moon. Constructing UC secure and constant-round group key exchange protocols via secret sharing. *EURASIP J. Wirel. Commun. Netw.*, 2008:4:1–4:9, January 2008. pages 14, 49, 87, 89, 96

[21] Qingfeng Cheng and Chuangui Ma. Security weakness of flexible group key exchange with on-demand computation of subgroup keys. *CoRR*, abs/1008.1221, 2010. pages 98

[22] Yvo Desmedt and Mike Burmester. Towards practical proven secure authenticated key distribution. In *Proceedings of the 1st ACM conference on Computer and communications security*, CCS '93, pages 228–231, New York, USA, 1993. ACM. pages 53

[23] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006. pages 47, 51

[24] Whitfield Diffie, Paul C. Van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Des. Codes Cryptography*, 2(2):107–125, June 1992. pages 51

[25] Ratna Dutta and Rana Barua. Overview of key agreement protocols. Cryptology ePrint Archive, Report 2015/289, 2005. pages 87

[26] Amos Fiat and Moni Naor. Broadcast encryption. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, pages 480–491, London, UK, 1994. Springer-Verlag. pages 49

[27] Marcin Gogolewski, Marek Klonowski, Przemyslaw Kubiak, Miroslaw Kutylowski, Anna Lauks, and Filip Zagórski. Kleptographic attacks on e-voting schemes. In *Proceedings of the 2006 international conference on Emerging Trends in Information and Communication Security*, ETRICS'06, pages 494–508, Berlin, Heidelberg, 2006. Springer-Verlag. pages 35

[28] Zbigniew Golebiewski, Miroslaw Kutylowski, and Filip Zagórski. Stealing secrets with SSL/TLS and SSH - Kleptographic attacks. In *Proceedings of the 5th international conference on Cryptology and Network Security*, CANS'06, pages 191–202, Berlin, Heidelberg, 2006. Springer-Verlag. pages 35

[29] M. Choudary Gorantla, Colin Boyd, and Juan Manuel González Nieto. Modeling key compromise impersonation attacks on group key exchange protocols. In *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography: PKC '09*, pages 105–123, Berlin, Heidelberg, 2009. Springer-Verlag. pages 51, 80, 82, 83, 84, 85, 92

[30] Lien Harn and Changlu Lin. Authenticated group key transfer protocol based on secret sharing. *IEEE Trans. Comput.*, 59(6):842–846, June 2010. pages 13, 49, 54, 55, 72

[31] Dennis Hofheinz, Jörn Müller-Quade, and Rainer Steinwandt. Initiator-resilient universally composable key exchange. In *Proc. European Symp. on Research in Computer Security*, ESORICS'03, pages 61–84. Springer, 2003. pages 90

[32] Chingfang Hsu, Bing Zeng, Qi Cheng, and Guohua Cui. A novel group key transfer protocol. Cryptology ePrint Archive, Report 2012/043, 2012. pages 13, 49, 54, 62, 63

[33] Ryo Ito, Hidenori Kuwakado, and Hatsukazu Tanaka. Image size invariant visual cryptography. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E82-A(10):2172–2177, October 1999. pages 30

[34] Philippe Janson and Gene Tsudik. Research: Secure and minimal protocols for authenticated key distribution. *Comput. Commun.*, 18(9):645–653, September 1995. pages 51

[35] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In *Proceedings of the 4th International Symposium on Algorithmic Number Theory*, ANTS-IV, pages 385–394, London, UK, 2000. Springer-Verlag. pages 47

[36] Ehud D. Karnin, Jonathan W. Greene, and Martin E. Hellman. On secret sharing systems. *IEEE Transactions on Information Theory*, 29:35–41, 1983. pages 13, 25, 26, 27

[37] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007. pages 99, 101, 103

[38] Jonathan Katz and Ji Sun Shin. Modeling insider attacks on group key-exchange protocols. In *Proceedings of the 12th ACM conference on Computer and communications security*, CCS '05, pages 180–189, New York, USA, 2005. ACM. pages 52, 79, 82, 90

[39] Hyun-Jeong Kim, Su-Mi Lee, and Dong Hoon Lee. Constant-round authenticated group key exchange for dynamic groups. In *Proceedings of ASIACRYPT'04*, pages 245–259. Springer, 2004. pages 87

[40] Mijin Kim, Namje Park, and Dongho Won. Cryptanalysis of an authenticated group key transfer protocol based on secret sharing. In *Grid and Pervasive Computing*, pages 761–766. Springer-Verlag, 2013. pages 14, 54, 71

[41] Daniel Kucner and Miroslaw Kutylowski. Stochastic kleptography detection. In *Public-Key Cryptography and Computational Number Theory*, pages 137–149, 2001. pages 35

[42] Daniel Kucner and Mirosław Kutyłowski. How to use un-trusty cryptographic devices. *Tatra Mountains Mathematical Publications*, 29:57–67, 2004. pages 35

[43] Chih-Hung Li and Josef Pieprzyk. Conference key agreement from secret sharing. In *Proceedings of the 4th Australasian Conference on Information Security and Privacy*, ACISP '99, pages 64–76, London, UK, 1999. Springer-Verlag. pages 49, 87

[44] Mark Manulis. *Provably secure group key exchange*. PhD thesis, 2007. pages 46, 47, 51, 52, 80, 81, 82, 83, 84, 85, 86, 90, 106

[45] Mark Manulis. Survey on security requirements and models for group key exchange. Technical Report 2006/02, Horst-Görtz Institute, Network and Data Security Group, January 2008. pages 46, 47, 51, 52, 80

[46] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1996. pages 21, 22, 45, 46, 47, 48, 51, 52, 99, 105

[47] Silvio Micali and Leonid Reyzin. Physically observable cryptography. In *Proceedings of the 1st First Theory of Cryptography Conference*, TCC'04. pages 86

[48] Elsayed Mohamed and Hassan Elkamchouchi. Kleptographic attacks on elliptic curve cryptosystems. *International Journal of Computer Science and Network Security*, 10(6):213–215, 2010. pages 35

[49] Elsayed Mohamed and Hassan Elkamchouchi. Kleptographic attacks on elliptic curve signatures. *International Journal of Computer Science and Network Security*, 10(6):264–267, 2010. pages 35

[50] Junghyun Nam, Moonseong Kim, Juryon Paik, Woongryul Jeon, Byunghee Lee, and Dongho Won. Cryptanalysis of a group key transfer protocol based on secret sharing. In *Proceedings of the Third international conference on Future Generation Information Technology*, FGIT'11, pages 309–315, Berlin, Heidelberg, 2011. Springer-Verlag. pages 13, 54, 56, 57, 72

[51] Moni Naor and Adi Shamir. Visual cryptography. In *Proceedings of the 13th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'94, pages 1–12. Springer, 1994. pages 28

[52] Ruxandra F. Olimid. About a visual secret sharing scheme. In *Proceedings of the 3rd International Conference on Security for Information Technology and Communications*, SECITC'10, pages 13–17, 2010. pages 13, 30, 31, 33, 34, 97

[53] Ruxandra F. Olimid. Python implementation of visual secret sharing schemes. *Journal of Information Systems & Operations Management*, 5(2.1):538–550, December 2011. pages 13, 29, 31, 33, 97

[54] Ruxandra F. Olimid. A new public key cryptosystem with key escrow capabilities. In *Proceedings of the 7th South East European Doctoral Student Conference*, DSC 2012, pages 610–624, 2012. pages 98

[55] Ruxandra F. Olimid. SETUP in secret sharing schemes using random number generators. 2012. Under review. pages 13, 36, 37, 39, 40, 41, 42, 97

[56] Ruxandra F. Olimid. A chain of attacks and countermeasures applied to a group key transfer protocol (abstract). In *(Pre)Proceedings of Western European Workshop on Research in Cryptology*, WEWoRC'13, 2013. pages 14, 54, 72, 74, 75, 76, 77, 78, 97

[57] Ruxandra F. Olimid. Cryptanalysis of a password-based group key exchange protocol using secret sharing. *Appl. Math. Inf. Sci*, 7(4):1585–1590, July 2013. pages 14, 54, 72, 74, 76, 78, 97

[58] Ruxandra F. Olimid. On the (in)security of group key transfer protocols. In *Proceedings of the Romanian Academy, series A*, volume 14, Special Issue RCD'13, pages 378–387, 2013. pages 51, 78, 97

[59] Ruxandra F. Olimid. On the (non)improvement of an authenticated group key transfer protocol based on secret sharing. 2013. Under review. pages 13, 54, 59, 61, 78, 97

[60] Ruxandra F. Olimid. On the security of an authenticated group key transfer protocol based on secret sharing. In *Proceedings of the 2013 international conference on Information and Communication Technology*, ICT-EurAsia'13, pages 399–408, Berlin, Heidelberg, 2013. Springer-Verlag. pages 13, 14, 54, 67, 69, 70, 71, 78, 97

[61] Ruxandra F. Olimid. On the vulnerability of a group key transfer protocol based on secret sharing. 2013. Under review. pages 13, 54, 62, 64, 65, 66, 78, 97

[62] Ruxandra F. Olimid. Provable secure constant-round group key agreement protocol based on secret sharing. In *Proceedings of International Joint Conference SOCO'13 - CISIS'13 - ICEUTE'13*, AISC 239, pages 489–498. Springer, 2013. pages 13, 14, 15, 27, 33, 49, 90, 91, 92, 93, 94, 96, 98

[63] Joseph Pieprzyk and Chih-Hung Li. Multiparty key agreement protocols. *IEE Proceedings-Computers and Digital Techniques*, 147(4):229–236, 2000. pages 48, 49, 71

[64] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978. pages 40, 59

[65] Germán Sáez. Generation of key predistribution schemes using secret sharing schemes. *Discrete Appl. Math.*, 128(1):239–249, May 2003. pages 49

[66] Amitabh Saxena. *New paradigms in group-oriented cryptography: non-interactive key agreement, chain signatures and additive proofs of knowledge.* PhD thesis, 2007. pages 47

[67] Amitabh Saxena and Ben Soh. A new paradigm for group cryptosystems using quick keys. In *Proceedings of the The 11th IEEE International Conference on Networks*, ICON'03, pages 385–389, 2003. pages 45, 47

[68] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979. pages 13, 21, 25, 26

[69] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. revised 2006. http://eprint.iacr.org/; http://www.shoup.net/papers/games.pdf. pages 80, 81, 106

[70] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 443–461, Berlin, Heidelberg, 2009. Springer-Verlag. pages 86

[71] Michael Steiner. *Secure group key agreement*. PhD thesis, 2002. pages 53

[72] Michael Steiner, Gene Tsudik, and Michael Waidner. Cliques: A new approach to group key agreement. In *IEEE International Conference on Distributed Computing Systems*, pages 380–387. IEEE Computer Society Press, 1998. pages 51

[73] Yi Sun, Qiaoyan Wen, Hongxiang Sun, Wenmin Li, Zhengping Jin, and Hua Zhang. An authenticated group key transfer protocol based on secret sharing. *Procedia Engineering*, 29(0):403 – 408, 2012. 2012 International Workshop on Information and Electronics Engineering. pages 13, 27, 49, 54, 64, 68

[74] Web-site. Python Image Library (PIL). Last accessed: July 2013. http://www.pythonware.com/products/pil/. pages 32

[75] Web-site. Python programming language, Official website. Last accessed: July 2013. http://www.python.org/. pages 32

[76] Qianhong Wu, Yi Mu, Willy Susilo, Bo Qin, and Josep Domingo-Ferrer. Asymmetric group key agreement. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 153–170, Berlin, Heidelberg, 2009. Springer-Verlag. pages 98

[77] Yacov Yacobi and Zahava Shmuely. On key distribution systems. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 344–355, New York, USA, 1989. Springer-Verlag New York, Inc. pages 51

[78] Andrew C. Yao. Theory and application of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 80–91, Washington, DC, USA, 1982. IEEE Computer Society. pages 101

[79] Adam Young and Moti Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 89–103, London, UK, 1996. Springer-Verlag. pages 35, 36

[80] Adam Young and Moti Yung. Kleptography: using cryptography against cryptography. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, pages 62–74, Berlin, Heidelberg, 1997. Springer-Verlag. pages 35

[81] Adam Young and Moti Yung. The prevalence of kleptographic attacks on discrete-log based cryptosystems. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '97, pages 264–276, London, UK, 1997. Springer-Verlag. pages 35

[82] Adam Young and Moti Yung. Malicious cryptography: kleptographic aspects. In *Proceedings of the 2005 international conference on Topics in Cryptology*, CT-RSA'05, pages 7–18, Berlin, Heidelberg, 2005. Springer-Verlag. pages 35

[83] Wei Yuan, Liang Hu, Hongtu Li, and Jianfeng Chu. An efficient password-based group key exchange protocol using secret sharing. *Appl. Math. Inf. Sci*, 7(1):145–150, January 2013. pages 14, 49, 54, 59, 72, 73

[84] Wei Yuan, Liang Hu, Hongtu Li, and Jianfeng Chu. Security and improvement of an authenticated group key transfer protocol based on secret sharing. *Appl. Math. Inf. Sci*, 7(5):1943–1949, September 2013. pages 13, 54, 58, 59, 60

[85] Jianjie Zhao, Dawu Gu, and M. Choudary Gorantla. Stronger security model of group key agreement. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 435–440, New York, USA, 2011. ACM. pages 52, 80, 85

Index