Provable encryption schemes for distributed systems

Mohamad El Laz

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Schémas de Chiffrement Prouvables pour les Systèmes Distribués

Mohamad El Laz

INRIA, équipe INDES

Présentée en vue de l'obtention du grade de docteur en informatique **d'**Université Côte d'Azur
Dirigée par : Tamara Rezk, Research Director, *Inria équipe INDES*
Co-encadrée par : Benjamin Grégoire, Senior Researcher, *Inria équipe STAMP*
Soutenue le : 30 Mars 2022

Devant le jury, composé de :
Rapporteurs :
Karthikeyan Bhargavan, Research Director, *Inria Paris*
David Naumann, Professor, *Stevens Institute of Technology*
Examinateurs :
Alejandro Hevia, Professor, *University of Chile*
Bruno Martin, Professor, *Université Côte d'Azur*

Inria
INVENTEURS DU MONDE NUMÉRIQUE

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE STIC

THÈSE

pour obtenir le titre de

Docteur en Informatique

Présentée et soutenue par
Mohamad EL LAZ

PROVABLE ENCRYPTION SCHEMES for DISTRIBUTED SYSTEMS

Thèse dirigée par Tamara Rezk

Devant le jury composé de :

Tamara Rezk Research Director, Inria Sophia-Antipolis	**Directrice de thèse**
Benjamin Grégoire Senior Researcher, Inria Sophia-Antipolis	**Co-encadrant**
Karthikeyan Bhargavan Research Director, Inria Paris	**Rapporteur**
David Naumann Professor, Stevens Institute of Technology	**Rapporteur**
Alejandro Hevia Professor, University of Chile	**Éxaminateur**
Bruno Martin Professor, Université Côte d'Azur	**Éxaminateur**

RESUMÉ

Cette thèse propose l'utilisation de schémas de chiffrement prouvables pour obtenir la sécurité de bout en bout de systèmes distribués. Nous analysons d'abord les conditions suffisantes d'un schéma de cryptage prouvable utilisé dans les systèmes de vote et découvrons plusieurs incohérences dans la satisfaction de ces hypothèses dans les implémentations du schéma de cryptage ElGamal. Nous proposons et comparons différentes méthodes pour obtenir des implémentations indiscernables d'attaques en texte clair choisies. Nous étudions également les schémas de cryptage de diffusion et proposons un nouveau schéma basé sur ElGamal. Nous mettons en œuvre et comparons différents schémas de cryptage par diffusion en fonction de temps d'exécution et d'espace mémoire. De plus, nous considérons des scénarios basés sur des schémas de cryptage de diffusion pour délivrer de manière sécurisée des messages dans des systèmes distribués hiérarchiques. Nous étendons ces scénarios à une architecture plus complexe où des mises à jour logicielles sont nécessaires, en combinant des schémas de cryptage par diffusion et des protocoles d'attestation à distance. Pour exprimer différents niveaux de confidentialité et d'intégrité, nous utilisons des classes de sécurité présentant un ordre entre elles. Notre idée clé est qu'en faisant correspondre des sous-groupes de nœuds de diffusion à des niveaux de sécurité, nous pouvons contrôler que les informations circulent en toute sécurité du serveur vers les nœuds appartenant à des classes de sécurité différentes. Nous le démontrons par le biais de deux systèmes de types et de preuves de solidité concernant une nouvelle propriété de flux d'informations sécurisé pour le code serveur adapté aux notres architectures.

Mots clés: ElGamal, Schémas de chiffrement, Systèmes distribués, Flux d'informations sécurisé, Systèmes de types.

Abstract

This thesis proposes the use of provable encryption schemes to obtain end-to-end security of distributed systems. We first analyze sufficient conditions of a provable encryption scheme used in voting systems and discover several inconsistencies in the satisfaction of these hypotheses in the implementations of the ElGamal encryption scheme. We propose and compare different methods to obtain chosen plaintext attacks indistinguishable implementations. We also study broadcast encryption schemes and propose a new scheme based on ElGamal. We implement and compare different broadcast encryption schemes in means of execution time and memory space. Furthermore, we consider scenarios based on broadcast encryption schemes to securely deliver messages in hierarchical distributed systems. We extend these scenarios to a more complex architecture where software updates are required, combining broadcast encryption schemes and remote attestation protocols. To express different levels of confidentiality and integrity, we use security classes featuring an order among them. Our key insight is that by mapping broadcast subgroups of nodes to security levels, we can control that information securely flows from the server to the nodes belonging to different security classes. We demonstrate this via two type systems and soundness proofs with respect to a new secure information flow property for server code fitting our architectures.

Keywords: ElGamal, Encryption schemes, Distributed systems, Secure information flow, Type system.

ACKNOLEDGEMENTS

There are no appropriate words to express my gratitude and respect for my thesis advisors, Tamara Rezk and Benjamin Grégoire. Tamara has inspired me and helped me realize my journey as a PhD student. She has been a mentor, a friend, a second mother and much more. Without her nothing of this would have been feasible.

My sincere thanks also go to the members of my thesis jury and exam committee: Professors Alejandro Hevia, Bruno Martin, David Naumann, and Karthikeyan Bhargavan. They generously gave their time to offer me helpful comments toward improving my work. In particular, Prof. Alejandro Hevia showed me the potential of applying specific cryptographic primitives to solve information flow problems which helped me develop a broader perspective to my thesis.

There is no way to express how much it meant to me to have been a member of Indes and Marelle Teams. These exceptional colleagues and friends inspired me over the many years: Adam Khayam, Martin, Carlo, Ignacio, Héloise, Maxime, Yoon Seok, Manuel Serrano, Cécile, Sophie, Maxime Dénès, Cyril Cohen, Laurent Théry, Laurence Rideau, Yves Bertot and many others.

I cannot forget friends who went through hard times together, cheered me on, and celebrated each accomplishment: Zak, Ziad, Adam, Tota, Andrea, Pedro. I deeply thank my parents, especially my mother for her endless love, patience and her overseas support. I also thank my girlfriend Deborah for her love, support, and encouragement that helped me get through this agonizing period in the most positive way.

CONTENTS

CHAPTER 1

INTRODUCTION

Whenever we talk about communication, we may think about secret communications or the communication of a secret. Cryptography can ensure the communication of secrets over insecure channels. With the beginning of public-key cryptography due to Diffie and Hellman [Diffie and Hellman, 1976], many cryptographic schemes have been proposed. Their security depends on hard computational problems such as integer factorization and discrete logarithm. Even though they rely on such complex problems, cryptographic schemes may leak information about their encrypted message [Lipton, 1981]. The security of cryptographic schemes was inferred from the absence of known attacks until the work of Golwasser and Micali [Goldwasser and Micali, 1982] in which they introduced the idea of provable security. In particular, they proposed the notion of semantic security also known as polynomial indistinguishability (IND) . Their scheme was the first probabilistic encryption scheme to be provably secure under standard cryptographic mathematical assumptions. A provable security defines the meaning of security in a given condition and proves that a cryptographic scheme achieves it (under certain assumptions). Afterwards, many cryptographic schemes have been proved to be IND secure, particularly IND-CPA and IND-CCA. In this thesis, we investigate the security of encryption schemes. Specifically, we analyze the ElGamal encryption scheme [ElGamal, 1985] and focus on a

group property in which the Decisional Diffie-Hellman (DDH) assumption holds. The ElGamal encryption scheme is know to be IND-CPA under the DDH assumption over safe prime order groups. We analyze 26 libraries that implement ElGamal scheme and verify which libraries does respect the DDH assumption. Throughout the analysis, we point out four different encoding and decoding techniques and compare them. Actually, many cryptographic schemes are based on the DDH assumption to achieve IND-CPA security.

The DDH assumption is also useful for other cryptographic constructions as broadcast encryption schemes. Broadcast encryption schemes [Fiat and Naor, 1993], allow a sender to securely communicate messages or key information with a privileged group of nodes. The first security notion of broadcast encryption schemes required that any coalition of nodes can not learn any secret about the content of the broadcast. In our work, we investigate broadcast encryption schemes and discuss their security aspects. On one hand, we propose a new scheme based on ElGamal and implement it along with other existing schemes. We compare and evaluate the execution time of the key generation, the encryption and the decryption process of these schemes. We also measure the maximum key storage and the ciphertext size for 1 node in different subgroups.

On the other hand, we consider scenarios in which a server maps broadcast subgroups of nodes to levels in information flow security lattices to ensure secure information flow policies. In a confidentiality lattice, data is labeled as *high* (for secret) and *low* (for public), where an attacker is assumed to observe the data labeled as *low* and information can only flow from *low* to *high*. In an integrity lattice, data is labeled as *trusted* and *untrusted*, where an attacker is assumed to control the data labeled as *untrusted* and information from *untrusted* data does not affect *trusted* data (information can flow from *trusted* to *untrusted*). In a confidentiality and integrity lattice, namely a product lattice, data is labeled with a confidentiality and integrity levels. Information can only flow from *public* and *trusted* data to *secret* and *untrusted* data. Information flow policies [Sabelfeld and Myers, 2003, Denning and Denning, 1977] focus on preventing information flow from secret to public sources. To prevent insecure information flow , static information

flow enforcement [Volpano et al., 1996] or dynamic enforcement [Askarov and Sabelfeld, 2009, Bielova and Rezk, 2016] can be employed. In [Volpano et al., 1996], Volpano, Irvine and Smith proved through a type system that information flow policies ensure secure information flow policies.

By mapping broadcast encryption subgroups of nodes to levels in information flow security lattices we study secure information flow in the server code. We first consider a scenario in which a server broadcasts to nodes with static security levels (where we verify the use of the correct encryption keys) then we extend it to a scenario for nodes with dynamic security levels where software updates are needed to verify the correct behavior of nodes (and are encrypted to prevent attacks as the injection of malicious code to compromise the software update image [IETF, 2017]) by relying on broadcast encryption schemes and remote attestation. In the latter, our goal is to allow for automate decision-making and easy key management at the server side while preserving confidentiality and integrity of information. Since broadcast subgroups of nodes are mapped to security classes, this allows the server to control the secure information flow and to record which nodes necessitate software updates. Hence, only node that verify the correct loading of software update (through remote attestation) could move to higher integrity level.

We demonstrate our ideas through two type systems of the server code that ensure the compliance of our scenarios with a formally defined information flow policy. The first type system checks that messages broadcasted to a specific subgroup of nodes are not leaked to nodes in security levels with less privileges. It also checks that cryptographic keys variables and variables that record nodes according to their security class are not maliciously modified or wrongly manipulated. Regarding the second type system, besides checking the aforementioned variables, it also controls that variables containing information of nodes belonging to the same security level are updated only if the remote attestation succeeds. We finally prove the soundness of our type systems with respect to a new secure information flow policy by induction on the height of the typing derivation tree of $\Gamma \vdash p$.

The manuscript is organized as follows:

- In Chapter 2, we give a mathematical overview in which we briefly discuss the group theory, and hard computational problems. We then introduce encryption schemes, particularly ElGamal encryption scheme discussing its *IND-CPA* security and its respective security parameters. In the last part, we briefly discuss the *RSA* scheme, *RSA-OAEP* and the *IND-CCA* security of the *RSA* scheme.

- In Chapter 3, we investigate the security of ElGamal implementations. We manually analyze the source code of 26 libraries that implement the ElGamal encryption scheme and give an overview of our results. Finally, we identify four different message encoding and decoding techniques and we discuss the different designs.

- In Chapter 4, we investigate broadcast encryption schemes. We propose a new scheme base on ElGamal and discuss its security parameters. We then implement and compare three existing broadcast encryption schemes in terms of execution time, key storage, and ciphertext size.

- In Chapter 5, we consider scenarios for broadcasting messages and software updates to a set of nodes in hierarchical distributed systems. The purpose is to allow the server to securely communicate with nodes whilst maintaining the integrity and confidentiality of communications. We build on ordered security classes to map broadcast subgroups of nodes to security levels to control secure information flow between server and nodes in different security classes. We also present two typing systems to control that information between server and nodes flows in a secure way. Finally, we instantiate a theorem to show how the proposed architecture works backed with a soundness proof regarding a new secure information flow property for the server side that suits our architecture.

- In Chapter 6, the conclusion and directions for future work resume the results obtained in this thesis, and point out some future developments.

- Appendix A and Appendix B include the proofs of the theorems discussed in Chapter 5. Appendix C provides the ocaml implementations of the three schemes compared in Chapter 4.

CHAPTER 2

BACKGROUND

This chapter is devoted to recall some basic notions and definitions that will be used in the subsequent chapters of this thesis. We begin by group theory preliminaries in Section 2.1 and its related notations as an introduction for Chapter 3. In Section 2.2, we present the discrete logarithm problems (DLP) on which ElGamal encryption scheme in Chapter 3 is based on, and, the factorization problem on which the RSA scheme relies on. In Section 2.3 we introduce the encryption schemes and discuss their related security problems. In Section 2.4, we focus on ElGamal encryption scheme and its security parameters that will be discussed in details in Chapter 3. Finally, in Section 2.5, we evoke the RSA cryptosystem that will be used in Chapter 4, and discuss some of its security notions.

2.1 Group Theory Preliminaries

In this section we recall some basic definitions and notations concerning group theory that are used throughout Chapter 3.

A group is a set of objects with an operation defined between any two objects in the set and satisfying the following axioms:

Definition 1 (Group)**.** *A group consists of a non-empty set G together with a binary operation $\bullet$, $< G, \bullet >$, such that the following properties hold:*

1. *Closure Axiom:* $\forall a, b \in G$: $a \bullet b \in G$.
2. *Associativity Axiom :* $\forall a, b, c \in G$: $a \bullet (b \bullet c) = (a \bullet b) \bullet c$.
3. *Identity Axiom:* $\exists e \in G$ *such that, for any* $a \in G$, $a \bullet e = e \bullet a = a$. *The element e is called the* ***identity*** *element.*
4. *Inverse Axiom:* $\forall a \in G$, *there exists an element* $a^{-1} \in G$ *such that* $a \bullet a^{-1} = a^{-1} \bullet a = e$.

In the rest of the thesis, we omit the operation $\bullet$ and use G to designate a group $< G, \bullet >$.

Definition 2 (Finite and Infinite Groups)**.** *A group G is said to be finite if the number of elements in the set G is finite, otherwise, the group is infinite. $|G|$ denotes the number of elements in G and is called the order of G.*

Definition 3 (Abelian Group)**.** *A group G is abelian if for all* $a, b \in G$, $(a \bullet b) = (b \bullet a)$.

In other words, an abelian group is a commutative group. In this thesis we only deal with finite abelian groups. This said, all the groups that appear in Chapter 3 are abelian, and we always omit the prefix "abelian".

Example 1 (Additive vs. Multiplicative Groups)**.** *We briefly introduce two examples to highlight the difference between an additive group and a multiplicative group:*

1. *The set of integers $\mathbb{Z}$ a group under addition $+$, namely $(\mathbb{Z}, +)$, where $e = 0$ and $a^{-1} = -a$. This is an* ***additive*** *and* ***infinite*** *group. The same applies to the set of rational numbers $\mathbb{Q}$, the set of real numbers $\mathbb{R}$ and the set of complex numbers $\mathbb{C}$.*
2. *Non-zero elements of $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ under multiplication "$\cdot$", are groups with $e = 1$ and a^{-1} being the multiplicative inverse. Such groups are denoted as $\mathbb{Q}^*$, $\mathbb{R}^*$ and $\mathbb{C}^*$, namely $(\mathbb{Q}^*, \cdot)$, $(\mathbb{C}^*, \cdot)$ and $(\mathbb{R}^*, \cdot)$, respectively. Such groups are called* ***multiplicative*** *groups and are* ***infinite****.*

In the rest of the dissertation, we only rely on multiplicative groups and the multiplicative notation "$\bullet$" for the operations of the groups.

In what follows, we introduce the notion of a subgroup, order of a group and Langrange's theorem. We rely on these definitions to achieve security (this will be discussed in detail in Section 2.4 of this chapter, and in Chapter 3).

Definition 4 (Subgroup)**.** *A subgroup of a finite group G is a non-empty subset H of G which is itself a group under the same operation as that of G. We write $H \subseteq G$ to denote that H is a subgroup of G, and $H \subset G$ to denote that H is a proper subgroup of G (i.e., $H \neq G$).*

Definition 5 (Order of a Group)**.** *The number of elements in a finite group G is called the order of G and is denoted by $\#G$.*

Example 2 (Subgroups and Order of a Group)**.** *We introduce some examples on subgroups and orders of groups:*

1. *The additive group $(\mathbb{Z}_{10}, + \ (mod\ 10)$), where $\mathbb{Z}_{10} = \{0,1,2,3,4,5,6,7,8,9\}$, has the following subgroups: $(\{0\}, +)$, $(\{0,5\}, +)$, $(\{0,2,4,6,8\}, +)$ and $(\mathbb{Z}_{10}, +)$.*

2. *The multiplicative group $(\mathbb{Z}^*_{10}, \bullet \ (mod\ 11)$), where $\mathbb{Z}^*_{11} = \{1,2,3,4,5,6,7,8,9,10\}$, has the following subgroups: $(\{1,3,4,5\}, \bullet)$ and $(\mathbb{Z}^*_{11}, \bullet)$.*

3. *Under multiplication, $\mathbb{Q}^* \subseteq \mathbb{R}^* \subseteq \mathbb{C}^*$, where $\mathbb{Q}^*$ is the group of non-zero rational numbers, $\mathbb{R}^*$ the group of non-zero real numbers and $\mathbb{C}^*$ the group of non-zero complex numbers.*

4. *$\#\mathbb{Z}_n = n$.*

Definition 6 (Order of Group Element)**.** *Let G be a multiplicative group with an element $a \in G$, and let e be the identity. The order of the element a is the least positive integer $i \in \mathbb{N}$ satisfying $a^i = e$, and is denoted by $ord(a)$. If such an integer i does not exist, then a is called an element of infinite order.*

The following theorem is important for cryptography as it establishes that the order of a subgroup divides the order of its group. Such property is important as we show in Subsection 2.4.1 for ElGamal encryption scheme.

Theorem 1 (Lagrange's Theorem)**.** *If H is a subgroup of the finite group G, then $\#H|\#G$.*

Corollary 1 (Lagrange)**.** *Let G be a finite group and $a \in G$ be any element. Then $ord(a)|\#G$.*

Proof. For any $a \in G$, if $a = e$ then $ord(a) = 1$ and so $ord(a)|\#G$ is a trivial case. Let $a \neq e$. Since G is finite, we have $1 < ord(a) < \infty$. Elements

$$a, a^2, ..., a^{ord(a)} = e \tag{2.1}$$

are necessarily distinct. Suppose that they were not distinct, then $a^r = a^s$ for some non-negative integers r and s satisfying $1 \leq r < s \leq ord(a)$. By applying the *inverse axiom* of $(a^r)^{-1}$ to both sides, we will have: $a^{r-s} = e$ where $0 < s - r < ord(a)$. This contradicts Definition 6 of $ord(a)$ being the least positive integer satisfying $a^{ord(a)} = e$. It is easy to check that the $ord(a)$ of elements in 2.1 form a subgroup of G. By Langrange's theorem, $ord(a)|\#G$. ■

Corollary 1 is a direct application of Lagrange's Theorem and provides a relationship between the order of a group and the orders of elements in the group. Such relationship is important for public-key cryptosystems such as the security ElGamal encryption scheme as we show in Chapter 3.

Another important notion we use in Chapter 3 is the notion of cyclic groups and group generators.

Definition 7 (Cyclic Group, Group Generator)**.** *A group G is said to be cyclic if there exists an element $g \in G$ such that for any $b \in G$, there exists an integer $i \geq 0$ such that $b = g^i$. The element g is then called a generator of G. And G is called the group generated by g, written as $G =< g >$.*

In Figure 2.1, we show an example of a generator of the multiplicative cyclic group $\mathbb{Z}_{11}^*$. $\mathbb{Z}_{11}^* = \{1,2,3,4,5,6,7,8,9,10\}$ is a group under multiplication modulo 11 and the element 2 is a generator and provides a cyclic view for $\mathbb{Z}_{11}^*$.

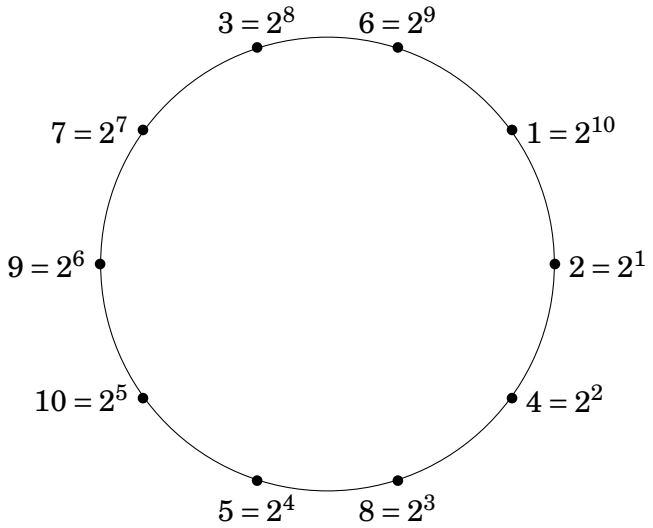

Figure 2.1: Generator of multiplicative group $\mathbb{Z}_{11}^*$

We use the following corollary to show what elements of a cyclic group can be a generator of the group:

Corollary 2. *A prime-order group is cyclic, and any non-identity element in the group is a generator.*

Proof. Let G be a group of prime order p. Let $g \in G$ be any non-identity element. From Corollary 1, $ord(g)|p$. Since $g \neq e$ and $ord(g) \neq 1$, then it has to be the case that $ord(g) = p$. Therefore, $< g >= G$, i.e. g is a generator of G. ■

2.2 Hard Computational Problems

In this section, we present two well-known hard computational problems [Martin, 2004]. In Subsection 2.2.1, we introduce the *discrete logarithm problem* and its related problems on which the ElGamal cryptosystem (See Algorithm 5) in Chapter 3 is based on. In Subsection 2.2.2, we introduce the *factorization problem* used in Chapter 4.

2.2.1 Discrete Logarithm Problems

This subsection is devoted to the discrete logarithm problem (*DLP*), an assumption used by cryptographic protocols such as ElGamal encryption scheme [ElGamal, 1985]. Consider a finite cyclic group G and a generator g. The assumption states that, given $h \in G$, it is computationally unfeasible for an adversary to find an integer x such that $h = g^x$. In what follows, we concisely list the definition of the discrete logarithm and some related problems. In what follows, we use q to denote a prime number, and $\mathbb{Z}_q$ to denote $< \mathbb{Z}_q, \bullet \ (mod\ q) >$.

Definition 8. *(Discrete Logarithm Problem (DLP)) Given $g^x \in G$, where G is a cyclic group of prime order q, g the generator of G, and $x \in \mathbb{Z}_q$, compute x.*

Definition 9. *(Computational Diffie-Hellman (CDH)) Given $g^x \in G$ and $g^y \in G$, where G is a cyclic group of prime order q, g the generator of G, and $x, y \in \mathbb{Z}_q$, compute $g^{xy} \in G$.*

Definition 10. *(Decisional Diffie-Hellman (DDH)) Given two distributions D_1 and D_2, where $D_1 = (g^x, g^y, g^{xy})$, $D_2 = (g^x, g^y, g^z)$ and x, y, z are randomly distributed in $\mathbb{Z}_q$. Distinguish D_1 from D_2.*

If the discrete logarithm problem is easy to solve, then the *DHP* is also easy to solve, and thus, the cryptographic protocol that relies on such assumption is considered insecure. Therefore, we are interested in finding difficult instances of the *DLP*. For instance, the hardness of the discrete logarithm problem depends on the representation of the group G. Note that $DDH \leq_p CDH \leq_p DL$ [Boneh, 1998] where $\leq_p$ indicates polynomial time reductions (See Definition 11, Algorithm 1 and 2).

The discrete logarithm problem (*DLP*) is thought to be mathematically difficult, at least when implemented in cryptographically strong groups of large order (e.g., when q is of size 2048 bits or more). In 2015, a group of researchers published a paper [Adrian et al., 2015] in which they publicly reported a security vulnerability in TLS, called Logjam, that allows users to downgrade connections to “export-grade” Diffie-Hellman key exchange (based on the difficulty of solving the discrete logarithm problem) that ranges from 512

to 1024-bit keys. The researchers used the number field sieve discrete log algorithm to compute arbitrary discrete logs in a 512-bit group. While solving the discrete logarithm problem for 2048-bit prime is believed to be beyond anyone's reach, the authors of the paper estimated the feasibility of an attack against 1024-bit prime to be at least within the range of nation-state attackers such as NSA.

Definition 11. *(Polynomial Time Reduction) Let X and Y be two computational problems. Then X is said to* ***polytime reduce*** *to Y, written $X \leq_p Y$ if*

- *There is an algorithm which solves X using an algorithm which solves Y.*
- *This algorithm runs in polynomial time if the algorithm for Y does.*

Algorithm Reducing *CDH* to *DLP*

1: Given g^x and g^y, find $g^{x \cdot y}$.
2: Use an oracle to solve *DLP* by computing $y = DLP(g, g^y)$.
3: Compute $(g^x)^y = g^{x \cdot y}$.
4: Then **CDH** is no harder than **DLP**: $CDH \leq_p DLP$.

Algorithm Reducing *DDH* to *CDH*

1: Given g^x, g^y and g^z, determine if $g^z = g^{x \cdot y}$.
2: Use an oracle to solve *CDH* by computing $g^{x \cdot y} = CDH(g, g^x, g^y)$.
3: Check whether $g^{x \cdot y} = g^z$.
4: Then **DDH** is no harder than **CDH**: $DDH \leq_p CDH$.

2.2.2 Factorization Problem

In this subsection, we define the *integer factorization problem* on which the *RSA* cryptosystem [Rivest et al., 1978] relies. This subsection gives an overview on the complexity problem that stands behind *RSA* and discuss a related problem, namely the Deciding Composite Residuosity introduced by [Paillier, 1999].

The integer factorization problem states that it is easy to find the product of the multiplication of two factors, but the inverse operation is difficult.

The integer factorization problem was used by Rivest et. al. in [Rivest et al., 1978] to build *RSA*. In what follows we define the *integer factorization problem*, then the *RSA* modulus and the safe *RSA* modulus used in the factorization problem.

Definition 12. *(*RSA *Modulus) An* RSA *modulus is an integer $N = p \cdot q$, where p and q are two distinct prime numbers.*

Definition 13. *(Euler Phi Function $\phi(N)$) $\phi(N)$ is the number of non-negative integers less than N that are relatively prime to N.*

Definition 14. *(e^{th} Root Problem) Let N be an* RSA *modulus, $y \in \mathbb{Z}_N^*$, and $e \geq 3$ a prime integer with $gcd(e, \phi(N)) = 1$. The e^{th} root problem is then to find $x \in \mathbb{Z}_N^*$, such that $y = x^e \ mod \ N$.*

The difficulty of the *RSA* problem depends on the difficulty of the *integer factorization problem*.

Definition 15. *(Integer factorization problem) Given a positive* RSA *modulus N, compute its decomposition into prime numbers $N = \prod {p_i}^{e_i}$ (unique up to reordering).*

2.3 Encryption Schemes

In this section, we give an overview on encryption schemes and discuss their security aspects. An encryption scheme consists of four algorithms: a setup, a key generator, that generates a pair of keys (encryption and decryption key), a probabilistic encryption algorithm that converts plaintexts into ciphertexts, and a decryption algorithm that transforms ciphertexts into plaintexts, using the adequate keys. When encryption schemes have only one private key for encryption and decryption, they are called *symmetric* key encryption schemes. In 1970 Diffie and Hellman [Diffie and Hellman, 1976] introduced public key encryption schemes, also know as *asymmetric* key encryption schemes, where the encryption key and the decryption key are different.

For what concerns Chapter 3, we rely on *asymmetric* key encryption schemes.

An encryption scheme is a set of algorithms defined as follows:

Definition 16. *(Encryption Scheme)*

- *Setup: Receiving η as input, the algorithm initializes the parameters needed by the scheme.*
- *Key generation: Given a security parameter η, the key generation algorithm $\mathbb{KG}(\eta)$ returns an encryption and decryption key pair (pk, sk);*
- *Encryption: Given an encryption key pk and a plaintext m, the encryption algorithm $\mathbb{E}(pk, m)$ computes a ciphertext corresponding to the encryption of m under pk;*
- *Decryption: Given a decryption key sk and a ciphertext c, the decryption algorithm $\mathbb{D}(sk, c)$ returns the plaintext corresponding to the decryption of c, if it is a valid ciphertext.*

The key generation and encryption algorithms may be probabilistic, while the decryption algorithm is always deterministic. For an encryption scheme to be correct, it is required that decryption be the inverse of the encryption: for every pair of keys (pk, sk) that can be output by the key generation algorithm, and every plaintext m, it must be the case that $\mathbb{D}(sk, \mathbb{E}(pk, m)) = m$.

2.3.1 Security Definitions of Encryption Schemes

An essential condition for an encryption scheme is the difficulty of retrieving an encrypted plaintext without the knowledge of the decryption key. Such condition may be weak in some applications since partial information about the plaintext could endanger the security of an entire scheme. Therefore, it must be unfeasible to learn anything about the plaintext from the ciphertext, following the principle *"whatever an eavesdropper can compute about the clear text given the ciphertext, he can also compute without the ciphertext"* [Goldwasser and Micali, 1982]. Schemes achieving this requirement, such as the Goldwasser-Micali scheme [Goldwasser and Micali, 1982], are called semantically secure.

An encryption scheme is said to be secure if the success probability of an adversary trying to break the scheme is insignificant. This notion is achieved by negligible functions.

Definition 17 (Negligible Function)**.** *A function* $v : \mathbb{N} \to \mathbb{R}$ *is said to be negligible if it decreases asymptotically faster than the inverse of any polynomial:*

$$\forall c \in \mathbb{N}, \exists n_c \in \mathbb{N} \quad s.t. \quad \forall n \in \mathbb{N}, n \geq n_c \Rightarrow |v(n)| < n^{-c}$$

A scheme is considered to be secure if it cannot be broken in polynomial time (with respect to the predefined security parameter). The idea of negligible probability encompasses this exact notion.

To satisfy the aforementioned definition, an encryption scheme must necessarily be probabilistic, otherwise an adversary could trivially detect to which message corresponds the challenge ciphertext by simply encrypting one of the messages it has chosen and comparing the resulting ciphertext with the challenge ciphertext. The reason for this choice is because negligible probability of success stays negligible after even a polynomially many attempts to break the system.

2.3.2 Adversarial Goals and Capacities

In terms of security for cryptosystems, there are different adversarial goals and capacities. Such goals and capacities must be well defined. In what follows, we review the adversarial goals and capacities against cryptographic schemes.

Adversarial goals. For semantic security, the adversary has three main goals: One-wayness, indistinguishability and malleability.

ONE-WAYNESS. A most important security notion for an encryption scheme is to achieve the property of one-wayness: an attacker should not be able to recover the plaintext matching a given ciphertext. However, this is a weak notion of security as unveiling almost all the plaintext is unsuccessful according to this definition. More formally, for any adversary, succeeding in inverting the effects of the encryption on a ciphertext c should occur with negligible probability.

INDISTINGUISHABILITY. An adversary should not learn any information about a plaintext given its encryption. An encryption scheme is said to be semantically secure or indistinguishable if no probabilistic polynomial time algorithm can break it.

MALLEABILITY. The goal of the adversary (observing the ciphertext c corresponding to the plaintext m), is to obtain a valid ciphertext c' of m' related to m.

Adversarial capacities. Adversarial capacities are defined via encryption and decryption oracles which are functions able to encrypt and decrypt even if the adversary does not have secret keys. Such oracles encrypt and/or decrypt arbitrary plaintexts and/or ciphertexts at the adversary's demand. The security of a cryptosystem depends on the adversary capacities. An adversary able to access an encryption oracle is model as *Chosen-plaintext attacks* (*CPA*), decryption oracle *Chosen-ciphertext attacks* (*CCA*) and *Adaptive hosen-ciphertext attacks* (*CCA2*).

CPA. A *chosen-plaintext attack* is an attack in which an adversary with access to an encryption oracle, can choose an arbitrary number of plaintexts to be encrypted to obtain their corresponding ciphertexts. The adversary send m_1, m_2, ... , m_n and receives $Enc(m_1)$, $Enc(m_2)$, ..., $Enc(m_n)$ from the oracle. In Figure 2.2, an adversary A queries the encryption oracle E by sending two plaintexts m_0 and m_1 and getting the ciphertexts c of one of the two messages randomly encrypted from E by picking a random bit $b \in \{0,1\}$. The adversary's strategy is to guess b and thus, guess to which message m_0 or m_1 corresponds the ciphertext c received from E.

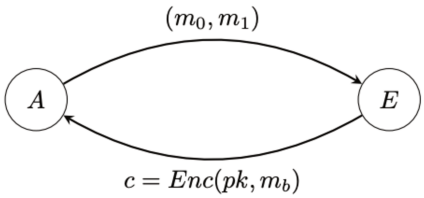

Figure 2.2: *CPA* attacks

CCA. The adversary in the *non-adaptive chosen ciphertext attack* have access to an encryption oracle and a decryption oracle. He chooses an arbitrary number of ciphertexts to obtain the corresponding plaintexts. However, upon the adversary receipt of the target ciphertext from E, the decryption oracle is no longer available. The adversary A queries the decryption oracle D by sending different ciphertexts c_i and getting the corresponding plaintexts m_i. The final goal of A is again to guess to which message m_0 or m_1 corresponds the ciphertext c received from E.

CCA2. In the *adaptive chosen-ciphertext attack*, the adversary attempts to gain partial information by making queries to a decryption oracle based on the results of previous decryptions. In this case, contrarily to *CCA*, the adversary can also query D even after receiving c from E. As shown in Figure 2.3, the decryption oracle is available forever, but A cannot query the decryption oracle with the ciphertexts that he receives from the encryption oracle. In fact, E hold a log of all transmitted c, therefore, D checks if the received c_i does not belong to log in order to decrypt it. In this case too, the goal of A is again to guess to which message m_0 or m_1 corresponds the ciphertext c received from E.

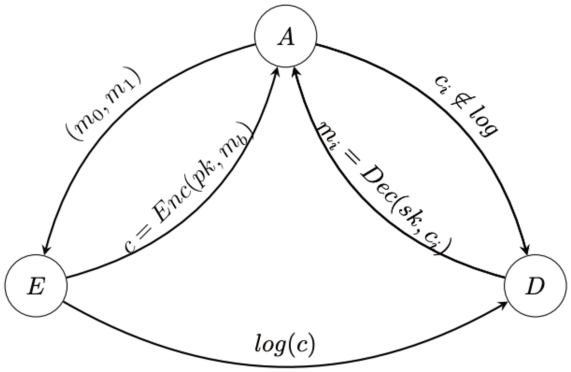

Figure 2.3: *CCA2* attacks

To summarize, the aforementioned attacks can be seen as follows:

- In *CPA*, an adversary possesses an encryption box.

- In *CCA*, an adversary possesses a conditional decryption box. The box expires before the target ciphertext is given to the adversary.

- In *CCA2*, an adversary possesses a decryption box as long as he does not feed the target ciphertext to the box.

2.3.3 Indistinguishability Against Attacks

Indistinguishibality against attacks is a fundamental security property of several cryptosystems. Indistinguishability means that an adversary is unable to distinguish between two ciphertexts. Such property under *chosen-plaintext attacks* is a essential necessity for *asymmetric* key encryption schemes and corresponds to the property of semantic security. In addition, some schemes are also distinguishable against *chosen-ciphertext attacks*.

A cryptosystem enjoys the indistinguishability property if there exists no adversary, given a message encryption randomly chosen between two elements, able to identify which message has been encrypted with a probability better than blind guessing. Such definition suggests that a scheme is secure as long as the adversary cannot learn any information about the original message.

In what follows, we present indistinguishability against *CPA* and *CCA*.

Indistinguishability against *chosen-plaintext attacks* (*IND-CPA*). Indistinguishability against *chosen-plaintext attacks* (*IND-CPA*) for *asymmetric* key encryption schemes is defined by a game between a polynomial time adversary and a challenger. The adversary selects two plaintexts of his choice and sends them to the challenger, who randomly selects one of the two plaintexts, encrypts it and sends the challenge ciphertext back to the adversary. The goal of the adversary is to find out which of the two plaintexts has been encrypted by the challenger.

Definition 18 (*IND-CPA*)**.** *An encryption scheme is said to be* IND-CPA *secure if the advantage of any efficient adversary is a negligible function of the security parameter, i.e., the adversary cannot do much better than a blind guess.*

Algorithm *IND-CPA* Game

1: The challenger generates an encryption key pk and a decryption key sk. Only the encryption key is public.
2: The adversary outputs a pair of messages (m_0, m_1) of equal length.
3: The challenger randomly selects a bit $b \in \{0,1\}$, and send the challenge ciphertext $c = Enc(pk, m_b)$.
4: The adversary outputs his guess b' of the value of b.
5: If $b = b'$, the adversary wins the game.

$$|Pr_{IND-CPA}[b = b'] - \frac{1}{2}| \quad \text{is negligible in the security parameter.}$$

Even though the adversary have knowledge about m_0, m_1 and pk, the encryption algorithm being probabilistic, means that the encryption of m_b is one of several valid ciphertexts. This prevents the adversary from learning some information by encrypting m_0 and m_1 to compare the result and therefore binds the advantage of the adversary.

Indistinguishability against *chosen-ciphertext attacks* (*IND-CCA*). Indistinguishability against *chosen-ciphertext attacks* (*IND-CCA*) is similar to *IND-CPA*, with the only difference that an adversary possesses access to a decryption oracle (instead of an encryption oracle in *IND-CPA*) to decrypt ciphertexts chosen by the adversary. The adversary selects two plaintexts of his choice and sends them to the challenger, who randomly selects one of the two plaintexts, encrypts it and sends the challenge ciphertext back to the adversary. The goal of the adversary is to find out which of the two plaintexts has been encrypted by the challenger. Note that the adversary **cannot** query the decryption oracle to decrypt ciphertexts after receiving the target ciphertext (in the case of *IND-CCA*) nor to decrypt the received ciphertext by the challenger (in the case of *IND-CCA2*).

Algorithm *IND-CCA* Game

1: The challenger generates an encryption key pk and a decryption key sk. Only the encryption key is public.
2: The adversary outputs a pair of messages (m_0, m_1) of equal length.
3: The challenger randomly selects a bit $b \in \{0,1\}$, and send the challenge ciphertext $c = Enc(pk, m_b)$.
4: The adversary outputs his guess b' of the value of b.
5: If $b = b'$, the adversary wins the game.

Definition 19 (*IND-CCA*)**.** *An encryption scheme is said to be* IND-CCA *secure if the advantage of any efficient adversary is a negligible function of the security parameter, i.e., the adversary cannot do much better than a blind guess.*

$$|Pr_{IND-CCA}[b = b'] - \frac{1}{2}| \quad \text{is negligible in the security parameter.}$$

2.4 ElGamal Encryption Scheme

ElGamal [ElGamal, 1985] is an *asymmetric* key encryption scheme, it enjoys homomorphic properties that are fundamental for the electronic voting systems. ElGamal scheme (see Algorithm 5) consists of three algorithms: *key generation(η)*, η being a security parameter, *encryption* $E(m, pk)$ with m being a plaintext and pk a public key, and *decryption* $D(c, sk)$ where c is a ciphertext and sk is a private key.

Algorithm ElGamal Scheme

1: Setup: Let G be a cyclic group of prime order q and g a generator.
2: Key Generation: Pick randomly a secret key $x \in \mathbb{Z}_q$, then compute $y = g^x$ to obtain the public key.
3: Encryption: Let $m \in G$ and $r \in \mathbb{Z}_q$ randomly selected. The resulting ciphertext is $c = (u, v) = (g^r, m.y^r)$.
4: Decryption: To recover the plaintext, one computes $m = v \cdot u^{-x}$.

2.4.1 Security of ElGamal Parameters

The ElGamal encryption scheme is known to be *IND-CPA* secure under the Decisional Diffie-Hellman assumption with the key length as its security parameter. A key point for the security of the ElGamal encryption scheme resides in the group $\mathbb{G}$ and its order [Boneh et al., 2000]. One should start by generating a pair of keys (public and private), then map the message into a group where the Decisional Diffie-Hellman assumption holds. Hence, the difficulty consists in finding an efficient invertible group encoding procedure so that one can recover the original message when decrypting. The ElGamal cryptosystem operates in a finite cyclic group, which by convention is written multiplicatively. For the sake of simplicity, we will restrict our discussion to the group of integers

from {1} to $\{p-1\}$ under multiplication mod p for some prime p, commonly denoted $\mathbb{Z}_p^*$ and subgroups of $\mathbb{Z}_p^*$ of prime order. Moreover, we will also use $|g|$ to denote the order of an element g in $\mathbb{Z}_p^*$ and $< g >$ to denote the cyclic subgroup of $\mathbb{Z}_p^*$ generated by g. Unless otherwise noted, assume multiplications and exponentiations involving elements of $\mathbb{Z}_p^*$ are done mod p. As if all subgroups of a cyclic group are cyclic and if $G = \langle g \rangle$ is cyclic, then for every divisor d of $|G|$ there exists exactly one subgroup of order d which may be generated [Rotman, 1999]. One may rely on this property to form a unique subgroup of quadratic residues elements. To achieve this goal, the idea is to use a Sophie Germain prime [Pollard, 1978]: it is a safe prime p of the form $2q+1$ where q is also prime. Safe primes of that form are important for modulo groups as they guarantee the existence of a subgroup of prime order. For ElGamal, using a safe prime p, where the order is $p-1=2q$ permits to form a subgroup of prime order q that forms the message space we need in order to encrypt messages. One may take advantage of the Lagrange Theorem (Theorem 1) [Pollard, 1978] that States that in a finite group G, the order of any subgroup H divides the order of the group, to conclude that the prime order subgroup has no subgroups being prime. Finally, the message space must be restrained to this prime order subgroup.

Quadratic Residues. To make the ElGamal cryptosystem IND-CPA secure, the Decisional Diffie-Hellman assumption must be respected. As a matter of fact, one needs to find which type of groups satisfies the underlying assumption. A good technique is to restrict the messages to form the subgroup of prime order q of quadratic residues. In what follows we will introduce further explanations and examples to better understand the role of quadratic residues for the security of the ElGamal encryption scheme.

Definition 20 (Quadratic Residue)**.** *An element $a \in \mathbb{Z}_n^*$ is said to be a **Quadratic Residue** modulo n if there exists $x \in \mathbb{Z}_n^*$, such that $x^2 \equiv a (mod\ n)$. Every such x is called a square root of a modulo n. If no such x exists, then a is called a **Quadratic Non-Residue** modulo n. We denote the set of all quadratic residues modulo n by QR_n and the set of all quadratic non-residues by QNR_n*

Quadratic residues are exactly those elements which can be written of the form g^i where i is even: $\{g^2, ..., g^{p-1}\}$ are distinct quadratic residues, while $\{g, gg^2, ..., gg^{p-1}\}$ are quadratic non-residues. There exists an efficient algorithm based on Euler's criterion for deciding quadratic residuosity in $\mathbb{Z}_p^*$, with p prime:

Definition 21 (Euler's Criterion)**.** *Let p be an odd prime. Then $a \in \mathbb{Z}_p^*$ is a quadratic residue modulo p iff $a^{\frac{p-1}{2}} \equiv 1$ (mod p).*

Therefore, by restricting all the messages to be Quadratic residues in a safe prime group, a polynomial time adversary cannot distinguish between elements as all the elements are then quadratic residues. Hereinafter, we will be using the Legendre symbol based on Euler's criterion for its convenient notation that reports the quadratic residuosity of a mod p.

Definition 22 (Legendre symbol)**.** *Let p be an odd prime , a an integer, such that $gcd(a, p) = 1$. The **Legendre symbol** (LS) is defined to be*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if} \quad a \in QR_p \\ -1 & \text{if} \quad a \in QNR_p \end{cases}$$

Example 3. *Let us compute the set of quadratic residues modulo* 11 *(QR_{11}), using **Legendre symbol** :*

Elements	LS	QR
1	1	✓
2	-1	✗
3	1	✓
4	1	✓
5	1	✓
6	-1	✗
7	-1	✗
8	-1	✗
9	1	✓
10	-1	✗

2.5 RSA Scheme

The *RSA* cryptosystem [Rivest et al., 1978] is an *asymmetric* key encryption scheme introduced by Rivest, Shamir, and Adleman. It is one of the most extensively used public key cryptosystems in the last century. It relies on the computational difficulty of factoring large integers (Subsection 2.2.2) , in particular the factorization of large composite prime number. However, in recent years, many cryptographic protocols have replaced *RSA* by elliptic curves because of faster computation.

2.5.1 The RSA Encryption

The *RSA* encryption consists of three algorithms: an *RSA* public key/private *key generation*, an *encryption* algorithm and a *decryption* algorithm.

Key Generation. An *RSA* key generation algorithm consists of the following steps described in Algorithm 6.

Algorithm Key Generation KG_{RSA}

1: Choose a pair of large random primes, namely p and q. Both p and q are kept secret.
2: Compute the public *RSA* modulus $N = p \cdot q$.
3: Select a large integer e, a public odd exponent where $3 < e < N-1$, prime to $\phi(N)$.
4: Compute the private exponent $d \equiv e^{-1} \ mod \ \phi(N)$.
5: Output the public key (N, e) and the private key (N, d).

Encryption. An *RSA* encryption operation is the exponentiation to the e^{th} power modulo N of a message $m \in N$:

$$c = ENC_{RSA}(m, e) = m^e \ mod \ N.$$

Where m is the message, c is the resulting ciphertext and $ENC_{RSA}(m, e)$ is the encryption algorithm. The message m is encrypted using the public exponent e.

Decryption. An RSA decryption operation is the exponentiation to the d^{th} power modulo N of a ciphertext c:

$$m = DEC_{RSA}(c,d) = c^d \ mod \ N.$$

Where $DEC_{RSA}(c,d)$ is the decryption algorithm. Note that the correctness of the *RSA* cryptosystem relies on the fact that the exponentiation to the d^{th} power modulo N, is the inverse of the exponentiation to the e^{th} power. In fact, to recover a message m, one should calculate: $m \equiv c^d \ mod \ N \equiv (m^e)^d \ mod \ N$, where $e \cdot d \equiv 1 \ mod \ N$.

2.5.2 The Security of RSA

The *RSA* encryption is deterministic, meaning that the same ciphertext is produced for the same key pair and plaintext. Being deterministic makes the scheme vulnerable to chosen-ciphertext attacks, where an adversary with access to a random oracle, can obtain the decryption of ciphertexts of its choice. To make *RSA* semantically secure against chosen-ciphertext attacks, padding is added to the ciphertext.

Typically, the padding function used is *OAEP* [Bellare and Rogaway, 1994], resulting in *RSA-OAEP*, for which there exists, in the random oracle model, a loose reduction of the *RSA* problem towards *IND-CCA* attacks [Bellare and Rogaway, 1994, Shoup, 2000, Barthe et al., 2011a]. *OAEP*, namely Optimal Asymmetric Encryption Padding, is a technique for converting the *RSA* trapdoor permutation into a chosen-ciphertext secure system in the random oracle model. Moreover, *OAEP* relies on a pseudo-random number generator for ensuring the indistinguishability of ciphertexts by making the encryption algorithm probabilistic, instead of deterministic as in the original *RSA* scheme.

The *RSA-OAEP* cryptosystem consists of triple $(KG_{RSA}, ENC_{RSA}, DEC_{RSA})$ obtained by a trapdoor permutation *RSA* on $\{0.1\}^k$ and two hash functions G and H:

$$G : \{0,1\}^{k_0} \rightarrow \{0,1\}^{k-k_0}$$
$$H : \{0,1\}^{k-k_0} \rightarrow \{0,1\}^{k_0}$$

Where KG_{RSA} is the key generation function of *RSA*, ENC_{RSA}, the encryption function of *RSA*, and, DEC_{RSA}, the decryption function of *RSA*.

The Optimal Asymmetric Encryption Padding (*OAEP*) scheme applied to *RSA* consists of three algorithms:

Algorithm *RSA-OAEP*

1: Key Generation: The key generation algorithm specifies an instance of the function KG_{RSA} that generates the public key e and the private key d.
2: Encryption: Given a message $m \in \{0,1\}^n$, and a random value $r \leftarrow^R \{0,1\}^{k_0}$, the encryption algorithm computes $s = (m||0^{k_1}) \oplus G(r)$ and $t = r \oplus H(s)$, and outputs the ciphertext $c = ENC_{RSA}(e, s||t)$.
3: Decryption: Using the private key, the decryption algorithm extracts $(s||t) = DEC_{RSA}(d, c)$, then $r = t \oplus H(s)$ and finally $M = s \oplus G(r)$. If $[M]_{k_1} = 0^{k_1}$, the algorithm returns $[M]^n$, otherwise it returns "Reject".

In their paper [Bellare and Rogaway, 1994], Bellare et. al. proved that the *OAEP* encryption scheme is semantically secure and weakly plaintext-aware, provided that f is a one-way trapdoor function. While Shoup [Shoup, 2000] showed that it was absurd to extend the results obtained in [Bellare and Rogaway, 1994] to obtain adaptive chosen-ciphertext security exclusively under the one-wayness of the permutation.

RSA-OAEP was proved to be *IND-CCA* by Fujisaki et. al. [Fujisaki et al., 2004], and machine-checked proved by Barthe et. al. [Barthe et al., 2011b].

CHAPTER 3

SECURITY ANALYSIS OF ELGAMAL IMPLEMENTATIONS

Throughout the last century, especially with the beginning of public key cryptography due to Diffie-Hellman [Diffie and Hellman, 1976], many cryptographic schemes have been proposed. Their security depends on hard computational problems such as integer factorization and discrete logarithm. In fact, it is thought that a cryptographic scheme is secure if it resists cryptographic attacks over a long period of time. On one hand, since certain schemes may take several years before being widely studied in depth, they become vulnerable as time passes. On the other hand, a cryptographic scheme is a provable one, if it resists cryptographic attacks relying on mathematical hypothesis.

Being easily adaptable to many kinds of cryptographic groups, the ElGamal encryption scheme enjoys homomorphic properties while remaining semantically secure [Goldwasser and Micali, 1982], provided that the Decisional Diffie-Hellman (*DDH*) assumption holds on the chosen group. While the homomorphic property forbids resistance against chosen ciphertext attacks, it is very convenient for voting systems [Cortier et al., 2015].

The ElGamal encryption scheme [ElGamal, 1985] (See Section 2.4 of Chapter 2) is the most extensively used homomorphic encryption scheme for voting systems (see also Paillier [Paillier, 1999]). Moreover, ElGamal is the only homomorphic encryption scheme implemented by default in many hardware security modules [Volkamer, 2009, Orr and

Liam, 2016].

In order to be provably secure, the ElGamal encryption needs to be implemented on top of a group verifying the Decisional Diffie-Hellman (*DDH*) assumption. Since this assumption does not hold for all groups, one may have to wrap an encoding and a decoding phase to ElGamal to be able to have a generic encryption scheme. In this chapter, our main goal is to study ElGamal encryption scheme libraries to identify which implementations respect the *DDH* assumption.

We manually analyze the source code of 26 libraries that implement the ElGamal encryption scheme in the wild. We focus our analysis on understanding whether the *DDH* assumption is respected in these implementations, ensuring a secure scheme in which no information about the original message could be leaked. The *DDH* assumption is crucial for the security of ElGamal because it ensures indistinguishability under chosen-plaintext attacks (*IND-CPA*). Without a group satisfying the *DDH* assumption, encryption mechanisms may leak one bit of information about the plaintext and endanger the security of the electoral system, thus threatening the privacy in an election. For instance, when considering an approval ballot with a *yes or no* vote, leaking one bit of information signifies a full leakage of the vote. One way to comply with the *DDH* assumption is by using groups of prime order. In particular, when adopting safe primes, one can ensure the existence of a *large* prime order subgroup [Milne, 2011] and restrict messages to belong to this subgroup. Mapping plaintexts into subgroups is called message encoding. Such encoding necessitates to be efficient and precisely invertible to allow decoding after the decryption.

In Table 3.1, we give an overview of our results : out of 26 analyzed libraries, 20 are wrongly implemented because they do not respect the conditions to achieve *IND-CPA* security under the *DDH* assumption. This means that encryptions using ElGamal from any of these 20 libraries may leak one bit of information (see Section 3.1).

From the 6 libraries which respect the *DDH* assumption, we also study and compare various encoding and decoding techniques. We identify four different message encoding and decoding techniques summarized in Table 3.2. Finally, in Section 3.3, we discuss

the different designs and conclude which implementation is more efficient for voting systems.

In summary, our contributions in Chapter 3 are:

- We analyze 26 libraries implementing ElGamal encryption, and point out which ones are not satisfying the *DDH* assumption and hence are not semantically secure.
- We identify and compare 4 different message encoding and decoding techniques that comply with the *DDH* assumption.

We refer the reader to Chapter 2 in which we present various definitions and notions used in this chapter.

3.1 Breaking ElGamal

We consider a prime p and a generator $g \in \mathbb{Z}_p^*$. Given a public key g^x, the ElGamal encryption scheme encrypts a message $m \in \mathbb{Z}_p$ by computing $(g^r, m.g^{xr})$, with r chosen randomly in $\mathbb{Z}_p^*$. Using the private key x, decryption can be done by first computing (g^{xr}) and then dividing to retrieve m. The cryptosystem is not semantically secure when g is a generator of $\mathbb{Z}_p^*$, as information about the plaintext is leaked (See Section 2.4.1 of Chapter 2). Specifically, the Legendre symbol of $(m.g^{xr})$ uncovers the Legendre symbol of the message m. In order to prove that ElGamal is *IND-CPA* under the *DDH* assumption, one may choose g to be the generator of the group in which the *DDH* assumption holds and restrict the message space to this group. This way, the system is semantically secure as given (g^x) and (g^r), the secret value (g^{xr}) cannot be distinguished from a random element in the group. Therefore, $(m.g^{xr})$ cannot be distinguished from a random element and an attacker cannot learn any information about the original message. In what follows, we will show an example on how to break the *DDH* assumption and ElGamal encryption scheme for $\mathbb{Z}_p^*$ groups.

Example 4 (Breaking the *DDH* Assumption)**.** *In order to break the* DDH *assumption, one should be able to distinguish between two distributions having elements randomly*

distributed in a group. Let $p = 2q + 1 = 11$ *be the group on which we want to perform the attack (to ease the comprehension of the example, we consider small parameters size and not secure). Being prime,* p *has* $\{p-1\}$ *elements, half being quadratic residues (QR) and the other half being quadratic non-residues (QNR). Given a tuple* (g^x, g^y, g^c) *it is hard to decide whether* $c = xy$ *or* $c = z$*, where* z *is randomly generated. By Legendre Symbol, we can check the quadratic residuosity of the elements* g^x*,* g^y *and* g^c*:* $\left(\frac{x}{p}\right) = 1$ *if* x *is a square (i.e., quadratic residue), then* $\left(\frac{g^x}{p}\right) = 1$ *if* x *is even. Therefore,* g^x *is quadratic residue if* x *is a quadratic residue too. In addition, if* x *or* y *are even, then* xy *is even and* g^{xy} *is quadratic residue. Taking advantage of all the before-mentioned notions, we will give a detailed example on how to break the* DDH *assumption. A tuple is a valid* DDH *tuple if* $xy \equiv c \bmod p$ *and can be written as* (g^x, g^y, g^{xy})*.*

Let $p = 11$*, the challenge is to distinguish whether* $(4,5,9)$ *is* DDH_0 *or* DDH_1 *in* G_{11}*. For Euler's criterion,* $a \in \mathbb{Z}_p^*$ *is a QR iff* $a^{\frac{p-1}{2}} \equiv 1$ *(mod p).*

$$4^5 \equiv 1 mod 11 \qquad 5^5 \equiv 1 mod 11 \qquad 9^5 \equiv 1 mod 11$$

By testing the quadratic residuosity of the three elements, one can notice that all of them are quadratic residues. In this case, we cannot distinguish between xy *and* z*. Since the multiplication between two elements that are quadratic residues, results in an quadratic residue element, it might be that the third element of the tuple belongs to* DDH_0 *or* DDH_1*. Consequently, we are not able to break the* DDH *assumption.*

In this second example, the challenge is to distinguish whether $(4,5,8)$ *is* (g^x, g^y, g^{xy}) *or* (g^x, g^y, g^z) *in* G_{11}*.*

$$4^5 \equiv 1 mod 11 \qquad 5^5 \equiv 1 mod 11 \qquad 8^5 \not\equiv 1 mod 11$$

One can notice that the third element of the tuple is not a quadratic residue. Being both quadratic residues, the multiplication between $g^x = 4$ *and* $g^y = 5$ *must result in an quadratic residue element. However,* $g^c = 8$ *results being a non quadratic residue element. In this case, we are able to distinguish between* xy *and* z *by ensuring that* $(4,5,8)$ *is* DDH_1 *and break the* DDH *assumption as a consequence. We emphasize on the importance of choosing a safe prime group* p*, to be able to work in the subgroup of prime order* q *by*

restricting the elements to form the subgroup of quadratic residues. This guarantees the difficulty of such attacks on the DDH *assumption.*

Example 5. ***(Breaking the ElGamal Scheme using a*** QR ***generator)*** *Being in a group in which the* DDH *does not hold may leak information about the original plaintext. An attacker able to calculate the quadratic residuosity of an encrypted message, can learn one bit of information about the original message. An attacker can check whether the encryption of a message is a* QR *or not and therefore deduce whether the original message is a* QR *too. In fact, by taking* $g^{xr} \in QR$ *and encrypting the message as* $\mathscr{E}(m,k) = (g^r, m.g^{xr})$*, one can notice that if* $m.g^{xr} \in QR$ *then* $m \in QR$*.*
Consider the group G_{11}*, let* $x = 4 \in sk$*,* $r = 2 \in rnd$*,* $g = 3 \in QR$ *and* $y = g^x = 4 \in pk$*.*
In the above example, the group G_{11} *has* $\{1,3,4,5,9\}$ *as a subgroup of quadratic residues. Actually, encrypting a message* $m \in QR$ *with a public key* $pk \in QR$*, results always in an encryption* $\mathscr{E}(m,k) \in QR$*. Thus, by using* G_{11} *and taking messages belonging to its entire message space, one endangers the security of the scheme as he allows* QR *and* QNR *messages to be encrypted. An attacker able to calculate the quadratic residuosity of an element could learn one bit of information about the original message by performing the following attack:*

(a) $g \in QR$*, then* (g^{xr}) *is* QR*.*

(b) Check if $\left(\frac{\mathscr{E}(m,k)}{p}\right)$ *is* QR *or not.*

(c) If $\left(\frac{\mathscr{E}(m,k)}{p}\right) \in QR$*, the attacker can learn that the message* $m \in QR$*, as the multiplication between* QR *elements result always in a* QR *element.*

(d) If $\left(\frac{\mathscr{E}(m,k)}{p}\right) \in QNR$*, then the attacker can learn that the message* $m \in QNR$*.*

Let us check $\mathscr{E}(\underline{1},5) = (9,\underline{5})$*:*

(a) $\left(\frac{3}{11}\right) = 1 \Rightarrow g \in QR$*,* $\left(\frac{3^8}{11}\right) = 1 \Rightarrow (g^{xr}) \in QR$*.*

(b) $\left(\frac{5}{11}\right) = 1 \Rightarrow \mathscr{E}(1,5) \in QR$*.*

(c) Being $m.g^{xr} \in QR$ *leaks the quadratic residuosity of m. In fact* $\left(\frac{1}{11}\right) = 1 \Rightarrow m \in QR$.

The previous example could also be adopted for QNR *messages. In summary, the feasibility of calculating the quadratic residuosity of an element in a modulo prime groups, may leak information about the original message on top of groups that do not respect* DDH.

3.2 Study of Libraries

We focus our study on manually checking whether the underlying groups in ElGamal implementations satisfy the Decisional Diffie-Hellman assumption. A summary of our results regarding 26 analyzed libraries can be found in Table 3.1.

We split our work into three tasks to analyze the implementations. We first verify that the implementations use safe primes, next we check if they adopt quadratic residue generators to generate subgroups in which the *DDH* assumption holds. Finally, we analyze the message encoding techniques deployed to map the messages into the before-mentioned subgroups. This study led us to notice that a large number of the considered libraries are not *IND-CPA* secure as the encryption may leak at least one bit of information on the plaintext: 20 libraries do not respect the *DDH* assumption. Moreover, among these 20 libraries, 10 do not use a safe prime. In what follows, we describe in details the problems found in the investigated libraries using the following classification:

(A) *Libraries that do not respect the* DDH *assumption.* There are 20 libraries in this category. In this class we further classify the libraries that do not respect the *DDH* assumption due to 3 different concerns: libraries that do not deploy a safe prime, libraries that do not adopt a quadratic residue generator, and libraries that do not use a correct message encoding technique to map the messages into the intended subgroup.

(B) *Libraries that do respect the* DDH *assumption.* There are 6 libraries in this category. However, they do not all use the same encoding technique. Thus, we describe

in detail the 4 different encoding techniques, discussing their advantages and drawbacks.

Libraries' Relevance. We provide a brief description on the relevance of some of the chosen libraries for analysis. Belenios [Belenios, 2016] has been deployed on an online platform and it is used in more than 200 elections, both in academia and in education. Similarly, Helios [Helios, 2008] is used for the election of the International Association for Cryptographic Research members board, the ACM general elections, the election of UCLouvain president and for other student elections. On the other hand, the Estonian voting system [Estonia, 2017] has been used for the European Parliament elections, local government council elections and the election of the president of the Republic. Swisspost [Swisspost, 2018] then offers voting system services for cantons and municipalities in Switzerland, while Verificatum [Verificatum, 2017] is used in binding elections, student body elections and intra-party elections.

3.2.1 Libraries That Do Not Respect *DDH*

In this section, we will present the implementations that do not respect the conditions to achieve an *IND-CPA* secure ElGamal scheme. Working on groups modulo p, a secure ElGamal scheme has to adopt safe primes, a quadratic residue generator, and a message encoding technique to map the messages into a subgroup that respects the Decisional Diffie-Hellman assumption. Twenty out of twenty six libraries violate one or more of the before-mentioned conditions: all the libraries in this section do not employ message encoding techniques.

Lack of Safe Prime. In this category, 10 libraries [Diaz, 2017, Alves, 2015, Sidorov, 2016, Lee, 2017, Wang, 2017, Pankratiew, 2018, Pellegrini, 2017, Musat, 2017, Libgcrypt, 2013, Moscow, 2019a] do not use safe primes. Instead, these implementations focus on generating large numbers and checking their primality. This method does not guarantee the generation of a safe prime. In fact, a safe prime of the form $p = 2q+1$ where q is also a large prime, is essential as it forms a large prime subgroup of order q. Conversely, using an arbitrary large prime results in a $p-1$ group order, that can be decomposed into small

Library	Safe prime	*QR* Generator	Encoding	Decoding	*DDH*
Belenios [Belenios, 2016]	✓	✓	T3	T3	**Yes**
Botan [Botan, 2018]	✓	✓	✗	✗	**No**
Cryptology [Nasr, 2015]	✓	✗	✗	✗	**No**
Elgamal-api [Diaz, 2017]	✗	✗	✗	✗	**No**
ElGamal-cipher.py [Alves, 2015]	✗	✗	✗	✗	**No**
ElGamalExt [Sidorov, 2016]	✗	✗	✗	✗	**No**
ElGamal.h [Lee, 2017]	✗	✗	✗	✗	**No**
ElGamal.hs [Ridhuan, 2016]	✓	✗	T3	T3	**No**
Elgamal.hpp [Wang, 2017]	✗	✗	✗	✗	**No**
Elgamal.java [Pankratiew, 2018]	✗	✗	✗	✗	**No**
Elgamal-lib.c [Pellegrini, 2017]	✗	✗	✗	✗	**No**
Elgamal.py [Musat, 2017]	✗	✗	✗	✗	**No**
Elgamal.py [Riddle, 2014]	✓	✓	✗	✗	**No**
Elgamir [Elgamir, 2016]	✓	✗	✗	✗	**No**
Estonia [Estonia, 2017]	✓	✓	T1	T1	**Yes**
Helios [Helios, 2008]	✓	✓	T2	T2	**Yes**
Libgcrypt [Libgcrypt, 2013]	✗	✗	✗	✗	**No**
Microsoft [Microsoft, 2019]	✓	✓	T3	T3	**Yes**
Moscow 07-19 [Moscow, 2019a]	✗	✗	✗	✗	**No**
Moscow 09-19 [Moscow, 2019b]	✓	✓	T4	T4	**Yes**
Norway [Norvegia, 2017]	✓	✓	✗	✗	**No**
PyCrypto [Pycrypto, 2012]	✓	✗	✗	✗	**No**
PyCryptodome [Pycryptodome, 2018]	✓	✓	✗	✗	**No**
RSA-ElGamal [Ioannou, 2014]	✓	✗	✗	✗	**No**
Swisspost [Swisspost, 2018]	✓	✓	✗	✗	**No**
Verificatum [Verificatum, 2017]	✓	✓	T1	T1	**Yes**

Table 3.1: An overview on the 26 analyzed libraries (where T1, T2, T3 and T4 are listed in Table 3.2) .

prime order subgroups. Hence, small prime order subgroups are exposed to subgroup attacks in which the discrete logarithm is easy to compute by using the Pohlig-Hellman algorithm [Pohlig and Hellman, 1978] or the Pollard's rho algorithm [Pollard, 1978].

Lack of QR Generators. In this category, we discuss the libraries that do not use quadratic residue generators. Among the 20 libraries that do not respect *DDH*, 5 libraries [Nasr, 2015, Ridhuan, 2016, Elgamir, 2016, Pycrypto, 2012, Ioannou, 2014] use a safe prime $p = 2q + 1$ but do not choose a quadratic residue generator. Using a safe prime without a quadratic residue generator does not guarantee a subgroup of prime order q in which the *DDH* assumption stands. In what follows we show an example on the

feasibility of learning information about the plaintext when we do not employ a quadratic residue generator.

Example 6. *(**Breaking ElGamal without a QR generator**) To break ElGamal without a QR generator, it is sufficient to break the underlying assumption. The* DDH *assumption does not hold in $\mathbb{Z}_p^*$ if g is a generator of $\mathbb{Z}_p^*$. This is because the Legendre symbol of g^a reveals whether a is even or odd. Given (g^a, g^b, g^{ab}), one can compute the LS and compare the least significant bit of a, b and ab, which allows to distinguish between g^{ab} and a random element group. Having a distinguisher against* DDH *means having a distinguisher against ElGamal and therefore break ElGamal.*

Lack of Encoding. We point out the relevance of message encoding mechanisms as a crucial requirement in ElGamal scheme. Despite generating a safe prime and choosing a quadratic residue generator, 5 libraries [Botan, 2018, Riddle, 2014, Norvegia, 2017, Pycryptodome, 2018,Swisspost, 2018] do not use a message encoding to map the messages into a valid subgroup. However, by adopting the standard encryption scheme of ElGamal, the message space is not restricted to the expected subgroup. This imply that even in the presence of a safe prime and consequently the presence of a subgroup of prime order generated by a quadratic residue generator, the message to encrypt is not mapped into the designed subgroup. Hence, an attacker can gain knowledge about the original message and expose the entire scheme to total insecurity. To better understand the importance of using a message encoding method, we display an attack on how to break a scheme that does not map messages into the intended subgroup (see example in Section 3.1).

3.2.2 Libraries That Do Respect *DDH*

In this section, we will present the implementations that respect the *DDH* assumption and therefore implement an *IND-CPA* secure ElGamal scheme. As mentioned in the previous section, a well implemented library should adopt a safe prime, a quadratic residue generator, and a message encoding technique. Only 6 out of the 26 analyzed

libraries [Belenios, 2016, Estonia, 2017, Helios, 2008, Microsoft, 2019, Moscow, 2019b, Verificatum, 2017] respect all of the previously mentioned conditions. In the following paragraphs, we will discuss in detail the message encoding techniques implemented in these libraries. In particular, we can categorize four different techniques.

3.2.2.1 Limited Message Space and q-exponentiations

The Estonian and Verificatum In this paragraph, we will present two libraries that implement ElGamal using the same technique: the Estonian voting system and Verificatum. The Estonian government relies on Internet voting in a significant way for national elections. While the Estonian voting system [Estonia, 2017] is implemented in Java, Verificatum [Verificatum, 2017], which implements provably secure cryptography libraries for electronic voting systems, is implemented in JavaScript. To comply with the *IND-CPA* security of ElGamal, these two implementations adopt a safe prime, and generate the subgroup of prime order in which the *DDH* assumption holds. Both implementations allow messages m to be any integer from $[1, p-1]$. Hence, before encrypting, one checks if m is a QR by checking $m^q \equiv 1 \bmod p$:

Proof. Euler's criterion states that $a \in \mathbb{Z}_p^*$ is a quadratic residue modulo p iff $a^{\frac{p-1}{2}} \equiv 1$ (mod p). Being $q = \frac{p-1}{2}$, then $a^{\frac{p-1}{2}} = a^q \equiv 1$ (mod p). ■

```
switch (legendre(m)) {
case 1:
    return new ModPGroupElement(this, m);
case −1:
    throw new IllegalArgumentException("Can not encode as quadratic residue");
     }
// Negate if not a quadratic residue.
var value = new LargeInteger(bytesToUse);
return new ModPGroupElement(this, value);
```

Listing 3.1: Limited space and q-exponentiations ElGamal encoding [Verificatum, 2017].

If the equivalence is confirmed then m is encrypted as a QR, else the message is rejected and an error is raised (see Listing 3.5). In fact, these two implementations take as an input messages in $\mathbb{Z}_p^*$ and rejects half of the messages that are not QR instead of encoding them (also see Helios in Subsection 3.2.2.2). Besides that, using q-exponentiations to encrypt messages can be optimized as we will explain in the next section.

3.2.2.2 Encoding with q-exponentiations

Helios Helios [Helios, 2008], is a known library implemented in Python. It is used for voting systems and can be manipulated to meet the needs of the users. Helios, is vulnerable to ballot stuffing as a dishonest bulletin board could add ballots without anyone noticing [Belenios, 2016]. Being *IND-CPA* secure, ElGamal in Helios is implemented by generating a safe prime $p = 2q + 1$ where p and q are both large primes. It then selects a generator g of the subgroup of prime order q. Before encrypting a message m, a mapping to the prime order subgroup is done. As the implementation allows the message m to be any integer from $[0, p-1]$, one computes $m_0 = m + 1$ (to avoid picking $m = 0$) and checks whether m_0 is a QR. If ${m_0}^q \equiv 1 \bmod p$ then it outputs m_0 which belongs to the QR elements (as in Subsection 3.2.2.1), else, it outputs $-m_0$. Please note that $-m_0 \bmod p$ belongs also to the QR elements. Being a safe prime p and $p \equiv 3 \bmod 4$, ensures the fact that 1 is a square element and -1 is a non-square element. This is essential in turning a non-square element m into a square element $-m$. After decryption, one obtains m and checks if $m \leq q$. If $m \leq q$ then $m_0 = m$ otherwise $m_0 = -m$. To recover the message, one computes $m = m_0 - 1$ (see Listing 3.6).

Proof. This is because $x^2 \equiv (-x)^2 \bmod p$. So the squares of the first half of the nonzero numbers mod p give a complete list of the nonzero quadratic residues mod p. If p is an odd prime, the residue classes of $\{1^2, 2^2, ..., (\frac{p-1}{2})^2\}$ are distinct and give a complete list of the quadratic residues modulo p. So there are $\frac{p-1}{2}$ residues and $\frac{p-1}{2}$ non-residues. This

gives a complete list as x^2 and $(p-x)^2$ are equivalent mod p:

$$\begin{aligned} x^2 \equiv y^2 \, mod \, p &\Leftrightarrow p|x^2 - y^2 \\ &\Leftrightarrow p|(x-y)(x+y) \\ &\Leftrightarrow p|(x-y) \, or \, p|(x+y) \end{aligned}$$

which is impossible if x and y are two different members of the set. ■

As we will see in the next section, it is possible to reduce to 2 the q-exponentiations, and therefore obtain a more efficient encoding process.

```
if (encode_m) {
      var y = m.add(BigInt.ONE);
      var test = y.modPow(pk.q, pk.p);
      if (test.equals(BigInt.ONE)) {
        this.m = y;
      } else
        this.m = y.negate().mod(pk.p); }
```

Listing 3.2: ElGamal encoding with q-exponentiations [Helios, 2008].

3.2.2.3 Hard Decoding

Belenios and Microsoft Belenios [Belenios, 2016] is a verifiable voting system built upon Helios. It can be used to organize elections and perform verification. Contrary to Helios, Belenios provides eligibility verifiability as anyone can check that ballots are coming from legitimate voters: each voter receives a private credential, while the election server receives only the corresponding public credential. Therefore, even if the election server is compromised, no ballot can be added. Microsoft Election guard [Microsoft, 2019] is a library that verifies voting ballots. Concerning ElGamal encryption scheme a message m is encoded as g^m where g is the QR generator of the prime order subgroup: every element written in the form of g^m belongs to the subgroup generated by g. The choice of

using the exponential version of ElGamal is to benefit from turning the multiplicative homomorphism of ElGamal into an additive one. After decryption, one should compute the discrete logarithm of g^m in order to recover the initial message m. This is possible by using Pollard-lambda algorithm or brute force only if m is taken from a small subset and not from the entire interval $\{0,...,q-1\}$. Being in a subgroup of prime order q, where q is a large prime, it is not possible to decompose the subgroup in smaller subgroups (Lagrange Theorem [Pollard, 1978]) since the computation of the discrete logarithm is unfeasible in general (see Listing 3.7).

```
type factor = elt partial_decryption
let eg_factor x {alpha; _} =
    let zkp = "decrypt|" ^ G.to_string (g **~ x) ^ "|" in
            alpha **~ x,
     fs_prove [| g; alpha |] x (hash zkp)
let check_ciphertext c =
     Shape.forall (fun {alpha; beta} -> G.check alpha && G.check beta) c
```

Listing 3.3: Hard decoding implementation [Belenios, 2016].

3.2.2.4 Encoding with 2-exponentiations

Moscow Voting System For the elections of September 2019, the Russian government decided to employ an electronic voting system [Moscow, 2019a] to elect governors for local parliaments in Moscow. In July 2019, the source code, developed by the Moscow Department of Information Technology, was made public to test its vulnerabilities. At that time, the Moscow voting system was discovered to be subjected to two attacks by researchers [Gaudry, 2019]. In the first test, the researchers exploited the fact that the keys used are small: three keys of 256 bits were used. Discrete logarithms defined by small primes are easy to calculate in feasible time. Therefore, one can compute the discrete logarithm and recover the secret keys used for decryption. Moreover, one can decrypt messages employing the same time as a legitimate possessor of secret keys. After

reporting this issue, the developers of Moscow voting system increased the key size to 1024 bits. However, a second test was made to verify the security of the modified version. In this version, the message space was not restricted to the subgroup of quadratic residues as any message was allowed to be encrypted. By relying on subgroup attacks, one can get enough information about the voter's choice and indeed can reveal one bit of information about the original message (see Section 3.1). Therefore the Decisional Diffie-Hellman assumption did not hold and the system leaked strong information. Two days before the elections, the developers modify the source code [Moscow, 2019b] and adopt an efficient method to secure their voting system. To map a message m into the QR subgroup, it is sufficient to square the message: $m \rightarrow m^2$ (see Listing 3.8).

Proof. The quadratic residue theorem states that $a \in QR$ if $\exists\ x$ s.t. $x^2 \equiv a(p)$. Let $m = x$, then $m^2 \in QR$ if $m^2 \equiv m^2(p)$, which can be trivially satisfied. ■

After decryption, one obtains m by calculating its modular square root: $m = c^{\frac{q+1}{2}}$ [Nishihara et al., 2009]. The algorithm computes the square root of c iff $p \equiv 3 \bmod 4$, which is always the case when using a safe prime. As a matter of fact, the underlying group in this last version respects the *DDH* assumption. We will discuss the adopted method in the next section.

```
const sessionKey = trimBigInt(xoredRandomBigInt, this.moduleP.bitLength() − 1);
const squaredData = dataAsBI.modPow(new BigInt('2'), this.moduleP);
const a = this.generatorG.modPow(sessionKey, this.moduleP).toString();
const b = this.publicKey
                .modPow(sessionKey, this.moduleP)
                  .multiply(squaredData)
                 .remainder(this.moduleP)
                .toString();
```

Listing 3.4: ElGamal encoding with 2-exponentiations [Moscow, 2019b].

3.3 Comparison of encodings

	T1: Limited msg space & q-expon.	T2: q-expon.	T3: m-expon.	T4: 2-expon.
Encoding	$m^q \equiv 1\ (p)?\ m : \text{error}$	$m^q \equiv 1\ (p)?\ m : -m$	g^m	m^2
Decoding	m	$m < q\ ?: m : -m$	$log_g\ (g^m)$	$m^{\frac{q+1}{2}}$

Table 3.2: Message Encoding Comparison.

In the previous section, we have seen 4 different encoding techniques of ElGamal that comply with the *DDH* assumption. In Table 3.2, we summarize the four techniques by giving a general overview on the encoding and the decoding processes.

T1 The first message encoding technique (T1) checks whether a message m is a QR or not by checking the following equivalence: $m^q \equiv 1\ (p)$. If the equivalence holds, then the message is encrypted, otherwise an error is raised and the message is rejected. The decoding operation is simple as one outputs directly the message m.

T2 The second message encoding technique (T2) uses the same method as the previous one, but instead of raising an error and rejecting the messages, it maps the message as $-m$. For what concerns the decoding process, one first checks if $m < q$ and output m, otherwise it outputs $-m$.

T3 The third message encoding technique (T3) maps a message m as g^m. The decoding mechanism is hard in general and can be only applied on a small subset of messages in which the computation of discrete logarithm can be solved by brute force.

T4 Concerning the fourth message encoding technique (T4), one maps a message m as m^2 into the subgroup of order q. This squaring technique is sufficient to map any message as a QR element. In addition, it is efficient as it needs only 2 exponentiations for encoding any message. To decode, one computes the square root by modular exponentiation of m: $m^{\frac{q+1}{2}}$ to recover the original message.

Whereas in T4, the encoding technique is faster than T1 and T2 (2-exponentiations is more performant than q-exponentiations as q is large), the decoding process is T1

and T2 is faster. However, for what concerns electronic voting systems, usually several encryptions are made and only one decryption is needed to reveal the result of an election. We conclude that T4 is more efficient to apply to electronic voting systems. Additionally, as reported in the note on Moscow voting systems [Gaudry, 2019], this technique is efficient since the decryption (q-exponentiations) is usually done on high-end servers, while the encryption (2-exponentiations) is done on the voter's device.
In addition, T1, T2, T3, and T4 implement a QR generator using q-exponenentiations since they check its quadratic residuosity by calculating $g^q(p)$. However, one can simply implement a quadratic residue generator by using only 2-exponentiations instead of using q-exponentiations (q-exponentiations are used in [Belenios, 2016, Helios, 2008, Microsoft, 2019, Moscow, 2019b, Verificatum, 2017] for the quadratic residue generator). But clearly, a more direct and efficient way to calculate the generator is by fixing it in advance. For example $g = 4$ if we are working in $\mathbb{Z}_p^*$.
Note that the performance of the encoding is more important than the performance of the generator calculation, which occurs only once at the initial phase of the voting process. Moreover, the performance of the decoding is also less important than the encoding in a voting process as discussed previously.

We provide a reference implementation in Ocaml [Rémy, 2000] (see Listings 3.5, 3.6, 3.7, 3.8, 3.9, 3.10, 3.11) in which we apply the encoding and decoding process as in T4. In addition, in our implementation, the generation of the quadratic residue generator differs from all the other implementations as we use 2-exponentiations instead of q-exponentiations.

- We generate a safe prime of the form $p = 2 \cdot q + 1$ and check the primality of both p and q.

```
let rec random_safe_prime nbits =
  let q = sample (nbits − 1) in
  let q = Z.nextprime q in
  let p = Z.succ (Z. shift_left q 1) in
  if Z.probab_prime q 10 <> 0 && Z.probab_prime p 10 <> 0 then p
```

```
  else random_safe_prime nbits
```

Listing 3.5: Our Ocaml implementation for a safe prime generator.

- We implement a QR generator for safe prime order groups.

```
let generator pbits p =
  let q = Z.shift_right p 1 in
  let g = Z.succ (sample_le pbits (Z.sub p (Z.of_int 2))) in
  Z.powm g (Z.of_int 2) p
```

Listing 3.6: Our Ocaml implementation for a QR Generator.

- We define the property of the group we use in our implementation, with p a random safe prime group, and g its generator.

```
let sample_group pbits =
  let p = random_safe_prime pbits in
  let g = generator pbits p in { pbits = pbits }
```

Listing 3.7: Our Ocaml implementation for defining the group property.

- We encode the message to encrypt as a QR element: $(m+1)^2 \ mod \ p$.

```
let encode gr m = Z.powm (Z.succ m) (Z.of_int 2) gr.p
```

Listing 3.8: Our Ocaml implementation encoding a message as a QR element.

- For encryption, we first check that $0 < m < q-1$. Then we encrypt the message as a pair $(g^r \ mod \ p, pk^r \cdot encode(m) \ mod \ p)$. Notice that $r \ mod \ p$ is a random element in q and pk is the public key $g^x \ mod \ p$.

```
let encrypt gr pk m =
  if ((Z.leq Z.zero m) && (Z.lt m (Z.pred (q gr)))) then
    let r = sample_le (gr.pbits − 1) (q gr) in
```

```
    (Z.powm gr.g r gr.p, mulm gr (Z.powm pk r gr.p) (encode gr m))
  else raise (Invalid_argument "ElG encryption")
```

Listing 3.9: Our Ocaml implementation for a message encryption.

- Concerning the decryption, we decrypt the message m by using the private key sk. We first calculate the inverse of $g^r \ mod \ p$ with functions *mult* and *modulo*, then multiply the encoded encrypted message with the resulting inverse to obtain the decryption of m.

```
let decrypt gr sk (u,v) =
  let mult = Z.mul (Z.pred (q gr)) sk in
  let modulo = Z.powm u mult gr.p in
  let dec = mulm gr v modulo in
  decode gr dec
```

Listing 3.10: Our Ocaml implementation for a message decryption.

- Let $r = m^{\frac{q+1}{2}} \ mod \ p$. To decode a decrypted message, we check if $m \leq r$. If it is the case then $m = r$ else $m = p - r$.

```
let decode gr m =
  let p = gr.p in
  let q = q gr in
  let r = Z.powm m (Z.shift_right (Z.succ q) 1) p in
  let m = if Z.leq r q then r else (Z.sub p r) in
  (Z.pred m)
```

Listing 3.11: Our Ocaml implementation for decoding a decrypted message.

3.4 Related work

The ElGamal Scheme. The ElGamal encryption scheme was introduced in 1985 by Taher ElGamal [ElGamal, 1985]. It relies on Diffie-Hellman key exchange and is known to be semantically secure under the Decisional Diffie-Hellman (*DDH*) assumption where the discrete logarithm problem is hard to solve. In 2009, Barthe et al. [Barthe et al., 2009] proved that the ElGamal encryption scheme is secure against chosen-plaintext attacks (CPA) in the standard model assuming that the *DDH* problem is hard in the underlying group family by using the proof assistant Coq. The Cramer-Shoup cryptosystem (CS) [Cramer and Shoup, 1998] was developed by Ronald Cramer and Victor Shoup in 1998. It is a generalization of ElGamal's protocol and is provably secure against adaptive chosen ciphertext attacks (CCA), under the *DDH* assumption. Even though it is a modified version of ElGamal, the Cramer-Shoup cryptosystem cannot be used however as a substitute of ElGamal in voting systems. In fact, being resistant to CCA results in losing the homomorphic property of the scheme, which is fundamental for voting systems. In 2006, Benoît Chevallier-Mames et al. [Chevallier-Mames et al., 2006] proposed an ElGamal encryption alternative, using a new encoding-free technique. Their approach holds better performance than plain ElGamal without the necessity to map the message into a subgroup. The authors introduce the notion of the class function based on the difficulty of the Decisional Class Diffie-Hellman (DCDH) assumption. An essential improvement of the Encoding-Free version scheme is to avoid message conversion, providing a wider message space. ElGamal encoding-free is *IND-CPA*. However, to date, it is not known how to identify whether a group satisfies the DCDH assumption or not. An encoding-free version of ElGamal cryptosystem over elliptic curves has also been proposed in 2017 by Marc Joye et al. in [Joye and Libert, 2017].

Semantic Security. The mental poker [Rivest et al., 1979] is the first protocol known to be vulnerable to attacks because its encryption scheme does not respect the *DDH* assumption. The game of mental poker is an ordinary poker game and communications between players are done via messages since it is a game without physical cards. Being exposed to attacks, the mental poker game can leak one bit of information about the

cards by observing whether the encryption scheme preserves the quadratic residuosity of a number [Lipton, 1981]. Consequently, in 1982, Goldwasser and Micali introduced the first probabilistic public-key encryption scheme [Goldwasser and Micali, 1982] which is provably secure under standard cryptographic assumptions. It is based on the intractability of Quadratic Residuosity Assumption modulo a composite n. Considering that the distribution of quadratic residues and quadratic non-residues is not the same, one restricts oneself to a subset where the number of quadratic residues is equal to the number of quadratic non-residues, and takes only the messages that are quadratic residues to prevent an attacker from gaining any information about the original message.

Encoding Mechanisms. What we call message encoding refers to the mechanism that maps a message into a specific group. An approach to encode a message is the hash-ElGamal encoding. This scheme consists of including a hash function during the encryption process. Let $h : \mathbb{G} \to \{0,1\}^l, w \to h(w)$ be a hash function mapping elements to l-bit strings. The encryption of a message $m \in \mathbb{M}$ where the message space is defined as $\mathbb{M} = \{0,1\}^l$ is then given by $(g^r, m \oplus h(y^r))$. This encoding mechanism solves the problem of leaking information about the original message but unfortunately, it cannot be used for voting systems as it loses its homomorphic property. Another option to encode messages is exponent-ElGamal encoding [Cramer et al., 1997]. This technique takes advantage of a property where any element $w \in \mathbb{G} =< g >$ is uniquely represented as $w = g^z$ for some $z \in \mathbb{Z}/q\mathbb{Z}$. For any message $m \in \mathbb{Z}/q\mathbb{Z}$, the resulting encoding is g^m. The corresponding ciphertext is given by $(c_1, c_2) = (g^r, g^m y^r)$. In this case, the problem is in the decryption process: to retrieve the original message, the computation of the discrete logarithm in $\mathbb{G}$ is needed. Considering that the discrete logarithms are hard to solve in $\mathbb{G}$, this leads to limit the message space to a small set where the discrete logarithm problem is easy to solve by using brute force or Pollard's rho algorithm [Pollard, 1978].

Elliptic curve ElGamal is a different variant where a message m is represented as a point on an elliptic curve, more accurately, as a point on a prime order subgroup. Elliptic curve cryptography [Koblitz et al., 2000, Miller, 1985] offers smaller key sizes resulting in gains in speed and memory, and benefits of the absence of sub-exponential algorithms

that solve the elliptic curve discrete logarithm problem. However, encodings [Farashahi, 2014, Fadavi et al., 2018, Bernstein et al., 2013] mostly do not handle prime-order elliptic curves as there is no known polynomial time algorithm for finding a large number of points on an arbitrary curve. Furthermore, several encodings are hash-to-curve that are not invertible and therefore not compatible with group operation which destroys the homomorphic property [Faz-Hernandez et al., 2019].

Application to E-voting. We briefly discuss electronic voting systems that use ElGamal encryption. To get familiar with voting systems, we refer the interested reader to [Ne Oo and Aung, 2014, Cortier et al., 2016, Volkamer, 2009]. E-voting systems are important as they expand the participation of voters and offer an efficient way to count votes. However, without employing secure systems, the use of voting systems would be meaningless. E-voting uses public-key cryptography: ElGamal is the most common used encryption algorithm as it enjoys multiplicative homomorphic properties that allows the addition on the ciphertexts in order to count ballots without revealing the identity of the voters. Additionally, ElGamal enables re-encryption which results in a different ciphertext containing the same information. (Paillier encryption scheme [Paillier, 1999] is a possible option as well, relying on the DCR assumption). Various studies demonstrated the importance of ElGamal as an encryption scheme for electronic voting systems [Haines et al., 2019, Adida, 2008]. Additionally, several countries (e.g. Estonia [Kubjas et al., 2017], Norway [Puigalli and Guasch, 2012] and Russia [Babenko et al., 2017]) are using e-voting systems as a main mechanism for elections and are employing ElGamal to count and verify votes. Moreover, in a note of 15 November 2019, Pierrick Gaudry [Gaudry, 2019] reveals an attack about the Moscow voting system because it does not comply with the *DDH* assumption.

3.5 Conclusion

During our analysis, we have discovered a number of ElGamal scheme implementations that are not *IND-CPA* secure since they do not respect the *DDH* assumption. On one

hand, some implementations do not employ safe primes, an essential condition to form subgroups of large prime order in which the *DDH* assumption holds. On the other hand, other implementations do not apply message encoding mechanisms or use Quadratic Residue Generators. As a consequence, 20 out of the 26 analyzed libraries may leak one bit of information about the original message and therefore, may endanger the validity of an election. Finally, after comparing four different message encoding techniques that satisfy the *DDH* assumption, we conclude which implementation is most convenient for voting systems. We focused the current study on manually analyzing the *IND-CPA* security of open source code libraries of ElGamal encryption scheme. However, it is also possible to check the *IND-CPA* (in-) security when source code is not available. In fact, by applying the technique discussed in the Example 5 of Section 3.1, one can black-box test applications. In particular, such tests can be applied to ElGamal encryptions obtained by Hardware Security Modules (*HSM*) [Volkamer, 2009, Orr and Liam, 2016], which are used e.g. in the Estonian I-voting system [Springall et al., 2014].

In this chapter, we study correct implementations, regarding message encoding, of ElGamal over cyclic subgroups of $\mathbb{Z}_p^*$. In the case of ElGamal over elliptic curves the underlying hard problem, the elliptic curve discrete logarithm problem, compels the encoding of the plaintext message m as a point on on a prime-order subgroup G of an elliptic curve. We leave the study of the implementations of such encodings as future work.

CHAPTER 4

BROADCAST ENCRYPTION SCHEMES

Broadcast encryption schemes (BES) [Fiat and Naor, 1993, Boneh et al., 2005, Lee et al., 2019] offer an efficient solution in means of transmission length and key storage to securely broadcast messages to a privileged subset of nodes. They also aim to allow two nodes (unknown to each other), to agree on a common key. Broadcast encryption schemes provide cryptographic primitives to generate a common cryptographic key for a subgroup of nodes to ensure that only nodes of a privileged subgroup can decrypt a message. Generally, broadcast encryption schemes offer high security properties while adopting small key sizes [Boneh et al., 2005, Lee et al., 2019]. Some schemes are known to be collusion resistant, which ensures that non authorized nodes cannot learn anything about the broadcast message. Other schemes enjoy traitor-tracing characteristics, in the sense that a sender can trace back which dishonest nodes have leaked the decryption key to non authorized nodes. Broadcast encryption schemes are classified as stateless or stateful. While stateless schemes provide nodes with permanent keys, stateful schemes provide nodes with updatable keys under certain conditions i.e. join or revocation event.

In this chapter, we investigate broadcast encryption schemes while providing some examples and discussing their security aspects. We also propose a new broadcast encryption scheme based on ElGamal [ElGamal, 1985]. We finally implement and compare three

broadcast encryption schemes in terms of execution time, key storage and ciphertext space.

In Section 4.1, we give the general definition of a broadcast encryption scheme then present several schemes as the Fiat-Naor scheme,the ElGamal Baseline scheme, and the Boneh-Franklin scheme. In Section 4.1.4, we propose a new broadcast encryption scheme with constant size ciphertext and key storage based on ElGamal. Section 4.2 is devoted for the comparison and evaluation in terms of time and space of three broadcast encryption scheme. We implement and compare the execution time (Table 4.1) , ciphertext size (Table 4.3) and the maximum key storage (Table 4.2) in ElGamal Baseline, Fiat-Naor and the new scheme.

In summary, our contributions in Chapter 4 are:

- We propose a new broadcast encryption scheme based on ElGamal with constant size key storage and ciphertext.
- We implement and compare the execution time, the ciphertext size and the maximum key storage for nodes of three different broadcast encryption schemes.

4.1 General Definition of BES

Broadcast encryption schemes allow the sender to dynamically choose a privileged subset of nodes and send a ciphertext in such a way that only nodes in the privileged subset can read the message. We begin by formally defining what is a broadcast encryption scheme.

Definition 23. *A broadcast encryption scheme for a set of nodes S and a server consists of four primitives defining an encryption scheme and a broadcast primitive. The encryption scheme includes:*

- $Setup(S,\lambda)$ *An initial setup algorithm that given a set S of n nodes and a security parameter* λ*, generates a master key K for the server (only the server has K) and n public/private pairs of keys* (PK_i, SK_i)*, one pair for each node.* SK_i *is private to*

node n_i. Private key SK_i will be used for node n_i in S to compute the decryption key whenever n_i belongs to the privileged subset.

- *$KG(L,K)$ A key generation algorithm used at the server that takes as input a set $L \subseteq S$ of nodes and a master key K and generates an encryption key K_L to encrypt a message for nodes in L.*

- *$Encrypt(L,K_L,m)$ An encryption algorithm that takes as input a subset L of nodes, an encryption key K_L for subset L, and a message m. The algorithm outputs a ciphertext and a header $(H_1,\ldots,H_n)$.*

- *$Decrypt(L,SK_i,H_1,\ldots,H_n,C_m)$ A decryption algorithm that is used in a node n_i that takes as input a subset L, a private key SK_i for node n_i, headers $H_1,\ldots,H_n$ for all nodes in L, and a ciphertext C_m. First the algorithm calculates the decryption key for L by using SK_i and public keys of nodes in L and then it outputs the decrypted message.*

In what follows, we present three broadcast encryption schemes and propose a new scheme based on ElGamal encryption scheme [ElGamal, 1985].

4.1.1 Fiat-Naor

The first broadcast encryption scheme was proposed by A. Fiat and M. Naor [Fiat and Naor, 1993]. In their scheme, they consider a key distribution center and a set of nodes. Their idea is to assign predefined keys to nodes, and compute a common key whenever the center wants to broadcast a message to a privileged subset of nodes. To retrieve the message, each node has to compute the common key itself. This scheme is cryptographically equivalent to *RSA* [Rivest et al., 1978]. The scheme consists of four algorithms as follows:

- Setup. The server chooses a composite number that is hard to factorize $N = P \cdot Q$, where P and Q are large primes, an element g of high value kept secret, and $\phi(N) = (P-1)\cdot(Q-1)$.

- Key Generation. For each node $i \in T$, the server assigns a private key $g_i = g^{p_i}$ and public key $p_i : \{g_i, p_i\}$, where p_i and p_j are relatively prime for all $i, j \in T$ and T is the subset of targeted nodes to whom the server is willing to broadcast messages. The server also computes a common key $g_T = g^{\prod_{i \in T} p_i} \bmod N$.

- Encryption. To broadcast a message m, the server computes the encryption key as $K_{enc} = {g_T}^{-1} \; mod \; \phi(N)$, then encrypts m as $m^{K_{enc}} \; mod \; N$. The server broadcasts the following ciphertext: $(p_1, ..., p_k, m^{K_{enc}} \; mod \; N)$.

- Decryption. To decrypt, every node $i \in T$ evaluates $g_T = {g_i}^{\prod_{j \in T - \{i\}} p_j} \; mod \; N$, and computes $m = (m^{K_{enc}} \; mod \; N)^{g_T} \; mod \; N$.

Example 7. *In this example, we consider a server willing to broadcast a message m to a privileged subset of nodes* (n_0, n_1)*. We also show how a node* n_2 *(that does not belong to the privileged subset) cannot decrypt the message.*
Setup and Key generation. The server chooses a composite number that is hard to factorize $55 = 5 \cdot 11$ *and an element g of high value kept secret. The server also computes a common key* $g_T = g^{p_0 \cdot p_1} \; mod \; N$.

Phase 1: Setup and Key Generation

1: The server selects a generator $g = 48$.
2: For each node, the server assigns a private key $g_i = g^{p_i}$ and public key $p_i : \{g_i, p_i\}$
   a: For node n_0, the server assigns $\{27, 7\}$.
   b: For node n_1, the server assigns $\{53, 13\}$.
3: The server computes $g_T = 48^{7 \cdot 13} \; mod \; 55 = 37$.

Broadcast. To broadcast a message $m = 6$ *to nodes in* n_0 *and* n_1*, the server computes the encryption key as* $K_{enc} = 37^{-1} \; mod \; 55$*, then encrypts m as* $m^{K_{enc}} \; mod \; 55$*. The server broadcasts the following ciphertext:* $(n_0, n_1, m^{K_{enc}} \; mod \; 55)$.

Phase 2: Broadcast

1: The server selects a message $m = 6$.
2: To encrypt the m, the server computes the encryption key as $K_{enc} = 37^{-1} \; mod \; 55 = 3$.
3: The server encrypts m as $6^3 \; mod \; 55 = 51$.
4: The server broadcasts $(7, 13, 51)$.

Decryption. Upon receiving the broadcasted message, nodes n_0 and n_1, evaluate the decryption key $g_T = g_i^{\prod_{j \in T-\{i\}} p_j} \ mod\ 55$, and computes $m = (m^{K_{enc}} \ mod\ N)^{g_T} \ mod\ 55$. Note that node n_2 cannot compute the decryption as he cannot evaluates the decryption key.

Phase 3: Decryption for nodes n_0 and n_1

1: To decrypt, n_0 and n_1 compute the decryption key respectively as $27^{13} \ mod\ 55 = 37$ and $53^7 \ mod\ 55 = 37$.
   a: Node n_0 computes $m = 51^{37} \ mod\ 55 = 6$.
   b: Node n_1 computes $m = 51^{37} \ mod\ 55 = 6$.

Security Aspects. The security of Fiat-Naor broadcast encryption scheme is equivalent to the security of *RSA*. It is based on the difficulty of factorizing a large composite integer. This is the reason why the server chooses N, a composite of two large primes. Moreover, it is k-resilient, in the sense that is secure against a coalition of at most k non privileged nodes. This scheme can be adapted to use *RSA-OAEP* algorithms to obtain an *IND-CCA2* security [Barthe et al., 2011a]. Notice that elliptic curves cryptography offers the same security with smaller key size [Gura et al., 2004], one may consider replacing *RSA* algorithms relying on elliptic curves cryptography.

4.1.2 ElGamal Baseline

We present ElGamal Baseline scheme [Chhatrapati et al., 2021] is a simple baseline broadcast encryption scheme based on ElGamal encryption scheme. This scheme is the broadcast version of the basic scheme of ElGamal (See Chapter 3). The idea of the ElGamal Baseline is broadcast a message to a subset of nodes by encrypting k times the same message using k public keys of k nodes. While in the basic scheme, the server selects one private key and compute the corresponding public key, in the ElGamal Baseline scheme, the server selects k private keys and computes k public keys.

- Setup. The server chooses a safe prime order group $\mathbb{Z}_p^*$ where $p = 2 \cdot q + 1$, with p and q large primes. Let $\mathbb{G}_q$ be the subgroup of $\mathbb{Z}_p^*$ of order q.

- Key Generation. The server selects $g \in \mathbb{G}_q$ to be the generator of $\mathbb{G}_q$. For each node n_i with $i = 1,...,k$, the server chooses a random $x_i \in \mathbb{Z}_q$ and computes $y_i = g^{x_i} \ mod \ p$. The server outputs $g, y_1, ..., y_k$.
- Encryption. To encrypt a message $m \in \mathbb{G}_q$ to a set of nodes S, the server picks a random $r \in \mathbb{Z}_q$. For each node $i \in S$, the server computes $v_i = m \cdot {y_i}^r \ mod \ p$. The server broadcasts the following ciphertext: $(g^r \ mod \ p, v_1, ..., v_k) = (u, v_1, ..., v_k)$.
- Decryption. To decrypt, a node i computes $m = \frac{v_i}{u^{x_i}} \ mod \ p$.

Example 8. *Setup and Key Generation. The server chooses a safe prime order group* $\mathbb{Z}_{11}^*$ *and a generator* $g \in G_5$ *where* G_5 *is the subgroup of quadratic residues of* $\mathbb{Z}_{11}^*$ *[El Laz et al., 2020]. For nodes* n_0 *and* n_1*, the server chooses* $(x_0, x_1) \in \mathbb{Z}_5$ *and computes* $y_i = g^{x_i} \ mod \ 11$.

Phase 1: Setup and Key Generation

1: The server selects a generator $g = 4 \in G_5$.
2: For each node n_i with $(i = 0, 1)$, the server chooses $x_i \in \mathbb{Z}_5$ and computes $y_i = g^{x_i}$.
   a: For node n_0, the server picks $x_0 = 2$ and computes $y_0 = 4^2 \ mod \ 11 = 5$.
   b: For node n_1, the server picks $x_0 = 3$ and computes $y_1 = 4^3 \ mod \ 11 = 9$.
3: The server outputs $(43, 9)$.

Encryption. To encrypt a message $m \in \mathbb{G}_5$*, the server picks a random* $r \in \mathbb{Z}_5$*. For each node, the server computes* $v_i = {y_i}^r \ mod \ 11$*. The server broadcasts the following ciphertext:* $(u, v_0, ..., v_1)$.

Phase 2: Broadcast

1: The server selects a message $m = 3 \in G_5$ and a random element $r = 2 \in \mathbb{Z}_5$.
2: To encrypt the m, the server computes $v_i = {y_i}^r \ mod \ 11$.
   a: For node n_0, the server computes $v_0 = 3 \cdot 5^2 \ mod \ 11 = 9$.
   b: For node n_1, the server computes $v_1 = 3 \cdot 9^2 \ mod \ 11 = 9$.
3: The server broadcasts $(5, 9, 1)$.

Decryption. To decrypt, a node i *computes* $m = \frac{v_i}{u^{x_i}} \ mod \ 11$ *using its private key* x_i.

Security Aspects. The security of the ElGamal Baseline scheme is equivalent to the security of ElGamal [ElGamal, 1985] encryption scheme. Hence, it is *IND-CPA* under the assumption of *DDH* (We refer the reader to Section 2.4 of Chapter 2).

Phase 3: Decryption for node n_0 and n_1

1: To decrypt, n_0 and n_1 use their respective private keys 3 and 9.
   a: Node n_0 computes $m = \frac{9}{5^2}\ mod\ 11 = 3$.
   b: Node n_1 computes $m = \frac{1}{5^3}\ mod\ 11 = 3$.

4.1.3 Boneh-Franklin

Te broadcast encryption scheme proposed by A. Boneh and M. K. Franklin [Boneh and Franklin, 1999] is a traitor tracing scheme. Besides allowing to broadcast to a privileged subset of nodes, it also allows the server to identify a traitor in the system. The traitor in this scheme is a dishonest user who leaks the decryption key to an unauthorized node. Since the scheme is based on error-correcting codes, the tracing problem can be viewed as watermarking the distributed secret keys.

- Setup. The server chooses a safe prime order group $\mathbb{Z}_p^*$ where $p = 2 \cdot q + 1$, with p and q large primes. Let $\mathbb{G}_q$ be the subgroup of $\mathbb{Z}_p^*$ of order q.

- Key Generation. The server selects $g \in \mathbb{G}_q$ to be the generator of $\mathbb{G}_q$. For each node n_i with $i = 1,...,k$, the server chooses a random $r_i \in \mathbb{Z}_q$ and computes $h_i = g^{r_i}\ mod\ p$. The public key is the set $(y, h_1, ..., h_k)$, where y can be written in the form of $y = \prod_{i=1}^{k} {h_i}^{\alpha_i} = g^{\prod_{i=1}^{k} r_i \cdot \alpha_i}$ for random $\alpha_i \in \mathbb{Z}_q$.

  A private key is an element $\theta_i \in \mathbb{Z}_q$ such that $\theta_i \cdot \gamma^{(i)}$ is the representation of y with respect to the base $< h_1, ..., h_k >$.

  Let $\Gamma = \{\gamma^{(1)}, ..., \gamma^{(k)}\}$ where each $\gamma^{(i)} = (\gamma_1, ..., \gamma_k)$ a vector over $\mathbb{Z}_q$. The set Γ is fixed in advance and not secret. There exist several methods to compute Γ, we only show one method: for each node n_1, the server computes $\gamma^{(1)} = (\frac{\alpha_1}{\alpha_1}, \frac{\alpha_2}{\alpha_1}, ..., \frac{\alpha_k}{\alpha_1})$. Each node n_i then receives α_i which will also be its decryption key θ_i.

- Encryption. To broadcast a message $m \in \mathbb{G}$, the server first picks a random element $r \in \mathbb{Z}_q$ and encrypts the message m as $m \cdot y^r\ mod\ p$. The server then broadcasts the following ciphertext: $({h_1}^r, ..., {h_k}^r, m \cdot y^r\ mod\ p) = (H_1, ..., H_k, C)$.

- Decryption. To decrypt, a node n_i computes m as:

$$m = \frac{C}{U^{\theta_i}} \text{ where } U = \prod_{i=1}^{k} {H_i}^{\gamma_i}.$$

Example 9. *We consider a server willing to broadcast a message m to a targeted subset of nodes* (n_0, n_1).

Setup. The server chooses a safe prime order group $\mathbb{Z}_{11}^*$ *and a generator* $g \in G_5$*where* G_5 *is the subgroup of quadratic residues of* $\mathbb{Z}_{11}^*$ *[El Laz et al., 2020].*

Phase 1: Key Generation

1: The server selects a generator $g = 4 \in G_5$.
2: For each node n_i with $(i = 0, 1)$, the server chooses $r_i \in \mathbb{Z}_5$ and computes $h_i = g^{r_i}$.
   a: For node n_0, the server picks $r_0 = 2$ and computes $h_0 = 4^2 \ mod \ 11 = 5$.
   b: For node n_1, the server picks $r_1 = 4$ and computes $h_1 = 4^4 \ mod \ 11 = 3$.
3: The server outputs a set of public keys as (y, h_0, h_1).
   a: For node n_0, the server picks $\alpha_0 = 1$.
   b: For node n_1, the server picks $\alpha_1 = 3$.
   c: $y = \prod_{i=0}^{1} {h_i}^{\alpha_i} = 3$.
4: The server outputs a public set of vectors $\Gamma = (\gamma^{(0)}, \gamma^{(1)} = ((1,3),(2,1))$ calculated as follows:
   a: For $\gamma^{(0)}$, the server computes $(\frac{\alpha_0}{\alpha_0}, \frac{\alpha_1}{\alpha_0}) = (1,3)$.
   b: For node $\gamma^{(1)}$, the server computes $(\frac{\alpha_0}{\alpha_1}, \frac{\alpha_1}{\alpha_1}) = (2,1)$.

Key Generation. The server selects $r_i \in \mathbb{Z}_5$ *for each* n_i *and computes* $h_i = g^{r_i}$ *and outputs a set of public keys* (y, h_0, h_1, h_2)*, with* $y = \prod_{i=0}^{2} {h_i}^{\alpha_i}$ *and* $\alpha_i \in \mathbb{Z}_5$.

Since the server is willing to broadcast only to nodes n_0 *and* n_1*, he publishes* $\Gamma = \{\gamma^{(0)}\gamma^{(1)}\}$ *and securely assign the private keys* $\alpha_0 = \theta_0$ *to* n_0 *and* $\alpha_1 = \theta_1$ *to* n_1.

Broadcast. To broadcast a message $5 \in G_5$ *to nodes in* n_0 *and* n_1*, the server first chooses are random element* $4 \in \mathbb{Z}_5$*, then encrypts the message. The server then broadcasts the following ciphertext :* $({h_0}^r, {h_1}^r, m \cdot y^r) = (H_0, H_1, C)$.

Phase 2: Broadcast

1: The server selects a message $m = 5 \in G_5$ and a random element $r = 4 \in \mathbb{Z}_5$.
2: To encrypt the m, the server computes $C = Encrypt(\{n_0, n_1\}, 3, 5) = 5 \cdot 3^4 \ mod \ 11 = 9$.
3: The server then computes $H_0 = 5^4 = 9$ and $H_1 = 3^4 = 4$.
4: The server broadcasts $(H_0, H_1, C) = (9, 4, 9)$.

Decryption. Upon receiving the broadcasted message, node n_0 *and* n_1 *use* θ_0 *and* θ_1 *for* n_1 *to decrypt the ciphertext. The decryption process is described in the table below for* n_0*. The*

node n_1 computes the same operation by using its private key. Note that node n_2 cannot compute the decryption as it does not possess its private key.

Phase 3: Decryption for node n_0

1: To decrypt, n_0 uses its private key θ_0.

a: Node n_0 computes $m = \frac{C}{U^{\theta_0}} = \frac{9}{4^1} = 5$ where $U = H_0^{\gamma_0^{(0)}} \cdot H_1^{\gamma_1^{(0)}} = 4$.

Security Aspects. Besides being *IND-CPA* under the *DDH* assumption as proven in [Boneh and Franklin, 1999], the Boneh-Franklin scheme authors presented a modified version secure against adaptive attacks.

4.1.4 A New BES Based On ElGamal

We propose a new broadcast encryption scheme with constant size ciphertext and key storage based on ElGamal. Its security relies on the eth root problem (We refer the reader to Section in Chapter 2) and is *IND-CPA* under the assumption of *DDH* in the random oracle model (The proof has been made in Easycrypt [Barthe et al., 2013] but it is not part of the contribution of this thesis). In contrast with the broadcast encryption schemes discussed in Subsections 4.1.1, 4.1.2 and 4.1.3, in the new scheme, a node stores only one private key and one private key for all the subgroups it belongs to.

- Setup. The server chooses a public safe prime $s = 2 \cdot s' + 1$, with s and s' large primes and $2^k \leq s' \leq 2^{k+1}$. The server also chooses a composite number $N = p \cdot q$ hard to factorize, with $p = 2 \cdot p' + 1$, $q = 2 \cdot q' + 1$ large safe primes kept secret, $p, q \equiv 3 \ mod \ 4$, $\phi(N) = (p-1)(q-1)$ and $N < s'$.

  Let $\mathbb{G}_{s'}$ be the subgroup of $\mathbb{Z}_s^*$ of order s'. The server selects $g \in \mathbb{G}_{s'}$ to be the generator of $\mathbb{G}_{s'}$, with g being public. Finally, the server computes a list L of public keys pk_i for each node where pk_i, pk_j are relatively prime for all $i, j \in \mathbb{Z}_N^*$ such that $gcd(pk_i, pk_j) = 1$ and $gcd(\prod_{pk_i \in S} pk_i, \phi(N)) = 1$, for $\forall \ S \subseteq L$.

- Key Generation. Let $\mathbb{G}_{q'}$ be the subgroup of $\mathbb{Z}_N^*$ of order q'. The server selects $h \in \mathbb{G}_{q'}$, where h is kept secret. For each node i the server assigns a secret key $SK_i = h^{pk_i} \ mod \ N$. The server outputs $(g, pk_1, ..., pk_k)$.

- Encryption. The server evaluates $x' = h^{(\prod_{pk_i \in S} pk_i)} \ mod \ N$, where $S \subseteq L$. Let H be a hash function that takes a bitstring and returns a bitstring in $\{0,1\}^{k+1}$. The server calculates $x = H(x')$ and computes $y = g^x \ mod \ s$. To encrypt a message $m \in \mathbb{G}_{s'}$, the server picks a random $r \in \mathbb{Z}_{s'}$ and encrypts m as $m \cdot y^r \ mod \ s$. The server broadcasts the following ciphertext: $(\prod_{pk_i \in S} pk_i) \ mod \ N, g^r \ mod \ s, m \cdot y^r \ mod \ s) = (z, u, v)$.

- Decryption. To decrypt, node i first computes $pk = \frac{z}{pk_i}$. To calculate its decryption key k_{dec}, node i computes $k'_{dec} = SK_i^{pk} \ mod \ N$, then $k_{dec} = H(k'_{dec})$. To retrieve the message m, node i computes:

$$m = \frac{v}{u^{k_{dec}}} \ mod \ s.$$

Example 10. *Setup and Key Generation. The server chooses a safe prime order group* $\mathbb{Z}_{47}^*$ *and a generator* $g \in G_{23}$ *where* G_{23} *is the subgroup of quadratic residues of* $\mathbb{Z}_{47}^*$ *[El Laz et al., 2020]. It also chooses a composite number hard to factorize* $21 = 3 \cdot 7$ *with* $\phi(21) = (3-1)(7-1)$ *and* $21 < 23$. *The server selects* $g = 4 \in \mathbb{G}_{23}$ *and* $h = 1 \in \mathbb{G}_3$. *For each node* i *the server assigns a secret key* $SK_i = h^{sk_i} \ mod \ 21$, *where* sk_i, sk_j *are relatively prime for all* $i, j \in \mathbb{Z}_{21}^*$. *The server outputs* $(g, sk_1, ..., sk_k)$.

Phase 1: Setup and Key Generation

1: The server selects a generator $g = 4 \in G_{23}$, and $h = 1 \in \mathbb{G}_3$.
2: For each node n_i with $(i = 0, 1)$, the server assigns $SK_i = h^{pk_i} \ mod \ 21$.
   a: For node n_0, the server picks $pk_0 = 13$ and computes $SK_0 = 1^{13} \ mod \ 21 = 1$.
   b: For node n_1, the server picks $pk_1 = 5$ and computes $SK_1 = 1^5 \ mod \ 21 = 1$.
3: The server outputs $(4, 13, 5)$.

Encryption. In this example, we do not consider the hash function for encryption. The server evaluates $x' = h^{(\prod_{pk_i \in S} pk_i)} \ mod \ 21$, *and computes* $y = g^x \ mod \ 47$. *To encrypt a message* $m \in \mathbb{G}_{23}$, *the server picks a random* $r \in \mathbb{Z}_{23}$ *and encrypts* m *as* $m \cdot y^r \ mod \ 47$. *The server broadcasts the following ciphertext:* (z, u, v).

Phase 2: Broadcast

1: The server selects a message $m = 16 \in G_{23}$ and a random element $r = 3 \in \mathbb{Z}_{23}$.
2: The server evaluates $x = 1^{(13 \cdot 5)}\ mod\ 21 = 1$ and computes $y = 4^1\ mod\ 47 = 4$.
3: To encrypt $m = 16$, the server computes $16 \cdot 4^3\ mod\ 47 = 37$.
4: The server broadcasts $(2, 17, 37)$.

Decryption. In this example, we do not consider the hash function for decryption. To decrypt, n_0 and n_1 compute $pk = \frac{z}{pk_i}\ mod\ 21$ and respectively calculate their decryption key as $k_{dec} = SK_0^{pk}$ and $k_{dec} = SK_1^{pk}$.

Phase 3: Decryption for node n_0 and n_1

1: To decrypt, n_0 and n_1 calculate their respective private keys as:
   a: Node n_0 computes $pk = \frac{2}{13}\ mod\ 21 = 5$ and computes $k_{dec} = 1^5\ mod\ 21 = 1$.
   b: Node n_1 computes $pk = \frac{2}{15}\ mod\ 21 = 13$ and computes $k_{dec} = 1^13\ mod\ 21 = 1$.
2: Node n_0 and n_1 calculate $m = \frac{37}{17^1}\ mod\ 47 = 16$.

Security Aspects. We conjecture that the new scheme is *IND-CPA* under the assumption of *DDH* since it is based on ElGamal encryption scheme. Moreover, its security relies on the e^{th} root problem (See Section 2.2 of Chapter 2). This problem is hard as long as factoring the composite number N is hard.

An attacker willing to compute the decryption key needs to possess either a private key assigned by the server or $h \in \mathbb{Z}_N^*$. Since we choose h not only to be an element of a composite number hard to factorize, but also an element of the prime order subgroup $\mathbb{G}_{q'}$ of $\mathbb{Z}_N^*$, it is hard to compute the $e-th$ root of h since this problem is believed to be computationally hard in such composite numbers. Therefore, an attacker cannot compute the decryption key and will not be able to learn any information about the original message. Moreover, there are no algorithms able to calculate the $e-th$ root of composite modulus.

4.2 Evaluation and Comparison

In this section, we implement and compare three broadcast encryption schemes: the ElGamal Baseline scheme, the Boneh-Franklin scheme and the new scheme. We carry out

a runtime evaluation based on experimental values for the key generation, encryption and decryption in Table 4.1. Moreover, we compare the size of keys for 1 node in n subgroups (Table 4.2), and the size of ciphertext for u nodes in 1 subgroup (Table 4.3) for each broadcast encryption schemes. While the ElGamal Baseline and the Boneh-Franklin schemes are tested over a prime order group of 1024 $bits$, our scheme is tested over a 2048 $bits$ prime order group since we aim to generate a sufficient large composite number to achieve security guarantees. We implement the schemes using the Ocaml programming language [Rémy, 2000], and the runtimes are tested on a 2017 Macbook Pro with a 3.1 GHz Quad-Core Intel Core $i7$ and 16 GB RAM. The source code of our implementation can be found here.

4.2.1 Key Generation Time

We start by analyzing the key generation time for u nodes. The results can be observed in the first row of Table 4.1. This phase requires the generation of a safe prime order group, a quadratic residue generator and the computation of public and private keys. Notice that the key generation time of ElGamal Baseline is 100 % faster than the key generation time of the Boneh-Franklin scheme, and approximately 30 % slower than the key generation time in the new scheme.

- For ElGamal Baseline scheme, we generate a safe prime order group of 1024 $bits$ ($\mathbb{Z}_p^*$), and choose a quadratic residue generator g. For u nodes, we randomly sample a list of private keys x_i and compute a list y of public keys as a set of k times $y_i = g^{x_i}$. Such operations are usually fast to compute over $\mathbb{Z}_p^*$.

- For Boneh-Franklin scheme, we also generate a 1024 $bits$ prime order group and choose a quadratic generator. For k nodes, we randomly sample a list of r_i and compute a list h of public keys as a set of u times $h_i = g^{r_i}$. Moreover, we randomly sample a list of α_i and compute a public key y are the multiplication of u group exponentiation of $h_i^{\alpha_i}$. Finally, we generate a list γ of u vectors where each vector is a set of u group multiplication. Since the number of computations is bigger than

the one in ElGamal Baseline, it explains why it takes approximately more time to compute.

- For the new scheme, we generate a 2048 $bits$ prime order group and choose a quadratic generator g. We also generate a composite number H of two large primes such that $H < q$. The choice of a 2048 $bits$ prime order group is to obtain a group $\mathbb{Z}_H^*$ of 1024 $bits$. We provide the *keygen* function with a list of pre-computed co-prime numbers as input. In our scheme, this can be done because this list is public. Because of this, the computation time of this list is not considered in the evaluation time of key generation. We only consider the time to calculate the list y of private keys as a set of k times h^{n_i}. The new scheme is faster than ElGamal Baseline and Boneh-Franklin scheme. While in Boneh-Franklin, it takes 120 seconds the generate keys for 100 k nodes, in the new scheme it takes only 45 seconds.

We evaluate and compare the execution time for $2, 5, 100, 1k, 10k$ and $100k$ nodes.

In summary, for the key generation time, we conclude that:

- The ElGamal Baseline scheme is faster than Boneh-Franklin scheme, and time increases linearly with the increase of the number of nodes.
- The Boneh-Franklin scheme is slower than ElGamal Baseline and the new scheme, and the time increases linearly with the increase of the number of nodes.
- The new scheme is faster than the other two schemes, and time increases linearly with the increase of the number of nodes.

4.2.2 Encryption Time

In the encryption phase, we measure the computation time needed by the server to encrypt a message to a set of nodes. The results can be observed in the second row of Table 4.1. By analyzing the encryption time, we observe that the encryption time for ElGamal Baseline and Boneh-Franklin are analogous. In fact, during the encryption, in

ElGamal Baseline the server computes $v_i = (h_i)^r \cdot m$ k times in addition to g^r; while in Boneh-Franklin, the server computes H_i^r u times in addition to $m \cdot y^r$, which results in the same number of group operations.

- In ElGamal Baseline scheme, the server samples are a random $r \in \mathbb{Z}_q$ and computes k times $v_i = m \cdot {y_i}^r \ mod \ p$. By observing Table 4.1, we note that the encryption time increases linearly with the increase of the number of nodes. It is clear since the server encrypts u times the same message using the public key of u nodes.

- In Boneh-Franklin scheme, the server samples are a random $r \in \mathbb{Z}_q$ and computes k times $v_i = {H_i}^r \ mod \ p$. Again, by observing Table 4.1, we note that the encryption time increases linearly with the increase of the number of nodes. It is clear since the server encrypts the message only once, but it computes u times ${H_i}^r \ mod \ p$ using the public key of u nodes.

- In the new scheme, the server samples a random $r \in \mathbb{Z}_q$. It evaluates $x = h^{\prod_{i=1}^{k} n_i} \ mod \ H$ and computes $y = g^x \ mod \ p$. The server then encrypts the message once by employing a group multiplication. In Table 4.1, in contrast with the other two schemes, we observe that the encryption time in the new scheme is constant till 1000 k nodes. Note that, for a number of node greater than 10 k, the encryption time increases by little due to the fact that the server evaluates x for a larger number of nodes. However, the time increase results to be small and the new scheme is faster than in the other two schemes.

In summary, for the encryption time, we conclude that:

- The ElGamal Baseline and the Boneh-Franklin schemes have the same encryption time. Yet, the encryption time increases linearly with the increase of the number of nodes.
- The encryption time in the new scheme is the fastest among the three schemes. Furthermore, it is constant (the increase for larger set of nodes is limited) independently of the number of nodes in the system.

4.2.3 Decryption Time

For decryption time, we focus our analysis on the decryption time for 1 node since all nodes compute the same operation to retrieve the original message. The results can be observed in the third row of Table 4.1. The decryption time of the ElGamal Baseline and the new new scheme remain constant with the increase of the number of nodes. In both schemes, to decrypt, nodes either use their own private key or compute the decryption key by only using their private key. Yet, the ElGamal Baseline scheme is faster than the new scheme since nodes use directly their own private key to decrypt while in the new scheme nodes should first compute the decryption key in order to decrypt.

- In ElGamal Baseline scheme, a node computes one group multiplication using its private key x_i to calculate the message. That is why the decryption time for one node is constant since the node only uses its private key to decrypt without the need of computing a decryption key. Hence, its decryption key is independent from the number of nodes in the subgroup.

- In Boneh-Franklin scheme, a node first computes U as u multiplication of exponentiation ${H_i}^{\gamma_i}$. Then it computes a group multiplication using its private key α_i to calculate the message. That is why the decryption time increases linearly with the increase of the number of nodes in the system.

- In the new scheme, a node first calculates $n = \frac{z}{n_i}\ mod\ H$ which is a multiplication group operation independent of the number of nodes since it is computed modulo H. Then, it computes its decryption key using its private key SK_i. To retrieve the message, the node performs a multiplication group operation. As shown in Table 4.1, the decryption time is constant and faster than the one in Boneh-Franklin scheme. Comparing to ElGamal Baseline, the decryption time in the new scheme is greater by only 0.003 seconds.

In summary, for the decryption time, we conclude that:

- The Boneh-Franklin scheme is the slowest scheme in terms of decryption since the nodes computes $k+2$ group operations.
- In ElGamal Baseline, the decryption time is constant independently of the number of nodes.
- Analogously to ElGamal Baseline, the decryption time is constant and marginally slower since nodes compute an extra operation to decrypt the ciphertext.

Item	**Scheme**	**Time of execution per u nodes**					
		$u=2$	$u=5$	$u=100$	$u=1k$	$u=10k$	$u=100k$
Key generation	ElGamal Baseline	0.0011 s	0.0037 s	0.06 s	0.6 s	6 s	60 s
	Boneh-Franklin	0.0022 s	0.0074 s	0.12 s	1.2 s	12 s	120 s
	New Scheme	0.001 s	0.002 s	0.045 s	0.45 s	4.5 s	45 s
Encryption	ElGamal Baseline	0.0019 s	0.0035 s	0.044 s	0.44 s	4.4 s	44 s
	Boneh-Franklin	0.0019 s	0.0035 s	0.044 s	0.44 s	4.4 s	44 s
	New Scheme	0.005 s	0.005 s	0.005 s	0.005 s	0.01 s	0.06 s
Decryption for one node	ElGamal Baseline	0.001 s	0.001 s	0.001 s	0.001 s	0.001 s	0.001 s
	Boneh-Franklin	0.001 s	0.003 s	0.045 s	0.45 s	4.5 s	45 s
	New Scheme	0.004 s	0.004 s	0.004 s	0.004 s	0.004 s	0.004 s

Table 4.1: Execution time for u nodes.

4.2.4 Maximum Storage of Keys

In this section, we measure the maximum storage size of keys concerning only 1 node belonging to n subgroups. The results can be observed in Table 4.2.

- In the ElGamal Baseline, each node needs to store a private key x_i for each subgroup it belongs to. For n subgroups, a node have to store n keys of 1023 $bits$ each. From Table 4.2, we can observe that the size of key storage increases linearly with the increase of the number of subgroups. The ElGamal Baseline is much better than Boneh-Franklin scheme in all subgroups. Yet, in comparison with the new scheme, it is only better when the number of subgroups is 2.

- In the Boneh-Franklin scheme, each node has to store a private key α_i and a public key Γ_i for each subgroup. We assume for this calculation that each subgroup has a maximum of 2 nodes. This is an arbitrary choice to calculate a minimal lower bound for key storage in Boneh-Franklin scheme. Notice that the more nodes are in each subgroup , the larger space is needed for key storage in this scheme. For n subgroups, a node stores n private keys and $2 \cdot n$ keys of 1023 $bits$ with the total of $3n \cdot 1023$ $bits$. Analogously to ElGamal Baseline, the size of key storage increases linearly with the increase of the number of subgroups. In the Boneh-Franklin scheme, a node maximum storage of keys is greater than the maximum storage of keys in ElGamal Baseline by a factor of 3.

- In the new scheme, each node needs to store a public key n_i and a private key h^{n_i} for all the subgroups it belongs to. For n subgroups, a node stores only 2 keys of 2047 $bits$ each for the total of 4094 $bits$. From Table 4.2, we can observe that the maximum storage of keys for 1 node is constant for any number of subgroup. Only for $n = 2$, the ElGamal Baseline and Boneh-Franklin contain less key storage. The new scheme is better than the other schemes for $n = 5$ and above. For $n = 1k$, the maximum key storage size in the new scheme is 250 times smaller than the storage of ElGamal Baseline scheme. For $n = 100k$ the new scheme, the maximum

key storage size is smaller by factor of 75000 than the one in the Boneh-Franklin scheme.

In summary, for the key storage, we conclude that:

- The maximum key storage is ElGamal Baseline is three times smaller than the size of the one in the Boneh-Franklin scheme. In both cases, the key size depends on the number of nodes in the subgroup. In fact, the key size increases linearly with the increase of the number of nodes.
- In the new scheme, the maximum key storage size is smaller than the the size in the other two schemes. Beside being smaller, the size of key storage in the new scheme is constant.

Scheme	**Keys**	**Max storage of keys for 1 node in n subgroups (in bits)**						
		n	$n=2$	$n=5$	$n=100$	$n=1k$	$n=10k$	$n=100k$
Baseline ($p=1024\ bits$)	$\{x_{G_1},...,x_{G_n}\}$	$n\cdot 1023$ $bits$	2046 $bits$	5115 $bits$	102300 $bits$	1023000 $bits$	10230000 $bits$	102300000 $bits$
Boneh-Franklin [1] ($p=1024\ bits$)	$\{\alpha_{G_1},...,\alpha_{G_n}\}$, $\Gamma_i =< \gamma_{G_1},...,\gamma_{G_n} >^*$	$3n\cdot 1023$ $bits$	6138 $bits$	15345 $bits$	306900 $bits$	3069000 $bits$	30690000 $bits$	306900000 $bits$
New Scheme ($p=2048\ bits$)	n_i, h^{n_i}	4094 $bits$	4094 $bits$	4094 $bits$	4094 $bits$	4094 $bits$	4094 $bits$	4094 $bits$

Table 4.2: Maximum keys storage for 1 node in n subgroups.

4.2.5 Ciphertext Size

We analyze the ciphertext size of the three broadcast encryption schemes for u nodes in 1 subgroup. The results can be observed in Table 4.3.

- In the ElGamal Baseline scheme, the server broadcasts $\{g^r, v_1, ..., v_u\}$. Since we are working in a 1024 $bits$ groups, it translates to broadcasting 1024 $bits + ku \cdot$ 1024 $bits$ for a total of $(u+1)\cdot$1024 $bits$. In Table 4.3, we can observe that ciphertext size increases linearly with the increase of the number of subgroups.

[1] We only consider the minimal lower bound for key storage with 2 nodes per subgroup.

- In the Boneh-Franklin scheme, the server broadcasts $\{H_i^r,...,H_u^r,v\}$. In a 1024 $bits$ groups, this is equivalent to broadcasting $u \cdot 1024\ bits + 1024\ bits$ for a total of $(u+1) \cdot 1024\ bits$. Hence, as shown in Table 4.3, the ciphertext size increases linearly with the increase of the number of subgroups.

- In the new scheme, the server broadcasts $\{\prod_{i=1}^{u} n_i, g^r, v\}$. Since we are working in a 2048 $bits$ groups, it means that the server is broadcasting $3 \cdot 2048\ bits$ for a total of 6144 $bits$. The ciphertext size is constant independently of the number of subgroups. In contrast to the ElGamal Baseline and the Boneh-Franklin schemes, the new scheme offers constant size ciphertext. Moreover the ciphertext size in the new scheme is smaller than the one in the other two schemes for a number of subgroups greater than 5.

In summary, for ciphertext size, we conclude that:

- In the ElGamal Baseline and the Boneh-Franklin schemes, the ciphertext size increases linearly with the increase of the number of nodes.
- Besides offering smaller ciphertext size than the other two schemes, the new scheme enjoys a constant size ciphertext.

Scheme	**Ciphertext**	**Ciphertext size for u nodes in 1 subgroup (in bits)**						
		k	$u=2$	$u=5$	$u=100$	$u=1k$	$u=10k$	$u=100k$
Baseline ($p=1024\ bits$)	$\{g^r, v_1, ..., v_u\}$	$(u+1)\cdot 1024\ bits$	3072 $bits$	6144 $bits$	103424 $bits$	1034240 $bits$	10342400 $bits$	103424000 $bits$
Boneh-Franklin ($p=1024\ bits$)	$\{H_i^r,...,H_u^r,v\}$	$(u+1)\cdot 1024\ bits$	3072 $bits$	6144 $bits$	103424 $bits$	1034240 $bits$	10342400 $bits$	103424000 $bits$
New Scheme ($p=2048\ bits$)	$\{\prod_{i=1}^{u} n_i, g^r, v\}$	6144 $bits$	6144 $bits$	6144 $bits$	6144 $bits$	6144 $bits$	6144 $bits$	6144 $bits$

Table 4.3: Ciphertext size in 1 subgroup for k nodes.

4.3 Related Work

The concept of broadcast encryption was proposed by Fiat and Naor [Fiat and Naor, 1993] who presented a scheme that allows the sender to broadcast a message to all the users except the revoked ones (users with compromised keys). The security notion required that any coalition of users (up to a threshold) can not obtain any secret about the other users or the content of the broadcast. After that seminal work, several schemes were proposed to get good trade-offs between key storage cost and transmission cost [Boneh et al., 2005, Delerablée and andD. Pointcheval, 2007, Boneh et al., 2014, Agrawal and Yamada, 2020]. Further improvements were done in the context of multi-cast protocols [Canetti et al., 1999]. In these types of protocols, participants must agree on one or more keys to achieve confidentiality as well as authentication, with potentially many senders. For such settings, clients are no longer stateless (they must keep state beyond group information) but that allows protocols to cope with more diverse scenarios, like authentication in dynamic groups with or without group managers. For the case of confidentiality under single source broadcast, the scheme by Fiat and Naor [Fiat and Naor, 1993] provides a simple and efficient solution for the case of dynamic groups (ie. user revocation in [Canetti et al., 1999]). In terms of security notions, several schemes have been shown to achieve IND-CPA and IND-CCA [Boneh et al., 2005, Zhang et al., 2012, Yang, 2014, Chen et al., 2020]. BE can also be built on top of other primitives such as identity-based Encryption (IBE), which allows the sender to specify an arbitrary string as public key. This flexibility comes at the expense of requiring a central authority which, using a master key, can compute private keys for any identity. In *hierarchical identity-based encryption (HIBE)* [Horwitz and Lynn, 2002, Gentry and Silverberg, 2002], a collection of authorities is arranged in a organizational hierarchy (a tree). Any authority at level k in the hierarchy can issue private keys for any descendant in the hierarchy but cannot decrypt messages intended for identities. It turns out that HIBE schemes can be converted into public-key broadcast encryption schemes with a dynamic set of receivers (albeit with rather large ciphertext length [Naor et al., 2001, Boneh et al., 2005]). None of these constructions, however, seem to exploit the underlying hierarchy to BE schemes

with interesting properties among groups of receivers.

4.4 Conclusion

In this chapter, we investigated broadcast encryption schemes. We presented three existing schemes and discussed their security aspects. We also proposed a new broadcast encryption scheme based on ElGamal with constant size ciphertext and key storage. We implemented and compared, in terms of execution time, ciphertext and key storage space, three broadcast encryption schemes: the ElGamal Baseline scheme, the Boneh-Franklin scheme and the new scheme. Comparing to ElGamal Baseline, the new scheme is faster in key generation (given the fact that the public keys list is already pre-calculated) and encryption time but slightly slower in terms of decryption. Comparing to the Boneh-Franklin scheme, the new scheme results faster in the key generation, the encryption and the decryption time. Moreover, while in ElGamal Baseline and Boneh-Franklin the encryption time increases linearly with the increase of the number of nodes, the encryption time in the new scheme remains constant.

Concerning the key storage space, we showed that the new scheme contains a smaller size of keys with respect to the other two schemes, and enjoys a constant size key storage equivalent to 4094 even for $100k$ subgroups of nodes. For the ciphertext size, the new scheme is also better than the other two schemes. While in ElGamal Baseline and Boneh-Franklin schemes, the ciphertext size increases linearly with the increase of the number of subgroups, in the new scheme the ciphertext size remains constant and much smaller than in the other two schemes. For instance, the ciphertext in the new scheme for $10k$ subgroups is only 6144 $bits$, while in ElGamal Baseline and Boneh-Franklin scheme, the size is 10342400 $bits$. Since elliptic curve cryptography requires smaller key sizes, resulting in significant gains in speed and memory, it would be interesting to implement the broadcast schemes compared in this chapter over a prime-order subgroup G of an elliptic curve for comparison. We leave this as future work along with proving the *IND-CPA* security of the new scheme.

CHAPTER 5

TYPES FOR SECURE ARCHITECTURES IN DISTRIBUTED SYSTEMS

In this chapter, we propose two architectures for broadcasting messages to subgroups of nodes. Our idea is to map subgroups of nodes to levels in information flow security lattices to verify secure information flow in the server code. The purpose of the proposed architectures is to allow the server to securely communicate with a set of nodes in hierarchical distributed systems. We focus on implementing the idea via a type system for each architecture. Our main focus is to preserve, or more precisely, protect the confidentiality and integrity of the communications between server and nodes. We divide this chapter in two parts. In the first part, we present an architecture and a type system for broadcasting to nodes with static security levels. In the second part, we present a more complex architecture and type system in which we consider broadcasting software updates to nodes with dynamic security levels.

In Section 5.1, we introduce an architecture for broadcasting messages and define its model and the security properties we would like to achieve. We then present a server language with special syntax for broadcast encryption and for key generation. In Subsection 5.1.3, we present the semantics of our the language and in Subsection 5.1.4,

we develop a type system that enforces information flow security relying on the connection between BES privileged subgroups and security classes in distributed systems combined. Our typing rules enforce an appropriate usage of cryptographic primitives as broadcast encryption and keys. It verifies that cryptographic keys variables are not being altered. It also checks that messages designated to a privileged subgroup of nodes are not exposed to nodes with less authorization.

In Section 5.2, we introduce an architecture for broadcasting software updates and discuss its system model and the security properties we would like to enforce. In Subsection 5.2.2, we present a server language with syntax for broadcast encryption and syntax for remote attestation to verify the integrity of nodes. In Subsection 5.2.3 and Subsection 5.2.4, we give the semantics and the type system of our the language. Our type system enforces the integrity of nodes through remote attestation. Regarding the information flow policy, our type system captures insecure information flow between server and nodes belonging to different security classes. In particular, our typing rules ensure the update of variables holding information of nodes of the same security level, once the integrity of nodes is verified. Our main results comply with information flow security. Moreover, they guarantee that an attacker cannot distinguish between different executions, and therefore, its knowledge is not increased.

In Subsection 5.1.5, in our theorem, we express non-interference as preservation of confidentiality property and we parametrize it by an attacker observation level, and support our theorem with a soundness proof that we present in Appendix A. In addition to preserving confidentiality, in Subsection 5.2.5, we present a security property that preserves also integrity supported by a soundness proof in Appendix B. Both proofs are done by induction on the height of the typing derivation tree of $\Gamma \vdash p$. We conclude the chapter with related work and discuss conclusions and future work.

In summary, our contributions in Chapter 5 are:

- We consider two architectures: while the first architecture considers broadcasting messages to nodes with static security levels, the second architecture

considers broadcasting software updates to nodes with dynamic security levels.

- We propose two type systems. The first type system applies to a server language that features a cryptographic operation of broadcast encryption and one for generation of cryptographic keys to broadcast to a group of nodes with different security clearances. In addition to this, the second type system also considers integrity and endorsement. In particular, it monitors that variables containing information of nodes belonging to the same security level are updated only if the remote attestation succeeds.
- We support our work by soundness proofs for the two type systems, that ensure the compliance of type-checked server code with a formally defined information flow policy preserving confidentiality (for both type systems) and integrity of data (for the second type system).

5.1 Types for Broadcasting to Nodes with Static Security Levels

Security of distributed systems depends on protection mechanisms to ensure confidentiality of information traveling over an open network. Cryptography provides essential mechanisms for confidentiality. Cyptographic encryption schemes can provide strong security guarantees, such as IND-CPA or semantic security [Goldwasser and Micali, 1982]. However, even with plain encryption, the confidentiality of keys, plaintexts, and ciphertexts are interdependent [Rezk, 2018]: encryption with untrusted keys is clearly dangerous, and plaintexts should never be more secret than their decryption keys. Our proposal is that, to verify that information in the server flows to nodes with the appropriate clearances (e.g. verify the use of the correct encryption keys), we can map broadcast subgroups of nodes to levels in information flow security lattices. We implement this idea via a type system and provide a soundness proof with respect to a formally defined secure information flow property for server code.

5.1.1 An Architecture for Broadcasting Messages

We consider scenarios in which a server broadcasts messages with different confidentiality levels to nodes subgroups holding the appropriate clearance. We build on IND-CPA broadcast encryption schemes to preserve the message's confidentiality over a network and map subgroups of nodes to security lattices.

Security Lattices. In order to keep track of the confidentiality clearance of nodes, we use a mapping from nodes to security classes that we formalize as elements or levels in a lattice [Denning, 1976]. These security levels are used for messages so that the server can identify to which subgroups messages can be securely delivered. We use a lattice structure for modeling confidentiality, i.e. a partially ordered set together with least upper bound (join operator) and greatest upper bound (meet operator) on the set. In our architecture, the set of security classes is partially ordered by $\leq$ in a lattice $(\mathscr{L}, \leq)$. A confidentiality security level ranges over τ and indicates a read level. In the lattice, $\tau \leq \tau'$ means that τ' has more confidentiality than τ. We write $\bot$ for the lowest security level and $\top$ for the highest security level in lattice $\mathscr{L}$.

Example 11. *In this example, we show a confidentiality security lattice namely diamond lattice with three nodes placed as follows:* $L = \{n_0, n_1, n_2\}$, $M_1 = \{n_0\}$, $M_2 = \{n_1\}$ *and* $H = \{n_1\}$. *The partial order of the lattice is* $L \leq M_1 \leq H$ *and* $L \leq M_2 \leq H$. *IN the rest of the paper, we will be referring to the lattice in Figure 5.1.*

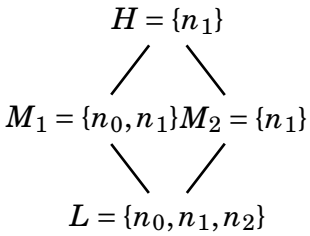

Figure 5.1: Security lattice

5.1.1.1 Models and Goals

This section covers the system model and the attacker model. We then informally introduce the security property that we want to provide.

System Model. We consider a framework where a single infrastructure provider, a server S, controls and manages a large set of networked nodes in a distributed system that we simply indicate as set of nodes N. Such framework is suitable for many IT systems in which the nodes can vary between smart cards, cloud systems and IoT systems. The server seeks to securely communicate with nodes belonging to privileged subgroups. Indeed, the server maps subgroups of nodes into security classes that we indicate as security levels.

Attacker Model. We assume passive attackers that listen to the network that the server uses to communicate with nodes. We also assume that an attacker can read all the public information but cannot prevent communications between a server and the nodes.

Security Properties We want to provide the following security properties:

- *Secure communication.* A server communicates with a specific node with confidentiality guarantees. In our architecture, confidentiality is ensured via the use of a IND-CPA [Rackoff and Simon, 1991] BES, the correct usage of cryptographic keys on the server for nodes at a given security level.
- *Secure information flow.* Information intended to more privileged security levels cannot flow into less privileged security levels. More specifically, messages with higher confidentiality clearances cannot be read by nodes with lower confidentiality clearances. This is ensured via a type system on the server code.

5.1.2 Syntax

In what follows, we consider the server language described below. It consists of programs, which can be expressions e or commands c.

$$
\begin{aligned}
(Programs)\ p &::= e \mid c \\
(Expressions)\ e &::= x \mid n \mid v \mid k \mid e \circ e' \\
(Commands)\ c &::= e := e' \mid c; c' \mid k := \mathsf{KG}(\{\mathsf{n}_1, ..., \mathsf{n}_\mathsf{i}\}, \mathsf{k}') \\
&\quad \mid \mathsf{while\ e\ do\ c} \mid \mathsf{if\ e\ then\ c\ else\ c'} \\
&\quad \mid \mathsf{sbroadcast}(\{\mathsf{n}_1, ..., \mathsf{n}_\mathsf{j}\}, \mathsf{k}, \mathsf{e})
\end{aligned}
$$

We let x, n, k range over variables and v ranges over strings, integers and boolean literals. We distinguish special variables $n, n_1, n_2, \ldots$ to designate nodes. We use $\circ$ for basic arithmetic and boolean operations.

Commands include standard statements (assign, sequence, if, while) and special statements (sbroadcast, KG). In the broadcast statement $\mathsf{sbroadcast}(\{\mathsf{n}_1, ..., \mathsf{n}_\mathsf{j}\}, \mathsf{k}, \mathsf{e})$, the server uses a broadcast encryption scheme to communicate a message e to a set of nodes (designated by $\{n_1, ..., n_j\}$) using an encryption key k. In the key generation statement $k := \mathsf{KG}(\{\mathsf{n}_1, ..., \mathsf{n}_\mathsf{i}\}, \mathsf{k}')$, the server uses a master key k' to generate the encryption key k for a set of nodes $\{n_1, ..., n_i\}$.

5.1.3 Semantics

A program is related to a memory μ which is a finite function that maps variables into values. We write $\mu[x := v]$ for the memory that assigns value v to a variable x. The semantics allows us to derive judgments of the form $\mu \vdash e \Rightarrow v$ for expressions and $\mu \vdash c \Rightarrow^t \mu'$ for commands. These judgments affirm that evaluating expressions e in memory μ results in literal v. Evaluating command c in memory μ results in a new memory μ' with t being a side effect that represents the command sends a message to the network. The semantics rules are given in Figure 5.2.

- In the *Base* rule, v evaluates itself.
- In the *Var* rule, the value v is applied to x.
- In the *Op* rule, e is evaluated to v and e' to v'; then the operation $e \circ e'$ between e and e' is evaluated into the operation $v \circ v'$.

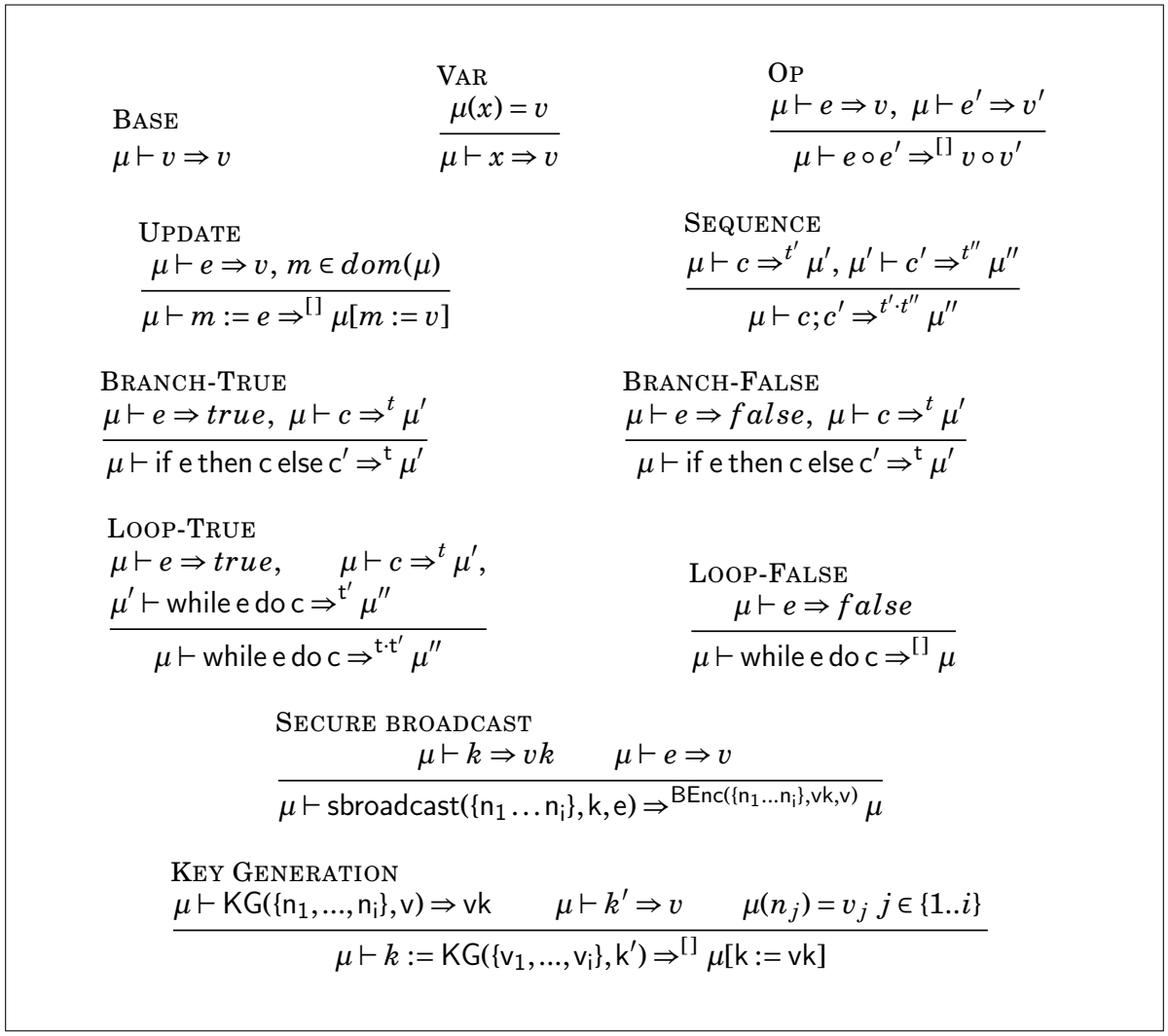

Figure 5.2: Semantics

- In the *Update* rule, e is evaluated in v; then since $m := e$, m is assigned v.
- In the *Sequence* rule, c is evaluated in μ' and c' is evaluated in μ''; then the sequence $c; c'$ is evaluated in μ''.
- In the *Branch-True* rule, e is true and c is evaluated in μ'; then the *if* statement is evaluated in μ'.
- In the *Branch-False* rule, e is false and c is evaluated in μ'; then the *if* statement is evaluated in μ'.

- In the *Loop-True* rule, e is true, c is evaluated in μ', *while* statement in μ' is evaluated in μ''; then the *while* statement in μ is evaluated in μ''.

- In the *Loop-False* rule, e is false; then the *while* statement is evaluated in μ.

- In the *Secure Broadcast* rule, the server takes as input a set of nodes $\{n_1,...,n_i\}$, the evaluation vk of key k in μ and the evaluation "m" of message e. To broadcast a message, the server employs a broadcast encryption scheme. We model the ciphertext that goes to the network by the annotation $\mathsf{BEnc}(\{\mathsf{n}_1,...,\mathsf{n}_i\},\mathsf{vk},\text{"m"})$.

- In the *Key Generation* rule, the server applies a key generation algorithm $\mathsf{KG}(\{\mathsf{n}_1,...,\mathsf{n}_i\},\mathsf{v})$, that takes in input a set of nodes $\{n_1,...,n_i\}$, the evaluation v of key k' in μ and outputs the encryption key vk.

5.1.4 Typing Rules

The types of the language are stratified as follows, where τ ranges over security levels from a confidentiality security lattice.

$$(Programs\ types)\ \rho ::= \tau \mid \tau\ var \mid \tau\ cmd \mid \tau\ Nvar(n) \mid \tau\ Kvar\ (n_1,\ldots,n_i) \mid \top\ MKvar$$

Type $\tau\ var$ is the type of a variable, $\tau\ cmd$ is the type of a command, and type $\top\ MKvar$ the type of master keys. Type $\tau\ Nvar(n)$ is the type of node variables. We write $\Gamma(n) = \tau\ Nvar(n1,...,n_i)$ to specify that τ is the maximum confidentiality clearance for node n. Type $\tau\ Kvar\ (n_1,\ldots,n_i)$ is the type of encryption keys, meaning that the encryption key is used for nodes $n_1,\ldots,n_i$, where each of these nodes have a minimal confidentiality clearance of τ.

The typing rules of our language are given in Figure 5.3. Typing judgments have the form: $\Gamma \vdash p : \rho$ where Γ is a typing environment mapping variables to variable security types from ρ. We write $\Gamma(x) = \rho$ to assign to the variable x type ρ.

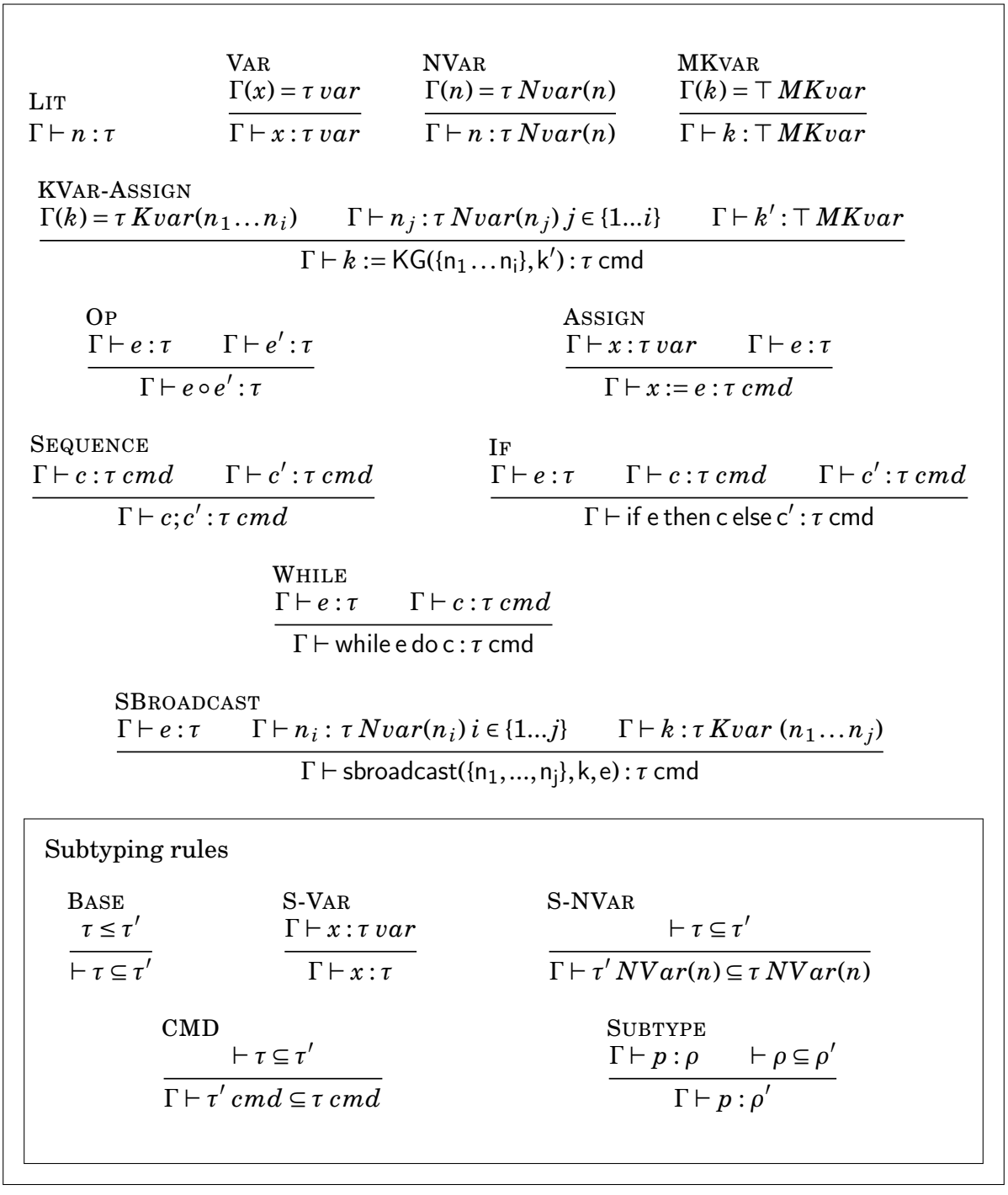

Figure 5.3: Typing rules

Our typing system includes standard rules (*Var, Assign, Sequence, If, While*) for secure information flow control [Volpano et al., 1996] and special rules (*Nvar, MKvar,*

Kvar-Assign, SBroadcast).

The *Var* rule binds the type $\tau\ var$ to a variable x. The *MKvar* rule binds the type $\top\ MKvar$ to master key variable k. The *SBroadcast* rule binds each node n_i to be of type $\tau\ Nvar(n_i)$ and e (the message to broadcast) of type τ. Moreover, it binds the key variable k to be of type $\tau\ Kvar(n_1 \ldots n_j)$. The *Kvar-Assign* rule binds each node n_j to be of type $\tau\ Nvar(n_j)$, k to be of type $\tau\ Kvar\ (n_1,\ldots,n_i)$ and the master key k' of type $\top\ MKvar$.

Notice that since the *Assign* rule requires x to be of type $\tau\ var$, it cannot be applied to a key variable. Hence our type system not only ensures secure information flow, but it also protects the use of keys, which ensure its integrity with respect to key generation.

The remaining rules of the type system constitute the subtyping logic and are given in the lower part of Figure 5.3. The *Base* rule states that τ is a subset of τ' if $\tau \leq \tau'$. The *Cmd* rule states that $\tau'\ cmd$ is a subset of $\tau\ cmd$ if τ is a subset of τ'. The *Subtype* rule states that a program p of type ρ can be bound to type ρ' if ρ is a subset of ρ'. The *S-Var* rule states that a variable x of type $\tau\ var$ can be bound to type τ, and the *S-NVar* rule states that $\tau'\ NVar(n)$ is a subset of $\tau\ Nvar(n)$ if τ is a subset of τ'.
In what follows, we display some examples and show how our type system can type secure programs or catch insecure ones. We consider the confidentiality lattice in Figure 5.1, where $L \leq M_1 \leq H$ and $L \leq M_2 \leq H$.

Example 12. *We consider a server willing to send a message m:* $\Gamma(m) = M_1\ var$ *to a set of nodes in* $\{n_0, n_1\}$ *such that* $\Gamma(n_0) = M_1\ NVar(n_0)$ *and* $\Gamma(n_1) = H\ NVar(n_1)$ *. We also consider a master key* k'*:* $\Gamma(k') = \top\ MKVar$ *and a key k:* $\Gamma(k) = M_1\ KVar(n_0, n_1)$*.*

We show that the following program p is typable in Γ*:*

$$k := \mathsf{KG}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{m}, \mathsf{k}'); \ \mathsf{sbroadcast}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{k}, \mathsf{m})$$

This program consists of a sequence of two programs p_1 *and* p_2*. To type* p_1*, we apply the KVar-Assign rule. This rule checks (a) the type of nodes* n_0 *and* n_1*, (b) the type of the master key* k'*, and (c) the type of the encryption key k for nodes* n_0 *and* n_1

such that $\Gamma(k) = M_1\ KVar(n_0, n_1)$. *In (a), we apply* ($a_1$) *the NVar rule that binds* n_0 *to* $\Gamma(n_0) = M_1\ NVar(n_0)$ *and* (a_2) *the subtype rule that binds* n_1 *to* $\Gamma(n_1) = M_1\ NVar(n_1)$.

$$\dfrac{\dfrac{\Gamma(n_1) = H\ NVar(n_1)}{\Gamma \vdash n_1 : H\ NVar(n_1)} \quad (a_2) \qquad \dfrac{\vdash M_1 \subseteq H}{\Gamma \vdash H\ NVar(n_1) \subseteq M_1\ NVar(n_1)}}{\Gamma \vdash n_1 : M_1\ NVar(n_1)}$$

Since all the constraints are valid, then p_1 *is typable.*

To type p_2, *we apply the SBroadcast. The SBroadcast rule checks (a) the type of the message to broadcast m, (b) the type of the nodes variables* n_0, n_1, *and (c) that the type of the encryption k corresponds to the type of nodes variable. In (a), we apply the S-Var then the Var rule that bind m to* $\Gamma(m) = M_1\ Var$. *Concerning (b), we apply the Nvar rule on nodes* n_0 *and* n_1 *that check their types ((b) for* n_0 *and* (b') *for* n_1). *In (c), the rule binds the encryption key to be of type* $M_1\ KVar(n_0, n_1)$.

$$\dfrac{\dfrac{\dfrac{\Gamma(m) = M_1\ Var}{\Gamma \vdash m : M_1\ Var}\ (a)}{\Gamma \vdash m : M_1} \qquad \dfrac{\Gamma(n_0) = M_1\ NVar(n_0)}{\Gamma \vdash n_0 : M_1\ NVar(n_0)}\ (b) \qquad \dfrac{\Gamma(n_1) = M_1\ NVar(n_1)}{\Gamma \vdash n_1 : M_1\ NVar(n_1)}\ (b')}{\Gamma \vdash \mathsf{sbroadcast}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{k}, \mathsf{m}) : \mathsf{M}_1\ \mathsf{cmd}}\ (c)\ (\textsc{SBroadcast})$$

Since all the constraints are valid, then p_2 *is typable and the sequence* $p_1; p_2$ *is typable.*

Example 13. *In contrast with Example 12, we consider a server willing to send a message m to a set of nodes in* $\{n_0, n_1\}$ *but generates a key also for a node* n_2. *We show that the following program is not typable:*

$$k := \mathsf{KG}(\{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2\}, \mathsf{k}');\ \mathsf{sbroadcast}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{k}, \mathsf{m})$$

In fact, the server program contains an error in the key generation (generating key also for n_2), which leads to a confidentiality problem since n_2 will also be able to read a message intended to n_0 and n_1 only. Hence, such error is detected by our type system.

Example 14. *In this example, we consider a server willing to send a message m: $\Gamma(m) = L\ var$ to a set of nodes $\{n_0, n_1, n_2\}$ but does not generate a key for node n_2. We show that the following program is not typable:*

$$k := \mathsf{KG}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{k}'); \ \mathsf{sbroadcast}(\{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2\}, \mathsf{k}, \mathsf{m}')$$

The server program contains an error in the key generation (does not generate a key for n_2), which leads to a problem since n_2 will not be able to decrypt. Therefore, our type systems detects such error.

Example 15. *In this example, we consider a server willing to send a message m': $\Gamma(m') = L\ var$ to a set of nodes $\{n_0, n_1, n_2\}$. We show that the following program is not typable:*

$$k := \mathsf{KG}(\{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2\}, \mathsf{k}'); \ \mathsf{m}' := \mathsf{m}; \ \mathsf{sbroadcast}(\{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2\}, \mathsf{k}, \mathsf{m}')$$

This program consist of a sequence of three programs p_1, p_2 and p_3. Despite the fact that the p_1 and p_3 are typable and the broadcast in p_3 seems to be secure; p_2 is not typable since it assigns a high value $M_1\ var$ to a lower value $L\ var$ and therefore, the entire program is not typable. Indeed, our type system detects this error, otherwise, the broadcast may leak confidential information $M_1\ var$ to nodes with lower confidentiality clearance $L\ Nvar(n_i), i \in \{0,1,2\}$.

5.1.4.1 Limitations of the Type System

In this section, we show some examples to highlight on the limitations of our type system. We consider a server willing to send a message to a set of nodes $\{n_0, n_1\}$. We also consider a master key $k' : \Gamma(k') = \top\ MKVar$ and a key $k : \Gamma(k) = M_1\ KVar(n_0, n_1)$. For the initial configuration, we let $\mu(k') = 3$ and $\mu(k) = 0$. To encrypt in a correct way, the server should generate the correct keys corresponding to the intended nodes. Ideally, a type system handling cryptographic primitives should detect any violation of the hypothesis of the

type system. Our type system is capable of detecting violations in some cases as in Example 16 and Example 17, but not in other cases as in Example 18.

We rely on this initial configuration to show how our system can be limited in certain cases.

Example 16.

$$p_1 \stackrel{\Delta}{=} k := \mathsf{KG}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{k}'); \ \mathsf{sbroadcast}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{k}, \mathsf{m})$$

The program p_1 is correct since the server first generates an encryption key for nodes n_0 and n_1, then broadcasts a message to those nodes with the correct key. The program is typable by our type system.

Example 17.

$$p_2 \stackrel{\Delta}{=} k := \mathsf{KG}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{k}'); \ \mathsf{k} = 0; \ \mathsf{sbroadcast}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{k}, \mathsf{m})$$

The program p_2 is wrong since the server first generates an encryption key for nodes n_0 and n_1 but then uses the key $k = 0$ to broadcast a message to those nodes. The server is using the wrong key to broadcast the message and therefore it cannot broadcast the message to the nodes. The program is not typable by the type system.

Example 18.

$$p_3 \stackrel{\Delta}{=} \mathsf{sbroadcast}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{k}, \mathsf{m})$$

The program p_3 is wrong since the server is broadcasting a message to nodes n_0 and n_1 without generating the right keys. However, since k have type $M_1\ KVar(n_0, n_1)$ in the initial configuration, our type system types this program despite the fact that the value in the initial memory is equal to zero. Unfortunately, our type system is limited in such cases since it is not able to detect such errors.

5.1.5 Security Properties

We define confidentiality (secure information flow between nodes of different security levels) as a new noninterference property [Sabelfeld and Myers, 2003].

We parametrize the definition by an attacker observation level, τ. For our security definition, it is useful to define an equivalence between memories. Intuitively, the definition $\tau - Equal$ memories states that two memories are equal from the view point of an attacker that can observe only parts of the memory with security level less or equal than τ.

Definition 24 ($\tau - Equal$ Memories)**.** *Two memories μ_0, μ_1 are $\tau - Equal$ for Γ, written $\mu_0 =_\tau^\Gamma \mu_1$,* iff $dom(\mu_0) = dom(\mu_1) \wedge \forall x \in \mu_0$ *such that* if $\Gamma(x) = \tau'$ var $\wedge\, \tau' \leq \tau$, *then* $\mu_0(x) = \mu_1(x)$.

We say that two memories μ_0 and μ_1 are $\tau - Equal$, $\mu_0 =_\tau^\Gamma \mu_1$ if they contain the same variables that have value less or equal than τ. Two memories are $\tau - Equal$ if the mapping of the same variables of the same type τ or lower have the same value in both memories. This definition considers only variables of type $\tau\ var$.

For our security property, we also need to consider messages that are sent to different groups of nodes. In the semantics of server programs, messages that are broadcasted are given as traces that parametrize the semantics relation. We define a filtering function to project parts of the broadcasted messages that are sent to nodes mapped to security levels less or equal than the attacker observation level.

Definition 25 (Filtering)**.** *Let* $\mathsf{filter}_\tau(t) = \mathsf{filter}'_\tau(t, [\,])$ *and* $\mathsf{filter}'_\tau([\,], t) = t$.

$$\mathsf{filter}'_\tau(\mathsf{BEnc}(\{n_1,...,n_k\}, vk, v') \cdot t', t) = \begin{cases} \mathsf{filter}'_\tau(t', t \cdot (\{n_1,...,n_k\}, v')) & \text{if } \Gamma(n_1) \sqcap \Gamma(n_2) ... \sqcap \Gamma(n_k) \leq \tau \\ \mathsf{filter}'_\tau(t', t) & Otherwise \end{cases}$$

We are now ready to formalize secure information flow in our architecture. Intuitively this definition states that for an attacker that observes memories and messages on the network at τ, the system starting with $\tau - Equal$ memories will lead to memories which are $\tau - Equal$. Thus, the attacker cannot distinguish the executions and will broadcast the same messages to nodes less or equal than τ in both executions.

Definition 26 (NonInterference)**.** *A server program p is NI at τ for Γ, written $NI^{\Gamma}_{\tau}(p)$,* iff $\forall \mu_0, \mu_1$ *such that* $\mu_0 =^{\Gamma}_{\tau} \mu_1 \wedge \mu_0 \vdash p \Rightarrow^{t_0} \mu'_0 \wedge \mu_1 \vdash p \Rightarrow^{t_1} \mu'_1$*, then* $\mu'_0 =^{\Gamma}_{\tau} \mu'_1 \wedge \mathsf{filter}_{\tau}(\mathsf{t}_0) = \mathsf{filter}_{\tau}(\mathsf{t}_1)$.

Our main result follows: a well-typed program is noninterferent at τ for Γ.

Theorem 2. *Let Γ be a typing environment and $\tau \in \mathscr{L}$ a security label. If $\Gamma \vdash p$ then p is NI^{Γ}_{τ}.*

We prove this theorem by induction on the height of the typing derivation tree of $\Gamma \vdash p$ in Appendix A.

Example 19. *Concerning the security property, we show that the program p of Example 12 is noninterferent at M_1. We let two memories $\mu_0 = \{k := vk_0, k' := vk'_0, n_0 = "n_0", n_1 = "n1", m := v\}$ and $\mu_1 = \{k := vk_1, k' = vk'_1, n_0 = "n_0", n_1 = "n1", m := v\}$. Notice that $\mu_0 =^{\Gamma}_{M_1} \mu_1$. By Definition 24, $\mu_0 =^{\Gamma}_{M_1} \mu_1$ since only variables of type τ var (in our example m) should be equal. After executing p, the value of m does not change in the resulting memories μ'_0 and μ'_1. Since only variables of type τ var are considered, then $\mu'_0 =^{\Gamma}_{M_1} \mu'_1$. In order to satisfy the full hypothesis in Definition 26, we need to prove that $\mathsf{filter}_{\mathsf{M}_1}(\mathsf{t}_0) = \mathsf{filter}_{\mathsf{M}_1}(\mathsf{t}_1)$. Actually, t_0 and t_1 are the traces of the messages broadcasted over the network through the Secure Broadcast rule in Figure 5.2:* $\mathsf{BEnc}(\{\mathsf{n}_1, ..., \mathsf{n}_k\}, \mathsf{k}, \mathsf{v})$*. For an execution that starts with μ_0, $t_0 = \mathsf{BEnc}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{vk}_3, \mathsf{v})$, thus $\mathsf{filter}_{\mathsf{M}_1}(\mathsf{t}_0) = [\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{v}]$. For an execution that starts with μ_1, $t_1 = \mathsf{BEnc}(\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{vk}_4, \mathsf{v})$ and $\mathsf{filter}_{\mathsf{M}_1}(\mathsf{t}_1) = [\{\mathsf{n}_0, \mathsf{n}_1\}, \mathsf{v}]$. Hence, $\mathsf{filter}_{\mathsf{M}_1}(\mathsf{t}_0) = \mathsf{filter}_{\mathsf{M}_1}(\mathsf{t}_1)$ and p is $NI^{\Gamma}_{M_1}$.*

Example 20. *We show that the program p of Example 15 does not comply with NI^{Γ}_{L} according to Definition 26. Let two memories $\mu_0 = \{k := vk_3, k' := vk'_3, n_0 = "n_0", n_1 = "n1", m := 4, m' := 2\}$ and $\mu_1 = \{k := vk_4, k' = vk'_4, n_0 = "n_0", n_1 = "n1", m := 5, m' := 2\}$. The memories μ_0 and μ_1 are L-Equal since the value of m' is the same in both memories, and m' is the only variable of type L var. The resulting memories of the execution of p are $\mu'_0 = \{k := vk_3, k' := vk'_3, n_0 = "n_0", n_1 = "n1", m := 4, m' := 4\}$ and $\mu'_1 = \{k := vk_4, k' = vk'_4, n_0 =$*

"n_0", n_1 = "$n1$", $m := 5, m' := 5$}. *Notice that the final memories* μ_0' *and* μ_1' *are not L-Equal since the value of* m' *is different, and* $\mathsf{filter}_{\mathsf{L}}(\mathsf{t}_0) = [\{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2\}, 4] \neq \mathsf{filter}_{\mathsf{L}}(\mathsf{t}_1) = [\{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2\}, 5]$. *Therefore p is not* NI_L^{Γ}.

5.2 Types for Broadcasting to Nodes with Dynamic Security Levels

Cryptographic protocols ensure confidentiality and integrity of information in distributed systems. The security of distributed systems, in some cases, depends also on constantly updating software with patches to deal with known vulnerabilities. Software updates play a crucial role in the life-cycle management of a node in a system. Regular software update processes are needed to guarantee the appropriate functioning of a node. A secure software update process has to prevent attempts of an attacker to inject malicious code in the software update image and therefore alter the behavior of an update, and has to be confidentiality protected [IETF, 2017]. Since the same software update may be distributed to several nodes, key management and decision-making is challenging on the server side. Having different necessities and vulnerabilities, nodes on the same network require different software updates at different times. Performing distribution of software updates in a secure way is a non-trivial task. In fact, the server needs to classify nodes into subgroups in order to ensure that only the intended devices receive the appropriate software updates. Furthermore, the server needs to encrypt software updates to prevent an attacker from injecting malicious code and compromising the software image [IETF, 2017], and needs to verify which nodes successfully installed the software updates [**?**] to keep a record of such nodes. These requirements push the server to take decisions that not only can be error prone, but can also compromise the security of the system. Hence, verification process is required in order to ensure the correctness of the entire process of software updates.

We build on broadcast encryption schemes and remote attestation protocols to propose an architecture to securely deliver software updates in hierarchical distributed systems.

Our goal is to allow for automate decision making at the server side while preserving confidentiality and integrity of all communication. We propose a type system to control that information securely flows from the server to the nodes belonging to different security classes, and demonstrate its security via a proof with respect to a well- defined secure information flow policy for server code fitting our architecture.

5.2.1 An Architecture for Software Updates

We consider a scenario in which a server broadcasts messages to nodes with different levels of confidentiality. Moreover, nodes may require software updates and are also classified with integrity levels according to their installed software. Thus, the server needs to keep track of subgroups of nodes classified according to their confidentiality and integrity levels.

Remote Attestation Protocols. Remote attestation protocols [Sailer et al., 2004, Shaneck et al., 2005, Banks et al., 2021] allow a sender (trusted party) to verify the integrity of a software running on a remote device. Remote attestation is performed over the network, where the sender (server) and the receiver (node) communicate directly with each other. Loading a software update to a node may not be completely trustworthy since an attacker can alter the software amid the loading process [Francillon, 2009]. In our design, we apply a remote attestation protocol to verify the integrity of software running on a node. A common approach to implement remote attestation is a challenge-response protocol.

Security Lattices. We use a lattice $\mathscr{L}$, a product of two lattices, a lattice of confidentiality levels $(\mathscr{L}_{\mathscr{C}}, \leq_C)$ and a lattice of integrity levels $(\mathscr{L}_{\mathscr{I}}, \leq_I)$. A confidentiality security level is written as s_C and indicates a read level while an integrity security level is written as s_I and indicates a write level. In the confidentiality lattice, $s_C \leq_C s'_C$ means that s'_C has more confidentiality than s_C. In the integrity lattice, $s_I \leq_I s'_I$ means that s_I has more

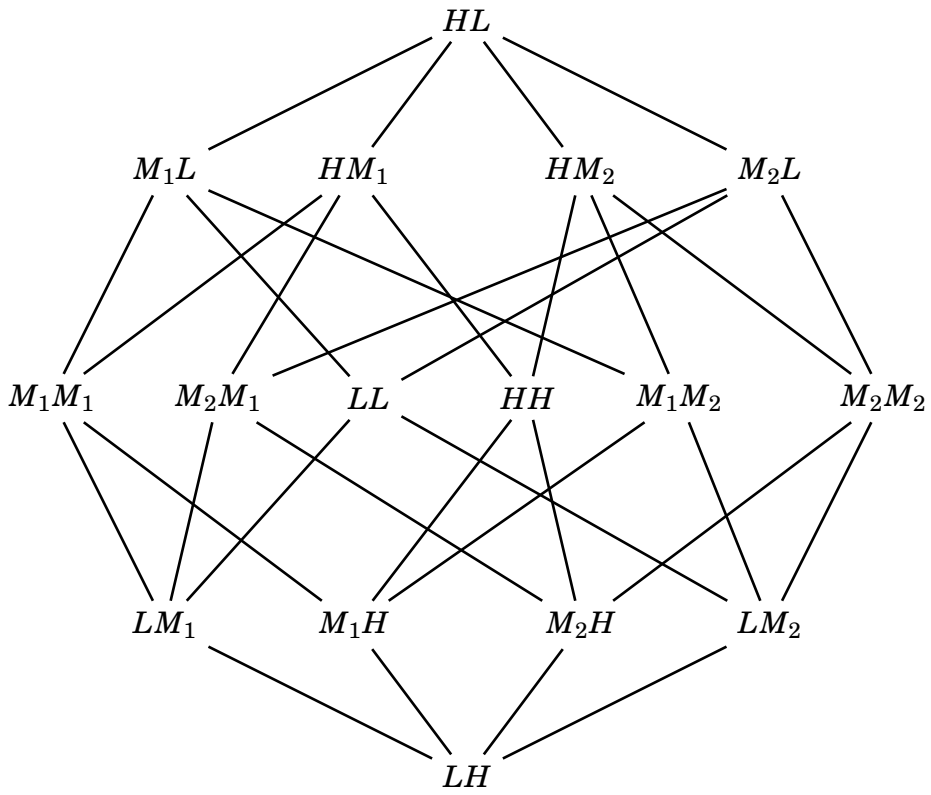

Figure 5.4: A diamond lattice of security levels.

integrity than s'_I. We write $\bot$ for the lowest security level and $\top$ for the highest security level in lattice $\mathscr{L}$. In Figure 5.4, we show a confidentiality and integrity security lattice.

5.2.1.1 Models and Goals

In this section, we present the system model and the security properties of the proposed architecture.

System Model. We consider a server that controls and manages a large set of networked nodes in a distributed system. The server aims to communicate securely with nodes belonging to privileged subgroups and maps subgroups of nodes into security classes. Subsequently, the server remote attest nodes and moves them to higher integrity security levels whenever the verification of the installed software update succeeds.

Attacker Model. We assume that an attacker manipulates the communication network that the server uses to communicate with nodes. An attacker can modify and inject messages over the network, or can mount man-in-the-middle attacks [Conti et al., 2016] or replay attacks [Syverson, 1994]. We also assume that an attacker can exploit software vulnerabilities and load malicious content to nodes. Finally, an attacker can access public information without being able to inhibit communications between a server and the nodes during an update nor make the nodes unavailable (e.g. resource exhaustion) [Zandberg et al., 2019].

Security Properties. The security properties needed for our architecture are the following:

- *Security level integrity.* The system guarantees that each node increases its integrity level after that a successful software update is attested. Moreover, a node at an integrity level cannot be tricked by the attacker to install an old software update. In our architecture, this is ensured via the protection of variables mapping security classes to nodes in the server and via a counter that is incremented with each communicated message to avoid replay attacks.

- *Secure communication.* A server communicates with a specific node with confidentiality, integrity, and freshness guarantees. Confidentiality and non-malleability of the ciphertexts are ensured via the use of a IND-CCA [Rackoff and Simon, 1991] BES, the correct usage of cryptographic keys on the server for nodes at a given security level; and integrity is ensured via the use of remote attestation protocols, the fact that only the servers can broadcast messages, and a protocol for security level integrity.

- *Secure information flow.* Information intended to more privileged security levels cannot flow into less privileged security levels. For confidentiality, messages with higher confidentiality clearances cannot be read by nodes with lower confidentiality clearances. As for integrity, software versions are mapped to integrity levels and

nodes only receive software updates from the server corresponding to their integrity level. Secure information flow is guaranteed through a type system on the server code.

5.2.2 Syntax

In what follows, we consider the server language described below. It consists of programs, which can be expressions e or commands c.

$$
\begin{aligned}
(Programs)\ p &::= e \mid c \\
(Expressions)\ e &::= x \mid L \mid K \mid pc \mid v \mid e \circ e' \\
(Commands)\ c &::= e := e' \mid c; c' \\
&\quad \mid \mathsf{while\ e\ do\ c} \mid \mathsf{if\ e\ then\ c\ else\ c'} \\
&\quad \mid \mathsf{for\ n} \in \mathsf{L\ endorse\ Ra(L, L', n)} \\
&\quad \mid \mathsf{sbroadcast(L, e, K)}
\end{aligned}
$$

We let x, K, L, pc range over variables and v ranges over strings, integers and Boolean literals. We distinguish special variables $\{L_1, L_2, L_3, ...\}$ to designate groups of nodes in a broadcast command. In our server language, we use special variable L to keeps set of nodes identifiers belonging to a common security level, K for cryptographic keys, and pc for counters. We use $\circ$ to compute basic arithmetic and Boolean operations.

Commands include standard statements (*assign, sequence, if, while*) and special statements (*Sbroadcast, Endorse Ra*). We only highlight on the special statements and we refer the reader to see Subsection 5.1.2 for the standard statements.

- The *Endorse Ra* statement $\mathsf{for\ n} \in \mathsf{L\ endorse\ Ra(L, L', n)}$ iterates over every node $n \in L$ to run a remote attestation protocol to decide whether to move n to another set of nodes L' (representing a more privileged security level) and remove it from L if the verification succeeds, or leave it in the same security level L if the verification fails.

- In the *Sbroadcast* statement sbroadcast(L, e, K), the server uses a broadcast encryption scheme to communicate a message e' to a set of nodes (designated by variable L) using a key K.

5.2.3 Semantics

A program is related to a memory μ which is a finite function that maps variables into values. We write $\mu[x := v]$ for the memory that assigns value v to a variable x. The semantics allows us to derive judgments of the form $\mu \vdash e \Rightarrow v$ for expressions and $\mu \vdash c \Rightarrow^t \mu'$ for commands. These judgments affirm that evaluating expressions e in memory μ results in literal v. Evaluating command c in memory μ results in a new memory μ' with t being a side effect that represents the command sends a message to the network. The semantics rules are given in Figure 5.5.

We only highlight on the *Secure Broadcast* rule and the *Endorse-Ra* rule. We refer the reader to see Subsection 5.1.3 for the other rules.

- In the *Secure Broadcast* rule, the server applies a key generation algorithm that takes in input a set of nodes {"n_1","n_2",...,"n_n"} $\in L$, the evaluation v' of K in μ and outputs a keys vk, sk: Key generation({"n_1","n_2",...,"n_n"}, v′). To broadcast a message, the server employs a broadcast encryption scheme. We model the ciphertext that goes to the network by the annotation BEnc(L, vk, "m"||"v").

- In the *Endorse-Ra* rule, for each node stored in the memory of variable L, the server applies a remote attestation protocol. If the remote attestation succeeds, it outputs a set of nodes S. This set of nodes is then moved into L' which represents a more privileged security level than L.

BASE
$$\mu \vdash v \Rightarrow v$$

VAR
$$\frac{\Gamma(x) = v}{\mu \vdash x \Rightarrow v}$$

OP
$$\frac{\mu \vdash e \Rightarrow v,\ \mu \vdash e' \Rightarrow v'}{\mu \vdash e \circ e' \Rightarrow^{[]} v \circ v'}$$

UPDATE
$$\frac{\mu \vdash e \Rightarrow v,\ m \in dom(\mu)}{\mu \vdash m := e \Rightarrow^{[]} \mu[m := v]}$$

SEQUENCE
$$\frac{\mu \vdash c \Rightarrow^{t'} \mu',\ \mu' \vdash c' \Rightarrow^{t''} \mu''}{\mu \vdash c; c' \Rightarrow^{t' \cdot t''} \mu''}$$

BRANCH-TRUE
$$\frac{\mu \vdash e \Rightarrow true,\ \mu \vdash c \Rightarrow^{t} \mu'}{\mu \vdash \text{if e then c else c}' \Rightarrow^{t} \mu'}$$

BRANCH-FALSE
$$\frac{\mu \vdash e \Rightarrow false,\ \mu \vdash c \Rightarrow^{t} \mu'}{\mu \vdash \text{if e then c else c}' \Rightarrow^{t} \mu'}$$

LOOP-TRUE
$$\frac{\mu \vdash e \Rightarrow true,\quad \mu \vdash c \Rightarrow^{t} \mu',\quad \mu' \vdash \text{while e do c} \Rightarrow^{t'} \mu''}{\mu \vdash \text{while e do c} \Rightarrow^{t \cdot t'} \mu''}$$

LOOP-FALSE
$$\frac{\mu \vdash e \Rightarrow false}{\mu \vdash \text{while e do c} \Rightarrow^{[]} \mu}$$

SECURE BROADCAST
$$\frac{vk = KG(\{"n_1","n_2",...,"n_n"\}, v')\ \ \mu \vdash K \Rightarrow v'\ \ \mu \vdash e \Rightarrow "m"\quad \mu(L) = \{"n_1","n_2",...,"n_n"\}\ \ \mu \vdash \text{pc} \Rightarrow \text{v}}{\mu \vdash \text{sbroadcast(L, e}||\text{pc, K)} \Rightarrow^{\text{BEnc(L,vk,"m"||"v")}} \mu}$$

ENDORSE-RA
$$\frac{\mu(L) = \{"n_0","n_1",...,"n_n"\}\ \ \mu(L') = \{"n'_0","n'_1",...,"n'_n"\}\ \ \mu \vdash \text{Ra(L)} \Rightarrow \text{S}\ \ \text{S} \subseteq \{"\text{n}_0",...,"\text{n}_\text{n}"\}}{\mu \vdash \text{for n} \in \text{L endorse Ra(L, L}', \text{n)} \Rightarrow \mu'[\text{L} := \text{L} \setminus \text{S}; \text{L}' := \text{L}' \cup \text{S}]}$$

Figure 5.5: Semantics

5.2.4 Typing Rules

The types of the language are stratified as follows.

$$
\begin{aligned}
(Data\ types)\ \tau &::= (s_C, s_I) \\
(Programs\ types)\ \rho &::= \tau \mid \tau\ var \mid \tau\ cmd \mid \tau\ Lvar \\
&\quad \mid \top\ Kvar \mid \bot\ Cvar
\end{aligned}
$$

Metavariables (s_C, s_I) range over the set of security classes partially ordered by $\leq$ in the product lattice $(\mathscr{L}, \leq)$ as described in Subsection 4.4.2. Hence, s_I represents the integrity level in $(\mathscr{L}_{\mathscr{I}}, \leq_I)$ and s_C represents the confidentiality level in $(\mathscr{L}_{\mathscr{C}}, \leq_C)$.

- Type $\tau\ var$ is the type of a variable.
- Type $\tau\ cmd$ is the type of a command.
- Type $\tau\ Lvar$ is the type for the security levels variables L.
- Type $\top\ Kvar$ is the type of keys variables K. Keys are placed in the top of the lattice with the highest confidentiality as only the server possesses the keys.
- The type $\bot\ Cvar$ designates the type of counters. Counters are used to enforce the integrity of our protocol. For that, they are placed in the bottom of the lattice with the highest integrity and the lowest confidentiality as only the server can increment the value of pc and everyone can read it.

The typing rules of our language are given in Figure 5.6. Typing judgments have the form: $\Gamma \vdash p : \rho$ where Γ is a typing environment mapping variables to variable security types from ρ. We write $\Gamma(x) = \rho$ to assign to the variable x type ρ.

Our typing system includes standard rules (*Var, Assign, Sequence, If, While*) for secure information flow control [Volpano et al., 1996] and special rules (*Lvar, Kvar, Assign-Counter, SBroadcast, Remote Attestation*).

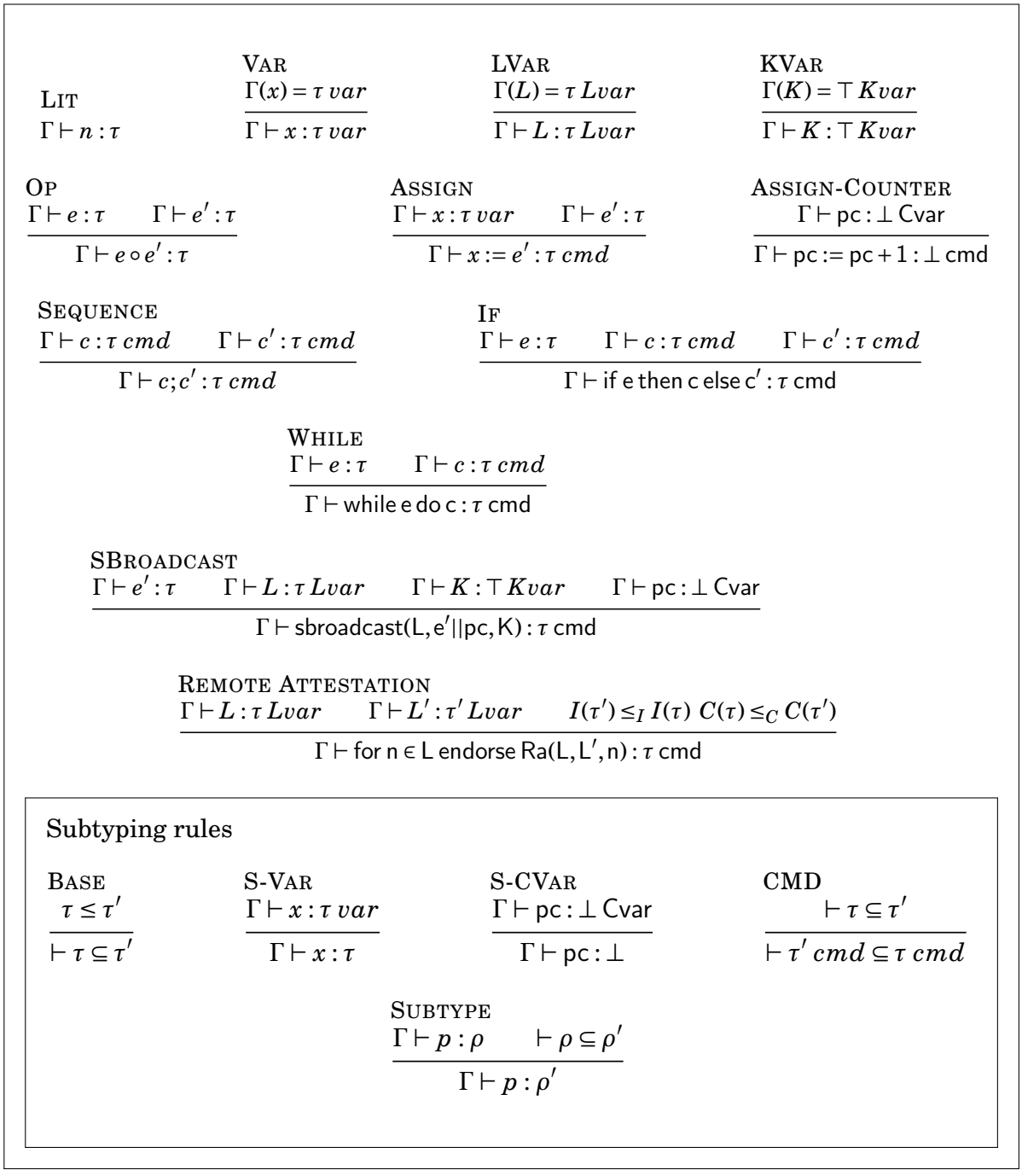

Figure 5.6: Typing rules

- The *Var* rule binds the type $\tau\ var$ to a variable x.

- The *LVar* rule binds the type $\tau\ Lvar$ to security levels variable L.

- The *KVar* rule binds the type $\top\, Kvar$ keys variable K.
- The *Op* rule binds e and e' to be of type τ.
- The *Assign* rule says that, in order to ensure a secure flow from e' to x, e' and x must agree on their security levels, which is conveyed by τ appearing in both hypotheses of the rule.
- The *Assign-Counter* rule binds pc to be of type $\bot\, CVar$ since only the server can write this variable and everyone can read it. Notice that since the *Assign* rule requires x to be of type $\tau\ var$, it cannot be applied to a variable of type $\bot\, Cvar$. Thus, for a program with an assignment to pc to be typable, such assignment can be only of the shape $\mathsf{pc} := \mathsf{pc} + 1$. This restricts the use of $\bot\, Cvar$ variable in a program. Hence our type system not only ensures secure information flow (see Section 5.4), but it also protects the use of counters, which ensure secure level integrity.
- The *Sequence* rule says that, in order to execute c then c', both c and c' must agree on the same type of command $\tau\ cmd$.
- The *If* rule says that, in order to execute the *if* command, e, c and c' must agree on their security levels τ.
- The *While* rule says that, in order to execute the *while* command, e and c and c' must agree on their security levels τ.
- The *SBroadcast* rule binds L to be of type $\tau\, Lvar$ and e' (the message to broadcast) of type τ. Moreover, it binds the keys variable K to be of type $\top\, Kvar$ and the counter variable of type $\bot\, Cvar$. A broadcast program is well-typed if the type of the message e' is $\leq$ than the type of L to secure the flow of information and avoid that lower security levels receive higher security level messages.
- The *Remote Attestation* rule requires L to be of type $\tau\, Lvar$ and L' of type $\tau'\, Lvar$ where $\tau \leq \tau'$. Furthermore, it checks if $I(\tau') \leq_I I(\tau)$ and $C(\tau) \leq_C C(\tau')$. This means

that in order to move a node from L to L', L' should have higher integrity level and at least the same confidentiality level.

The remaining rules of the type system constitute the subtyping logic and are given in the lower part of Figure 5.6.

- The *Base* rule states that τ is a subset of τ' if $\tau \leq \tau'$.
- The *Cmd* rule states that $\tau'\ cmd$ is a subset of $\tau\ cmd$ if τ is a subset of τ'.
- The *Subtype* rule states that a program p of type ρ can be bound to type ρ' if ρ is a subset of ρ'.
- The *S-Var* rule states that a variable x of type $\tau\ var$ can be bound to type τ.
- The *S-Cvar* rule states that a counter variable of type $\bot\ Cvar$ is bound to type $\bot$.

In what follows, we display some examples and show how our type system can type secure programs or catch insecure ones. We consider the confidentiality lattice in Figure 5.4.

Example 21. *We consider the diamond lattice in Figure 5.4, in which $HH \leq HM_2$ and $M_1M_2 \leq HM_2$ where $H \leq_I M_2$ which means that H has more integrity than M_2. We also consider a server willing to send a software update $SU_{M_1M_2}$ of type $M_1M_2\ var$ to a set of nodes in L_{HM_2} and upgrade the security level of node $n \in L_{HM_2}$ to L_{HH} if it succeeds to install correctly its software update. We finally consider a counter,* pc *of type $\bot\ Cvar$ and a key, K of type $\top\ Kvar$.*

We show that the following program p is typable:

$$\mathsf{pc := pc + 1;\ sbroadcast}(\mathsf{L_{HM_2}}, \mathsf{SU_{M_1M_2}}||\mathsf{pc}, \mathsf{K});$$
$$\mathsf{for\ n} \in \mathsf{L_{HM_2}\ endorse\ Ra}(\mathsf{L_{HM_2}}, \mathsf{L_{HH}}, \mathsf{n})$$

This program consists of a sequence of three programs p_1, p_2 and p_3.

- *In p_1 we apply the rule Assign-Counter that binds the type of* pc *to be $\bot\ Cvar$.*

$$\frac{\Gamma \vdash \mathsf{pc} : \bot\ \mathsf{Cvar}}{\Gamma \vdash \mathsf{pc} := \mathsf{pc} + 1 : \bot\ \mathsf{cmd}}\ \text{(ASSIGN-COUNTER)}$$

- *In p_2, the SBroadcast rule checks (A) the type of the message to broadcast SU, (B) the type of the security level variable L_{HM_2}, (C) the type of the key variable K and finally (D) the type of* pc.

$$\frac{(A) \qquad (B) \qquad (C) \qquad (D)}{\Gamma \vdash \mathsf{sbroadcast}(\mathsf{L}_{\mathsf{HM}_2}, \mathsf{SU}_{\mathsf{M}_1\mathsf{M}_2} || \mathsf{pc}, \mathsf{K}) : \mathsf{HM}_2\ \mathsf{cmd}}\ \textit{(SBROADCAST)}$$

A:

$$\frac{\dfrac{\dfrac{\Gamma(SU) = M_1M_2\ var}{\Gamma \vdash SU : M_1M_2\ var}\ \textit{(VAR)}}{\Gamma \vdash SU : M_1M_2}\ \textit{(S-VAR)} \qquad \dfrac{M_1M_2 \leq HM_2}{\vdash M_1M_2 \subseteq HM_2}\ \textit{(BASE)}}{\Gamma \vdash SU : HM_2}\ \textit{(SUBTYPE)}$$

B:

$$\frac{\Gamma(L_{HM_2}) = L_{HM_2}\ Lvar}{\Gamma \vdash L_{HM_2} : HM_2\ Lvar}\ \textit{(LVAR)}$$

C:

$$\frac{\Gamma(K) = \top\ Kvar}{\Gamma \vdash K : \top\ Kvar}\ \textit{(KVAR)}$$

D:

$$\frac{\Gamma \vdash \mathsf{pc} : \bot\ \mathsf{Cvar}}{\Gamma \vdash \mathsf{pc} := \mathsf{pc} + 1 : \bot\ \mathsf{cmd}}\ \text{(ASSIGN-COUNTER)}$$

Finally, we apply the Subtype rule on p_2 to bind it to type $HH\ cmd$:

$$\dfrac{(SBROADCAST) \qquad \dfrac{\text{(CMD)}\quad \dfrac{HH \leq HM_2}{\vdash HH \subseteq HM_2}\ \textit{(BASE)}}{\vdash HM_2\ cmd \subseteq HH\ cmd}}{\Gamma \vdash \mathsf{sbroadcast}(\mathsf{L_{HM_2}}, \mathsf{SU_{M_1M_2}}||\mathsf{pc}, \mathsf{K}) : \mathsf{HH\ cmd}}\ \textit{(SUBTYPE)}$$

Therefore, p_2 is typable.

- *In order to type p_3, we apply the Remote Attestation rule to check the integrity of $SU_{M_1M_2}$ running on nodes that belongs to the security level L_{HM_2} and subsequently to ensure that nodes n can be moved to the security level L_{HH} if the verification succeeds. In addition to checking (A, B) the types of the security levels variables, this rule constrains a confidentiality and integrity order between the two security levels :$I(HH) \leq_I I(HM_2)$ and $C(HM_2) \leq_C C(HH)$.*

$$\dfrac{\text{(REMOTE ATTESTATION)} \qquad (A) \qquad (B) \qquad (C)}{\Gamma \vdash \mathsf{for\ n} \in \mathsf{L_{HM_2}}\ \mathsf{endorse\ Ra}(\mathsf{L_{HM_2}}, \mathsf{L_{HH}}, \mathsf{n})}$$

$$\text{A:}\quad \dfrac{\Gamma(L_{HM_2}) = L_{HM_2}\ Lvar}{\Gamma \vdash L_{HM_2} : HM_2\ Lvar}\ \textit{(LVAR)}$$

$$\text{B:}\quad \dfrac{\Gamma(L_{HH}) = L_{HH}\ Lvar}{\Gamma \vdash L_{HH} : HH\ Lvar}\ \textit{(LVAR)}$$

Finally, we apply the Subtype rule on p_3 to bind it to type $HH\ cmd$ and therefore show that the program p_3 is typable.

As a last step, we apply the Sequence rule on p_1, p_2 and p_3 to show that p is typable.

$$\frac{\Gamma \vdash p_1 : HH\ cmd \qquad \Gamma \vdash p_2 : HH\ cmd \qquad \Gamma \vdash p_3 : HH\ cmd}{\Gamma \vdash p_1; p_2; p_3 : HH\ cmd}\ \textit{(SEQUENCE)}$$

Example 22. *We consider the diamond lattice in Figure 5.4, in which $LH \leq HH$ where $L \leq_C H$ which means that L has less confidentiality than H. In this example, we consider a server willing to send a message x of type $LH\ var$ to a set of nodes in L_{LH}. We also consider a variable y of type $HH\ var$, a counter,* pc *of type $\bot\ Cvar$, and a key, K of type $\top\ Kvar$. We show that the following program p is not typable:*

$$\mathsf{pc} := \mathsf{pc} + 1;\ x_{LH} := y_{HH}; \mathsf{sbroadcast}(\mathsf{L_{LH}}, \mathsf{x_{LH}} || \mathsf{pc}, \mathsf{K})$$

The program p consists of a sequence of three programs p_1, p_2 and p_3.

- *In p_1 we apply the rule Assign-Counter that binds the type of* pc *to be $\bot\ Cvar$.*

$$\frac{\Gamma \vdash \mathsf{pc} : \bot\ \mathsf{Cvar}}{\Gamma \vdash \mathsf{pc} := \mathsf{pc} + 1 : \bot\ \mathsf{cmd}}\ \text{(ASSIGN-COUNTER)}$$

- *In p_2 the Assign rule constrains the types x_{LH} and y_{HH}.*

$$\frac{\Gamma \vdash x_{LH} : \tau\ var \qquad \Gamma \vdash y_{HH} : \tau}{\Gamma \vdash x_{LH} := y_{HH} : \tau}\ \text{(ASSIGN)}$$

  - *We apply the Var rule on x_{LH} that checks if $\Gamma(x_{LH}) = LH\ var$ which is the case.*

$$\frac{\Gamma(x_{LH}) = LH\ var}{\Gamma \vdash x_{LH} : LH\ var}\ \text{(VAR)}$$

– *To bind y_{HH} to type LH, we apply the Subtype rule. To apply correctly the rule, $HH \subseteq LH$ requires to be satisfied. Since $HH \not\subseteq LH$, the Subtype rule fails.*

Hence, p_2 is not typable.

- *To type p_3, we apply the SBroadcast rule and check that all the constraints are satisfied.*

$$\frac{(A) \qquad (B) \qquad (C) \qquad (D}{\Gamma \vdash \mathsf{sbroadcast}(\mathsf{L_{LH}}, \mathsf{x_{LH}}||\mathsf{pc}, \mathsf{K}) : \mathsf{LH\ cmd}}\ (\textsc{SBroadcast})$$

A:
$$\frac{\Gamma(x_{LH}) = LH\ var}{\Gamma \vdash x_{LH} : LH\ var}\ (\textsc{Var})$$

B:
$$\frac{\Gamma(L_{LH}) = LH\ Lvar}{\Gamma \vdash L_{LH} : LH\ Lvar}\ (\textsc{LVar})$$

C:
$$\frac{\Gamma(K) = \top\ Kvar}{\Gamma \vdash K : \top\ Kvar}\ (\textsc{KVar})$$

D:
$$\frac{\Gamma(\mathsf{pc}) = \bot\ \mathsf{Cvar}}{\Gamma \vdash \mathsf{pc} : \bot}\ (\textsc{S-CVar})$$

Therefore p_3 is well-typed. Since the second program p_2 is not typable, the Sequence rule is not applicable and therefore the entire program is not typable. Indeed, p is not

secure despite the fact that the broadcast seems to be secure. Actually, the server is broadcasting a message that contains high confidentiality information (H) to nodes with confidentiality clearance (L) since x_{LH} contains information from y_{HH}. This flow of information is not allowed by our architecture as it leaks confidential information to nodes placed in L_{LH}. This may compromise the confidentiality and security of a system broadcasting important messages. It would allow the server to transmit confidential information to nodes situated in lower security classes.

Example 23. *We consider the diamond lattice in Figure 5.4, in which $LH \leq HH$ and L has less confidentiality than H. We consider a program p: $L_{LH} := x_{HH}$ that is not typable but our security property does not detect it and considers the program as secure.*

We apply the Assign rule on p and we immediately observe that it is not typable because the Assign rule binds the first element of the assignment to be of type $LH\ var$.

$$\frac{\Gamma \vdash L_{LH} : \tau\ var \qquad \Gamma \vdash x_{HH} : \tau}{\Gamma \vdash L_{LH} := x_{HH} : \tau\ cmd}\ (\textsc{Assign})$$

Since L_{LH} is of type $LH\ Lvar$ and since there are no rules in our type system that allow a variable of type $\tau\ Lvar$ to be typed as $\tau\ var$, our type system catches this wrong flow of information and accordingly p is not typable.

5.2.5 Security Properties

We define confidentiality and integrity of secure information flow between nodes of different security levels as a new noninterference property [Sabelfeld and Myers, 2003]. We parametrize the definition by an attacker observation level, τ. For our security definition, it is useful to recall the definition of equivalence between memories from Definition 24 in Subsection 5.1.5.

We also define a filtering function in the purpose of projecting parts of the broadcasted messages sent to nodes mapped to security levels less or equal than the attacker observation level.

Definition 27 (Filtering)**.** *Let* $\mathsf{filter}_\tau(\mathsf{t}) = \mathsf{filter}'_\tau(\mathsf{t}, [\,])$ *and* $\mathsf{filter}'_\tau([\,], \mathsf{t}) = \mathsf{t}$.

$$\mathsf{filter}'_\tau(\mathsf{BEnc}(\mathsf{L}, \mathsf{vk}, \mathsf{v}'||\mathsf{v}) \cdot \mathsf{t}', \mathsf{t}) = \begin{cases} \mathsf{filter}'_\tau(\mathsf{t}', \mathsf{t} \cdot (\mathsf{L}, \mathsf{v}')) & \text{if } \Gamma(\mathsf{L}) \leq \tau \\ \mathsf{filter}'_\tau(\mathsf{t}', \mathsf{t}) & Otherwise \end{cases}$$

Our main result follows: a well-typed program is noninterferent at τ for Γ.

Theorem 3. *Let* Γ *be a typing environment and* $\tau \in \mathscr{L}$ *a security label. If* $\Gamma \vdash p$ *then* p *is* NI^{Γ}_{τ}.

We prove this theorem by induction on the height of the typing derivation tree of $\Gamma \vdash p$ in Appendix B.

Example 24. *Concerning the security property, we show that the program* p *of Example 21 is noninterferent at* HM_2*. Let two memories* μ_0 *and* μ_1*:*

μ_0	μ_1
$\mathsf{pc} := 1$	$\mathsf{pc} := 2$
$K := vk'_1$	$K := vk'_3$
$L_{HH} := \{"n_0"\}$	$L_{HH} := \{"n_0"\}$
$L_{HM_2} := \{"n_1", "n_2"\}$	$L_{HM_2} := \{"n_1", "n_2"\}$
$SU_{M_1M_2} := v_2$	$SU_{M_1M_2} := v_2$

Notice that $\mu_0 =^{\Gamma}_{HM_2} \mu_1$*: By Definition 24,* $\mu_0 =^{\Gamma}_{HM_2} \mu_1$ *since only variables of type* $\tau\ var$ *(in our example* $SU_{M_1M_2}$*) should be equal.*

The resulting two memories μ'_0 *and* μ'_1 *are:*

μ'_0	μ'_1
$\mathsf{pc} := 2$	$\mathsf{pc} := 3$
$K := vk'_1$	$K := vk'_3$
$L_{HH} := \{"n_0", "n_2"\}$	$L_{HH} := \{"n_0", "n_2"\}$
$L_{HM_2} := \{"n_1"\}$	$L_{HM_2} := \{"n_1"\}$
$SU_{M_1M_2} := v_2$	$SU_{M_1M_2} := v_2$

After executing p, the value of $SU_{M_1M_2}$ *does not change in the resulting memories. Since only variables of type* $\tau\ var$ *are considered, then* $\mu'_0 =^{\Gamma}_{HM_2} \mu'_1$. *In order to satisfy the full hypothesis in Definition 27, we need to prove that* $\mathsf{filter}_{\mathsf{HM}_2}(\mathsf{t}_0) = \mathsf{filter}_{\mathsf{HM}_2}(\mathsf{t}_1)$. *Actually,* t_0 *and* t_1 *are the traces of the messages broadcasted over the network through the Secure Broadcast rule in Figure 5.5:* $\mathsf{BEnc}(\mathsf{L},\mathsf{K},\mathsf{SU}||\mathsf{pc})$.

- *For an execution that starts with* μ_0, $t_0 = \mathsf{BEnc}(\mathsf{L}_{\mathsf{HM}_2}, \mathsf{v}'_1, \mathsf{v}_2||1)$, *thus* $\mathsf{filter}_{\mathsf{HM}_2}(\mathsf{t}_0) = [\mathsf{L}_{\mathsf{HM}_2}, \mathsf{v}_2]$.

$$\begin{aligned}\mathsf{filter}_{\mathsf{HM}_2}(\mathsf{t}_0) &= \mathsf{filter}'_{\mathsf{HM}_2}(\mathsf{t}_0, [\,]) \\ &= \mathsf{filter}'_{\mathsf{HM}_2}(\mathsf{BEnc}(\mathsf{L}_{\mathsf{HM}_2}, \mathsf{v}'_1, \mathsf{v}_2||1) \cdot [\,], [\,]) \\ &= \mathsf{filter}'_{\mathsf{HM}_2}([\,], [\,] \cdot (\mathsf{L}_{\mathsf{HM}_2}, \mathsf{v}_2)) \\ &= [L_{HM_2}, v_2]\end{aligned}$$

- *For an execution that starts with* μ_1, $t_1 = \mathsf{BEnc}(\mathsf{L}_{\mathsf{HM}_2}, \mathsf{v}'_3, \mathsf{v}_2||2)$, *thus* $\mathsf{filter}_{\mathsf{HM}_2}(\mathsf{t}_1) = [\mathsf{L}_{\mathsf{HM}_2}, \mathsf{v}_2]$.

$$\begin{aligned}\mathsf{filter}_{\mathsf{HM}_2}(\mathsf{t}_1) &= \mathsf{filter}'_{\mathsf{HM}_2}(\mathsf{t}_1, [\,]) \\ &= \mathsf{filter}'_{\mathsf{HM}_2}(\mathsf{BEnc}(\mathsf{L}_{\mathsf{HM}_2}, \mathsf{v}'_3, \mathsf{v}_2||2) \\ &= \mathsf{filter}'_{\mathsf{HM}_2}([\,], [\,] \cdot (\mathsf{L}_{\mathsf{HM}_2}, \mathsf{v}_2)) \\ &= [L_{HM_2}, v_2]\end{aligned}$$

Hence, $\mathsf{filter}_{\mathsf{HM}_2}(\mathsf{t}_0) = \mathsf{filter}_{\mathsf{HM}_2}(\mathsf{t}_1)$ *and p is* $NI^{\Gamma}_{HM_2}$.

Example 25. *We show that the program p of Example 22 does not comply with* NI^{Γ}_{LH} *according to Definition 26.*

Let memories μ_0 *and* μ_1 *be:*

μ_0	μ_1
$x_{LH} := 2$	$x_{LH} := 2$
$y_{HH} := 4$	$y_{HH} := 5$
$L_{LH} := \{"n_1","n_2"\}$	$L_{LH} := \{"n_1","n_2"\}$
$\mathsf{pc} := 1$	$\mathsf{pc} := 2$
$K := vk'$	$K := vk'_1$

The memories μ_0 and μ_1 are LH-Equal since the value of x_{LH} is the same in both memories. The two memoriesμ'_0 and μ'_1 are the resulting memories of the execution of p.

μ'_0	μ'_1
$x_{LH} := 4$	$x_{LH} := 5$
$y_{HH} := 4$	$y_{HH} := 5$
$L_{LH} := \{"n_1","n_2"\}$	$L_{LH} := \{"n_1","n_2"\}$
$\mathsf{pc} := 2$	$\mathsf{pc} := 3$
$K := vk'$	$K := vk'_1$

The two memories μ'_0 and μ'_1 are not LH-Equal since the value of x_{LH} is different and therefore the program p is not NI^{Γ}_{LH}.

Example 26. *We show that the program p of Example 23 , is indeed NI^{Γ}_{LH} because our security property is not strong enough as to capture information flows to level variables (in contrast to our type system that do capture them).*

Let memories μ_0 and μ_1 be:

μ_0	μ_1
$L_{LH} := \{"n_1"\}$	$L_{LH} := \{"n_1"\}$
$x_{HH} := 1$	$x_{HH} := 4$

The two memories μ_0 and μ_1 are LH-Equal as our security property considers only variables of type LH var.

The resulting memories μ'_0 and μ'_1 of the execution of p are:

μ_0'	μ_1'
$L_{LH} := 1$	$L_{LH} := 4$
$x_{HH} := 1$	$x_{HH} := 4$

The resulting memories are also LH-Equal. Since p is an assignment command, there are no messages broadcasted over the network, and therefore the traces are empty. Thus, $\mathsf{filter}(\mathsf{t}_0) = \mathsf{filter}(\mathsf{t}_1)$ *and p is* NI^{Γ}_{LH} *despite the fact that p is not typable.*

5.3 Related Work

Confidentiality vs. integrity. The fundamental purpose of controlling information flow is to provide confidentiality and integrity properties of code running on computers. Concerning confidentiality, sensitive or secret data should be prevented from flowing to public targets, and jointly, concerning integrity, untrusted data should be prevented from flowing or affecting trusted outputs [Sabelfeld and Myers, 2003, Li et al., 2003, Biba, 2014]. An information flow policy can be defined through a security lattice [Denning and Denning, 1977], that classifies and labels the data. A security lattice can represent confidentiality, integrity, and the combination of both. In a confidentiality lattice, data is labeled as *high* (for secret) and *low* (for public), where an attacker is assumed to observe the data labeled as *low* and information can only flow from *low* to *high*. In an integrity lattice, data is labeled as *trusted* and *untrusted*, where an attacker is assumed to control the data labeled as *untrusted* and information from *untrusted* data does not affect *trusted* data (information can flow from *trusted* to *untrusted*). In a confidentiality and integrity lattice, namely product lattice, data is labeled with a confidentiality and integrity levels. Information can only flow from *public* and *trusted* data to *secret* and *untrusted* data.

Secure information flow. Information flow policies [Denning and Denning, 1977, Sabelfeld and Myers, 2003, Banerjee and Naumann, 2002a] mainly focus on controlling the leak of information from secret data to public data through a program. Particularly, secure information flow can protect the confidentiality and integrity of data flows. Goguen et. al [Goguen and Meseguer, 1982] first introduced noninterference in the form of

simulation relations in the intent of formalizing information flow policies. By employing static enforcement through a type system, Volpano and Smith proved that the latter ensures noninterference [Volpano et al., 1996]. Afterwards, several researchers have proposed type systems for secure information flow [Smith, 2001, Fournet and Rezk, 2008, Fournet et al., 2009, C.Fournet et al., 2011, Magazinius et al., 2010, Bastys et al., 2018, Sabelfeld and Sands, 2000, Banerjee and Naumann, 2002b, Barthe et al., 2006]. In fact, to prevent insecure flows, one may use static information flow enforcement [Volpano et al., 1996] or dynamic enforcement [Askarov and Sabelfeld, 2009, Bielova and Rezk, 2016]. In our work, we choose to apply an static mechanism to enforce secure information flow through a type system. In the literature of secure information flow, several works consider combining cryptography to information flow [Laud, 2001, Askarov et al., 2015, Gregersen et al., 2019, Fournet and Rezk, 2008, Fournet et al., 2009, C.Fournet et al., 2011] including our work. None of these works consider the association of security classes to subgroups of nodes in distributed systems and combine them with BES. Moreover, none of these works except [Fournet and Rezk, 2008] consider endorsement of integrity: while they obtain endorsement via cryptographic signatures, we obtain the endorsement through remote attestation. In our configuration, we enforce the integrity of communications by employing remote attestation protocols. The goal is to verify that nodes has successfully installed software updates, and therefore are moved to higher integrity security levels. By doing that, the server can identify which nodes necessitate software updates and the system can avoid leaking information to nodes placed in lower security levels.

5.4 Conclusion

We propose two architectures to securely deliver messages (and software updates) in hierarchical distributed systems relying on broadcast encryption schemes (and remote attestation). By associating broadcast encryption keys to security classes, we pave the way to enforcement of secure information flow between server and nodes. We demonstrate

this by building two type systems for server code. In the first type system, nodes are associated to static security levels and in the second type system, nodes are associated to dynamic security levels, which is an appropriate scenario for delivering software updates. We also prove the soundness of the two type systems with respect to a new secure information flow property. While the first property preserves only confidentiality, the second property preserves both confidentiality and integrity of secure information flow. To the best of our knowledge, we are the first to propose the association of security classes to cryptographic keys in BES and the employment of remote attestation in order to verify secure information flow. Our current results are mainly theoretical, but being based on known bricks such as broadcast encryption schemes and remote attestation an implementation is feasible. We leave the experimental evaluation of our results as future work.

CHAPTER 6

CONCLUSION

In this thesis, we investigate the security of encryption schemes. First, we focus our study on the security properties that the ElGamal encryption scheme relies on. ElGamal encryption scheme is semantically secure on top of groups that comply with the Decisional Diffie-Hellman (*DDH*) assumption. Specifically, it is *IND-CPA* over safe prime order groups (in a safe prime order group, the *DDH* assumption is respected). We inspect ElGamal encryption scheme libraries in order to identify which implementations respect the *DDH* assumption. From the analysis of the implementations, we identify and compare four message encoding and decoding techniques that satisfy the aforementioned assumption.

Since the *DDH* assumption applies also to other cryptographic constructions, we investigate broadcast encryption schemes and discuss their security aspect. We propose a new broadcast encryption scheme based on ElGamal that enjoys constant size ciphertext and key storage. We implement three different broadcast encryption schemes in Ocaml, and compare them in means of execution time and ciphertext and key space. It turns out that the our scheme is faster than the other compared schemes offering constant time encryption and decryption even for a large set of nodes. Comparing to Boneh-Franklin scheme and ElGamal Baseline, the new scheme shows significant advantage in terms of

execution time, ciphertext space and key storage.

Because broadcast encryption schemes involve communication with a large set of nodes, we consider scenarios that employ broadcast encryption schemes to securely communicate with nodes. We propose an architecture to deliver messages to nodes with static security levels, and we extend it to a more complex architecture to deliver software updates to nodes with dynamic security levels. We associate broadcast encryption keys to security classes to enforce secure information flow between server and nodes, in particular, to preserve the confidentiality and integrity of communication. To demonstrate our idea, we build two type systems (one for each architecture) and prove their soundness with respect to a new secure information flow property that preserves confidentiality and integrity of information.

In what follows, we detail some perspective for future work, building on the work presented so far.

Security of Implementations and elliptic curves. An interesting study related to our work is to analyze and study implementations of ElGamal over elliptic curves. Since we focus our analysis on message encoding over cyclic subgroups of $\mathbb{Z}_p^*$, it is possible to extend it and examine in detail how encodings are implemented using the elliptic curves cryptography [Koblitz et al., 2000]. Another extension regarding this topic is to black-box test applications that employ ElGamal encryption scheme through Hardware Security Modules [Volkamer, 2009].

Security of broadcast encryption schemes. In this thesis, we propose a broadcast encryption scheme based on ElGamal and implement it over prime order groups. In the short term, we plan to prove that this scheme is *IND-CPA* using the Easycrypt tool [Barthe et al., 2013]. We conjecture that this proof is closely related to the proof of ElGamal [ElGamal, 1985] combined with the higher order residuosity hypothesis used in the proof of the Paillier Encryption scheme [Paillier, 1999]. As a second line of future work, we would like to adapt the scheme to elliptic curve cryptography to seek optimization in terms of speed, memory space, and small key sizes. Since elliptic

curve cryptography offers the same security level while adopting smaller keys, it is interesting to implement the schemes evaluated in Chapter 4 to compare results and draw conclusions on the benefit of employing elliptic curves cryptography for secure broadcasting. Considering the comparison done with the Boneh-Franklin scheme, we plan to further investigate traitor-tracing schemes as they offer an efficient and secure way to detect malicious nodes in the system and find an optimized way (in term of speed and memory) to integrate traitor-tracing approaches to our scheme.

Real life applications of the proposed architecture and type systems. Considering that the proposed architectures are based on well-known notions as broadcast encryption schemes and remote attestation, and since they are supported by to type systems for enforcing secure information flows, our aim is to provide an implementation of our type systems (i.e. in Hop.js [Serrano and Prunet, 2016]) for real life applications.

References

Adida, B. (2008). Helios: Web-based open-audit voting. In *USENIX '08*, pages 335–348.

Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Béguelin, S. Z., and Zimmermann, P. (2015). Imperfect forward secrecy: How diffie-hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 5–17. ACM.

Agrawal, S. and Yamada, S. (2020). Optimal broadcast encryption from pairings and LWE. In *Advances in Cryptology - EUROCRYPT '20*, volume 12105 of *Lecture Notes in Computer Science*, pages 13–43. Springer.

Alves, P. (2015). Public source code of library n.5. https://github.com/pdroalves.

Askarov, A., Moore, S., Dimoulas, C., and Chong, S. (2015). Cryptographic enforcement of language-based information erasure. In *IEEE 28th Computer Security Foundations Symposium, CSF '15*, pages 334–348. IEEE Computer Society.

Askarov, A. and Sabelfeld, A. (2009). Tight enforcement of information-release policies for dynamic languages. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF '09*, pages 43–59. IEEE Computer Society.

Babenko, L., Pisarev, I., and Makarevich, O. B. (2017). A model of a secure electronic voting system based on blind intermediaries using russian cryptographic algorithms. In *SIN '17*, pages 45–50.

Banerjee, A. and Naumann, D. A. (2002a). Secure information flow and pointer con-

finement in a java-like language. In *15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada*, page 253. IEEE Computer Society.

Banerjee, A. and Naumann, D. A. (2002b). Secure information flow and pointer confinement in a java-like language. In *15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada*, page 253. IEEE Computer Society.

Banks, A. S., Kisiel, M., and Korsholm, P. (2021). Remote attestation: A literature review. *CoRR*, abs/2105.02466.

Barthe, G., Dupressoir, F., Grégoire, B., Kunz, C., Schmidt, B., and Strub, P. (2013). Easycrypt: A tutorial. In *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer.

Barthe, G., Grégoire, B., and Béguelin, S. Z. (2009). Formal certification of code-based cryptographic proofs. In *POPL '09*, pages 90–101.

Barthe, G., Grégoire, B., Lakhnech, Y., and Béguelin, S. Z. (2011a). Beyond provable security verifiable IND-CCA security of OAEP. In *Topics in Cryptology - CT-RSA '11 - The Cryptographers' Track at the RSA Conference*, volume 6558 of *Lecture Notes in Computer Science*, pages 180–196. Springer.

Barthe, G., Grégoire, B., Lakhnech, Y., and Béguelin, S. Z. (2011b). Beyond provable security verifiable IND-CCA security of OAEP. In *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 180–196. Springer.

Barthe, G., Rezk, T., and Naumann, D. A. (2006). Deriving an information flow checker and certifying compiler for java. In *2006 IEEE Symposium on Security and Privacy (S&P 2006), 21-24 May 2006, Berkeley, California, USA*, pages 230–242. IEEE Computer Society.

Bastys, I., Piessens, F., and Sabelfeld, A. (2018). Prudent design principles for information

flow control. In *Proceedings of the 13th Workshop on Programming Languages and Analysis for Security, PLAS@CCS '18,*, pages 17–23. ACM.

Belenios (2016). Public source code of library n.1. https://github.com/glondu/belenios.

Bellare, M. and Rogaway, P. (1994). Optimal asymmetric encryption. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer.

Bernstein, D. J., Hamburg, M., Krasnova, A., and Lange, T. (2013). Elligator: elliptic-curve points indistinguishable from uniform random strings. In *ACM SIGSAC '13*, pages 967–980.

Biba, K. J. (2014). Integrity considerations for secure computer systems. In *Technical Report, ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA, April 1977*.

Bielova, N. and Rezk, T. (2016). A taxonomy of information flow monitors. In *Principles of Security and Trust - 5th International Conference, POST*, volume 9635 of *Lecture Notes in Computer Science*, pages 46–67. Springer.

Boneh, D. (1998). The decision diffie-hellman problem. pages 48–63.

Boneh, D. and Franklin, M. K. (1999). An efficient public key traitor tracing scheme. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 338–353. Springer.

Boneh, D., Gentry, C., and Waters, B. (2005). Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Advances in Cryptology - CRYPTO '05*, Lecture Notes in Computer Science, pages 258–275. Springer.

Boneh, D., Joux, A., and Nguyen, P. Q. (2000). Why textbook elgamal and RSA encryption are insecure. In *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 30–43. Springer.

Boneh, D., Waters, B., and Zhandry, M. (2014). Low overhead broadcast encryption from multilinear maps. In *Advances in Cryptology - CRYPTO '14*, volume 8616 of *Lecture Notes in Computer Science*, pages 206–223. Springer.

Botan (2018). Public source code of library n.2. https://github.com/randombit/botan.

Canetti, R., Garay, J. A., Itkis, G., Micciancio, D., Naor, M., and Pinkas, B. (1999). Multicast security: A taxonomy and some efficient constructions. In *Proceedings IEEE INFOCOM '99*, pages 708–716. IEEE Computer Society.

C.Fournet, Planul, J., and Rezk, T. (2011). Information-flow types for homomorphic encryptions. In *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS '11*, pages 351–360. ACM.

Chen, L., Li, J., and Zhang, Y. (2020). Adaptively secure efficient broadcast encryption with constant-size secret key and ciphertext. *Soft Comput.*, 24(6):4589–4606.

Chevallier-Mames, B., Paillier, P., and Pointcheval, D. (2006). Encoding-free elgamal encryption without random oracles. In *PKC '06*, pages 91–104.

Chhatrapati, A., Hohenberger, S., Trombo, J., and Vusirikala, S. (2021). A performance evaluation of pairing-based broadcast encryption systems. *IACR Cryptol. ePrint Arch.*, page 1526.

Conti, M., Dragoni, N., and Lesyk, V. (2016). A survey of man in the middle attacks. *IEEE Commun. Surv. Tutorials*, 18(3):2027–2051.

Cortier, V., Fuchsbauer, G., and Galindo, D. (2015). Beleniosrf: A strongly receipt-free electronic voting scheme. *IACR*, 2015.

Cortier, V., Galindo, D., Küsters, R., Müller, J., and Truderung, T. (2016). Sok: Verifiability notions for e-voting protocols. In *IEEE, SP '16*, pages 779–798.

Cramer, R., Gennaro, R., and Schoenmakers, B. (1997). A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT '97*, pages 103–118.

Cramer, R. and Shoup, V. (1998). A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO '98*, pages 13–25.

Delerablée, C. and andD. Pointcheval, P. P. (2007). Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. In *Pairing-*

Based Cryptography - Pairing '07, volume 4575 of *Lecture Notes in Computer Science*, pages 39–59. Springer.

Denning, D. E. (1976). A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243.

Denning, D. E. and Denning, P. J. (1977). Certification of programs for secure information flow. *Commun. ACM*, 20(7):504–513.

Diaz, A. (2017). Public source code of library n.4. https://github.com/vrnvu/elgamal-ap.

Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Trans. Inf. Theory '76*, pages 644–654.

El Laz, M., Grégoire, B., and Rezk, T. (2020). Security analysis of elgamal implementations. In *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications, ICETE 2020 - Volume 2: SECRYPT, Lieusaint, Paris, France, July 8-10, 2020*, pages 310–321. ScitePress.

ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory '85*, pages 469–472.

Elgamir (2016). Public source code of library n.14. https://github.com/d5c5ceb0/elgamir.

Estonia (2017). Public source code of library n.15. https://github.com/vvk-ehk/ivxv.

Fadavi, M., Farashahi, R. R., and Sabbaghian, S. (2018). Injective encodings to binary ordinary elliptic curves. In *SAC '18*, pages 434–449.

Farashahi, R. R. (2014). Hashing into hessian curves. *IJACT '14*, pages 139–147.

Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R. S., and Wood, C. A. (2019). Hashing to Elliptic Curves. https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-05. Technical report.

Fiat, A. and Naor, M. (1993). Broadcast encryption. In *Advances in Cryptology - CRYPTO '93*, volume 773, pages 480–491.

Fournet, C., Guernic, G. L., and Rezk, T. (2009). A security-preserving compiler for distributed programs: from information-flow policies to cryptographic mechanisms. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS '09,*, pages 432–441. ACM.

Fournet, C. and Rezk, T. (2008). Cryptographically sound implementations for typed information-flow security. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '08*, pages 323–335. ACM.

Francillon, A. (2009). *Attacking and Protecting Constrained Embedded Systems from Control Flow Attacks*. PhD thesis, Institut National Polytechnique de Grenoble - INPG.

Fujisaki, E., Okamoto, T., Pointcheval, D., and Stern, J. (2004). RSA-OAEP is secure under the RSA assumption. *J. Cryptol.*, 17(2):81–104.

Gaudry, P. (2019). Breaking the encryption scheme of the moscow internet voting system. *CoRR '19*.

Gentry, C. and Silverberg, A. (2002). Hierarchical id-based cryptography. In *Advances in Cryptology - ASIACRYPT '02*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566. Springer.

Goguen, J. A. and Meseguer, J. (1982). Security policies and security models. In *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, pages 11–20. IEEE Computer Society.

Goldwasser, S. and Micali, S. (1982). Probabilistic encryption and how to play mental poker keeping secret all partial information. In *ACM '82*, pages 365–377.

Gregersen, S. O., Thomsen, S. E., and Askarov, A. (2019). A dependently typed library for static information-flow control in idris. *CoRR*, abs/1902.06590.

Gura, N., Patel, A., Wander, A., Eberle, H., and Shantz, S. C. (2004). Comparing elliptic curve cryptography and RSA on 8-bit cpus. In *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 119–132. Springer.

Haines, T., Goré, R., and Tiwari, M. (2019). Verified verifiers for verifying elections. In *ACM SIGSAC*, CCS '19, pages 685–702.

Helios (2008). Public source code of library n.16. https://github.com/benadida/helios-

server.

Horwitz, J. and Lynn, B. (2002). Toward hierarchical identity-based encryption. In *Advances in Cryptology - EUROCRYPT '02*, volume 2332 of *Lecture Notes in Computer Science*, pages 466–481. Springer.

IETF (October 30, 2017). A firmware update architecture for internet of things devices.

Ioannou, O. (2014). Public source code of library n.24. https://github.com/oorestisime.

Joye, M. and Libert, B. (2017). Encoding-free elgamal-type encryption schemes on elliptic curves. In *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 19–35. Springer.

Koblitz, N., Menezes, A., and Vanstone, S. A. (2000). The state of elliptic curve cryptography. *Des. Codes Cryptogr. '00*, pages 173–193.

Kubjas, I., Pikma, T., and Willemson, J. (2017). Estonian voting verification mechanism revisited again. *IACR*, 2017.

Laud, P. (2001). Semantics and program analysis of computationally secure information flow. In *Programming Languages and Systems, 10th European Symposium on Programming, ESOP*, volume 2028 of *Lecture Notes in Computer Science*.

Lee, J., Kim, J., and Oh, H. (2019). BESTIE: broadcast encryption scheme for tiny iot equipments. *IACR Cryptol. ePrint Arch.*

Lee, R. (2017). Public source code of library n.7. https://github.com/rayli-bot/modulus-calculation.

Li, P., Mao, Y., and Zdancewic, S. (2003). Information integrity policies. In *In Proceedings of the Workshop on Formal Aspects in Security and Trust, Sept. 2003*.

Libgcrypt (2013). Public source code of library n.17. https://github.com/gpg/libgcrypt.

Lipton, R. J. (1981). How to Cheat at Mental Poker. In *Proceeding of AMS short course on Cryptology '81*.

Magazinius, J., Askarov, A., and Sabelfeld, A. (2010). A lattice-based approach to mashup security. In *Proceedings of the 5th ACM Symposium on Information, Computer anD Communications Security, ASIACCS*, pages 15–23. ACM.

Martin, B. (2004). *Codage, Cryptologie et Applications*. Collection technique et scientifique des télécommunications. resses Polytechniques et Universitaires Romande.

Microsoft (2019). Public source code of library n.18. https://github.com/microsoft/electionguard-verifier.

Miller, V. S. (1985). Use of elliptic curves in cryptography. In *CRYPTO '85*, pages 417–426.

Milne, J. S. (2011). Fields and galois theory (v4.22).

Moscow (July, 2019a). Public source code of library n.19. https://github.com/moscow-technologies.

Moscow (September, 2019b). Public source code of library n.20. https://github.com/moscow-technologies.

Musat, A. (2017). Public source code of library n.12. https://github.com/andreeamusat/.

Naor, D., Naor, M., and Lotspiech, J. B. (2001). Revocation and tracing schemes for stateless receivers. In *Advances in Cryptology - CRYPTO '01*, Lecture Notes in Computer Science, page 41–62. Springer.

Nasr, H. (2015). Public source code of library n.3. https://github.com/ananasr/cryptology.

Ne Oo, H. and Aung, A. (2014). A survey of different electronic voting systems. *IJSER '14*.

Nishihara, N., Harasawa, R., Sueyoshi, Y., and Kudo, A. (2009). A remark on the computation of cube roots in finite fields. *IACR '09*, page 457.

Norvegia (2017). Public source code of library n.21. https://www.regjeringen.no/en/topics/elections-and-democracy/den-norske-valgordningen/the-norwegian-electoral-system/id456636.

Orr, D. and Liam, K. (2016). SAC '15. Lecture Notes in CS '16.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99*, pages 223–238.

Pankratiew, A. (2018). Public source code of library n.10. https://github.com/n1ghtf1re/public-key-ciphers.

Pellegrini, J. (2017). Public source code of library n.11.

http://aleph0.info/jp/software/elgamal.

Pohlig, S. C. and Hellman, M. E. (1978). An improved algorithm for computing logarithms over gf(p) and its cryptographic significance (corresp.). *IEEE Trans. Inf. Theory*, 24:106–110.

Pollard, J. M. (1978). Monte carlo methods for index computation $({mod} p)$.

Puigalli, J. and Guasch, S. (2012). Cast-as-intended verification in norway. In *EVOTE '12*, pages 49–63.

Pycrypto (2012). Public source code of library n.22. https://github.com/dlitz/pycrypto.

Pycryptodome (2018). Public source code of library n.23. https://github.com/legrandin/pycryptodome.

Rackoff, C. and Simon, D. R. (1991). Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology - CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer.

Rémy, D. (2000). Using, understanding, and unraveling the ocaml language. from practice to theory and vice versa. In *APPSEM '00*, pages 413–536.

Rezk, T. (April, 2018). *Secure Programming, HdR Thesis*. PhD thesis, University of Nice-Sophia Antipolis, France.

Riddle, R. (2014). Public source code of library n.13. https://github.com/ryanriddle/elgamal.

Ridhuan, I. (2016). Public source code of library n.8. https://github.com/ilyasridhuan/elgamal.

Rivest, R. L., Shamir, A., and Adleman, L. M. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126.

Rivest, R. L., Shamir, A., and Adleman, L. M. (1979). Mental poker. *The Mathematical Gardner '79*, pages 120–126.

Rotman, J. (1999). *An Introduction to the Theory of Groups*. Graduate Texts in Mathematics '99.

Sabelfeld, A. and Myers, A. C. (2003). Language-based information-flow security. *IEEE J. Sel. Areas Commun.*, 21(1).

Sabelfeld, A. and Sands, D. (2000). Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW '00, Cambridge, England, UK, July 3-5, 2000*, pages 200–214. IEEE Computer Society.

Sailer, R., Jaeger, T., Zhang, X., and Doorn, L. V. (2004). Attestation-based policy enforcement for remote access. In *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS '04*, pages 308–317. ACM.

Serrano, M. and Prunet, V. (2016). A glimpse of hopjs. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, pages 180–192. ACM.

Shaneck, M., Mahadevan, K., Kher, V., and Kim, Y. (2005). Remote software-based attestation for wireless sensors. In *Security and Privacy in Ad-hoc and Sensor Networks, Second European Workshop, ESAS '05*, volume 3813 of *Lecture Notes in Computer Science*, pages 27–41. Springer.

Shoup, V. (2000). OAEP reconsidered. *IACR Cryptol. ePrint Arch.*, page 60.

Sidorov, V. (2016). Public source code of library n.6. https://github.com/bazzilic/elgamalext.

Smith, G. (2001). A new type system for secure information flow. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 115–125. IEEE Computer Society.

Springall, D., Finkenauer, T., Durumeric, Z., Kitcat, J., Hursti, H., MacAlpine, M., and Halderman, J. A. (2014). Security analysis of the estonian internet voting system. In *ACM SIGSAC '13*, pages 703–715.

Swisspost (2018). Public source code of library n.25. https://gitlab.com/swisspost/evoting-solution.

Syverson, P. F. (1994). A taxonomy of replay attacks. In *Seventh IEEE Computer Security Foundations Workshop - CSFW '94*, pages 187–191. IEEE Computer Society.

Verificatum (2017). Public source code of library n.26. https://github.com/verificatum.

Volkamer, M. (2009). *Evaluation of Electronic Voting*. Lecture Notes in Business Inf.

Proc. '09.

Volpano, D. M., Irvine, C. E., and Smith, G. (1996). A sound type system for secure flow analysis. *J. Comput. Secur.*, 4(2/3):167–188.

Wang, K. (2017). Public source code of library n.9. https://github.com/wangkepfe/cryptography.

Yang, Y. (2014). Broadcast encryption based non-interactive key distribution in manets. *J. Comput. Syst. Sci.*, 80(3):533–545.

Zandberg, K., Schleiser, K., Padilla, F. A., Tschofenig, H., and Baccelli, E. (2019). Secure firmware updates for constrained iot devices using open standards: A reality check. *IEEE Access*, pages 71907–71920.

Zhang, L., Hu, Y., and Wu, Q. (2012). Adaptively secure identity-based broadcast encryption with constant size private keys and ciphertexts from the subgroups. *Math. Comput. Model.*, 55(1-2):12–18.

APPENDIX A

PROOF OF THEOREM 2 OF SECTION 5.1

To prove Theorem 2, we need to introduce two Lemmas:

Lemma 1. *(LowExpression)* $\forall \tau, \tau', \mu_0, \mu_1$, *if* $\mu_0 =_\tau^\Gamma \mu_1$ *and* $\Gamma \vdash e : \tau'$ *and* $\tau' \leq \tau$ *and* $\mu_0 \vdash e \Rightarrow v_0$ *and* $\mu_1 \vdash e \Rightarrow v_1$, *then* $v_0 = v_1$.

Lemma 2. *(HighCommand)* $\forall \tau, \tau', \mu, \mu'$, *t, if* $\Gamma \vdash c : \tau'$ *cmd and* $\tau' \not\leq \tau$ *and* $\mu \vdash c \Rightarrow^t \mu'$, *then* $\mu =_\tau^\Gamma \mu'$ *and* $\mathsf{filter}_\tau(\mathrm{t}) = [\,]$.

A.1 Proof of Lemma 1

Proof. We prove Lemma1 by structural induction on e with $P_{Struct}(e) = n$. According to our type system, specifically the subtyping rule *S-Var*, only variables of type $\tau\ var$ can be typed as τ. That is the reason we omit the cases of *NVar* and *MKVar*.

It follows that:

- $P_{Struct}(v) = 1$.
- $P_{Struct}(x) = 1$.
- $P_{Struct}(e \circ e') = P_{Struct}(e) + P_{Struct}(e')$.

From the hypothesis of Lemma 1, we have that:

(H_1) $\mu_0 =_\tau^\Gamma \mu_1$.

(H_2) $\Gamma \vdash e : \tau'$.

(H_3) $\tau' \leq \tau$.

(H_4) $\mu_0 \vdash e \Rightarrow v_0$.

(H_5) $\mu_1 \vdash e \Rightarrow v_1$.

And we need to prove that:

(G_1) $v_0 = v_1$.

Case 1 (Base Case)**.**

Subcase 1.1 (Lit: v)**.** *By (H_2) and the typing rule Lit, we have that:*

(H_6) $\Gamma \vdash v : \tau'$.

By (H_4) and the semantics rule Base, we have that:

(H_7) $\mu_0 \vdash v \Rightarrow v$.

By (H_5) and the semantics rule Base, we have that:

(H_8) $\mu_1 \vdash v \Rightarrow v$.

From (H_7) and (H_8), (G_1) is trivially true.

Subcase 1.2 (Var: x)**.** *By (H_2) and the subtyping rule S-Var (which is the only rule that allows variables of type τ' var to be typed as variables of type τ'), we have that:*

(H_6) $\Gamma \vdash x : \tau'\ var$.

By (H_6) and the typing rule Var, we have that:

(H_7) $\Gamma(x) = \tau'\ var$.

By (H_4) and the semantics rule Var, we have that:

(H_8) $\mu_0(x) = v_0$.

By (H_5) and the semantics rule Var, we have that:

(H_9) $\mu_1(x) = v_1$.

By Definition 24 and (H_1) of Lemma 1, we have that:

$$\forall y, \mu_0(y) = \mu_1(y),\ if \Gamma(y) = \tau'\ var \wedge \tau' \leq \tau$$

From this, and by (H_6) and (H_3), we have that :

(H_{10}) $\mu_0(x) = \mu_1(x)$.

By (H_{10}), (H_8) and (H_9), we conclude that $v_0 = v_1$, which is (G_1).

Case 2 (Op: $e' \circ e''$)**.** ***Inductive hypothesis for*** e

For $P_{Struct}(e) = n$, we have that:

(IH_1) $\mu_0 =_\tau^\Gamma \mu_1$.

(IH_2) $\Gamma \vdash e : \tau'$.

(IH_3) $\tau' \leq \tau$.

(IH_4) $\mu_0 \vdash e \Rightarrow v_0$.

(IH_5) $\mu_1 \vdash e \Rightarrow v_1$.

Then we conclude that:

(IH_6) $v_0 = v_1$.

Inductive case: height $= P_{Struct}(e) = n + 1$.

Op: $e' \circ e''$**.** *For the typing rule Op, we have that:*

(H_1*)* $\Gamma \vdash e' : \tau'$.

(H_2*)* $\Gamma \vdash e'' : \tau'$.

By (IH_4*) and the semantics rule Op, we have that:*

(H_3*)* $\mu_0 \vdash e' \Rightarrow v_0$.

(H_4*)* $\mu_0 \vdash e'' \Rightarrow v'_0$.

(H_5*)* $\mu_0 \vdash e' \circ e'' \Rightarrow v_0 \circ v'_0$.

By (IH_5*) and the semantics rule Op, we have that:*

(H_6*)* $\mu_1 \vdash e' \Rightarrow v_1$.

(H_7*)* $\mu_1 \vdash e'' \Rightarrow v'_1$.

(H_8*)* $\mu_1 \vdash e' \circ e'' \Rightarrow v_1 \circ v'_1$.

Concerning e', from the inductive hypothesis on e, it follows that:

(H_6*)* $v_0 = v_1$.

Concerning e", from the inductive hypothesis on e, it follows that:

(H_7*)* $v'_0 = v'_1$.

By the transitive property of $=$ *and since* $v_0 = v_1$ *and* $v'_0 = v'_1$*, we conclude that* $v_0 \cdot v'_0 = v_1 \cdot v'_1$*, which is* (G_1).

■

A.2 Proof of Lemma 2

Proof. We prove lemma2 by structural induction on c with $P_{Struct}(c) = n$. It follows that:

- $P_{Struct}(x := e') = 1$.
- $P_{Struct}(k := KG(\{n_1, ..., n_i\}, k')) = 1$.
- $P_{Struct}(\mathsf{sbroadcast}(\{\mathsf{n_1}, ..., \mathsf{n_j}\}, \mathsf{k}, \mathsf{e})) = 1$.
- $P_{Struct}(c'; c'') = P_{Struct}(c') + P_{Struct}(c'')$.
- $P_{Struct}(\mathsf{while\ e\ do\ c}) = \mathsf{P_{Struct}(c)} + 1$.
- $P_{Struct}(\mathsf{if\ e\ then\ c'\ else\ c''}) = \mathsf{MAX(P_{Struct}(c'), P_{Struct}(c''))} + 1$.

From the hypothesis of Lemma 2, we have that:

(H_1) $\Gamma \vdash c : \tau'\ cmd$.

(H_2) $\tau' \not\leq \tau$.

(H_3) $\mu \vdash c \Rightarrow^t \mu'$.

And we need to prove that:

(G_1) $\mu =_\tau^\Gamma \mu'$.

(G_2) $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

Case 1 (Base case)**.**

Subcase 1.1 (Assign: $x := e'$.)**.** *By the hypothesis of the lemma, we have that:*

(H_4) $\Gamma \vdash x := e' : \tau'\ cmd$.

By (H_4) and the typing rule Assign, we have that:

(H_5) $\Gamma \vdash x : \tau'\ var$.

(H_6) $\Gamma \vdash e : \tau'$.

By (H_5) and the typing rule Var, we have that:

(H_7) $\Gamma(x) = \tau'\ var$.

By (H_3) and the semantics rule Update, we have that:

(H_8) $\mu \vdash e \Rightarrow v$.

(H_9) $\mu \vdash x := e \Rightarrow^t \mu[x := v]$, *then* $\mu' = \mu[\mu[x := v]$.

By the semantics rule Update, we have that:

(H_{10}) $t = [\,]$.

For (H_2), we have that $\tau' \not\leq \tau$. *Since the only variable in which* μ *and* μ' *are different is x by (H_9) and the security level of the variable x is* $\tau' var$, *we can conclude that* $\mu =_\tau^\Gamma \mu'$ *which is (G_1).*

To prove (G_2), by (H_{10}), $t = [\,]$. *Therefore* $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

Subcase 1.2 (KVar-Assign: $k := KG(\{n_1,...,n_i\},k')$)**.** *By the hypothesis of Lemma 2 , we have that:*

(H_4) $\Gamma \vdash k := KG(\{n_1,...,n_i\},k') : \tau'\ cmd$.

By (H_4) and the typing rule KVar-Assign, we have that:

(H_5) $\Gamma(k) = \tau\ Kvar(n_i,...,n_i)$.

(H_6) $\Gamma \vdash n_j : \tau'\ Nvar(n_j)$.

(H_7) $\Gamma \vdash k' : \top\ MKvar$.

By (H_3) and the semantics rule Update, we have that:

(H_8) $\mu \vdash KG(\{n_1,...,n_i\},v) \Rightarrow vk$.

(H9) $\mu \vdash k' \Rightarrow v$.

(H10) $\mu(n_j) = v_j$.

(H11) $\mu \vdash k := KG(\{v_1, ..., v_i\}, k') \Rightarrow^t \mu[k := vk]$, *then* $\mu' = \mu[k := vk]$.

By the semantics rule Update, we have that:

(H12) $t = [\,]$.

For (H2), we have that $\tau' \not\leq \tau$. *Since the only variable in which* μ *and* μ' *are different is k by (H11) and the type of k is* $\tau' Kvar(n_1..., n_i)$ *by (H5), we can conclude that* $\mu =_\tau^\Gamma \mu'$ *which is (G1).*

To prove (G2), by (H12), $t = [\,]$. *Therefore* $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$, *which is (G2)*

Subcase 1.3 (SBroadcast: $\mathsf{sbroadcast}(\{\mathsf{n}_1, ..., \mathsf{n}_j\}, \mathsf{k}, \mathsf{e})$)**.** *By the hypothesis of the lemma, we have that:*

(H4) $\Gamma \vdash \mathsf{sbroadcast}(\{\mathsf{n}_1, ..., \mathsf{n}_j\}, \mathsf{k}, \mathsf{e}) : \tau'$ cmd.

By (H4) and the typing rule SBroadcast, we have that:

(H5) $\Gamma \vdash e : \tau'$.

(H6) $\Gamma \vdash n_i : \tau\, Nvar(n_i)$.

(H7) $\Gamma \vdash k : \tau\, Kvar\ (n_1 \ldots n_j)$.

By (H3) and the semantics rule Secure Broadcast, we have that:

(H8) $\mu \vdash \mathsf{sbroadcast}(\{\mathsf{n}_1, ..., \mathsf{n}_j\}, \mathsf{k}, \mathsf{e}) \Rightarrow^{\mathsf{t}} \mu$.

By the semantics rule Secure Broadcast, we have that:

(H9) $t = \mathsf{BEnc}(\{\mathsf{n}_1, ..., \mathsf{n}_k\}, \mathsf{vk}, \mathsf{v}')$.

For (H_2), we have that $\tau' \not\leq \tau$. Since the initial memory μ and the final memory μ are equal by (H_8), we conclude that $\mu =_\tau^\Gamma \mu'$ which is (G_1).

To prove (G_2), by (H_9), $t = \mathsf{BEnc}(\{\mathsf{n}_1,...,\mathsf{n}_k\},\mathsf{vk},\mathsf{v}')$. By Definition 25, and by (H_2) and (H_6), $\Gamma(n_1) \sqcap \Gamma(n_2) ... \sqcap \Gamma(n_k) \not\leq \tau$, therefore $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$; which proves ($G_2$)

Inductive hypothesis for c *For $P_{Struct}(c) = n$, we have that:*

(IH_1) $\Gamma \vdash c : \tau'\ cmd.$

(IH_2) $\mu \vdash c \Rightarrow^t \mu'.$

(IH_3) $\tau' \not\leq \tau.$

Then we conclude that:

(IH_4) $\mu =_\tau^\Gamma \mu'.$

(IH_5) $\mathsf{filter}_\tau(\mathsf{t}) = [\,].$

Inductive case: height $= P_{Struct}(c) = n + 1.$

Sequence: $c'; c''$**.** *For the typing rule Sequence, we have that:*

(H_1) $\Gamma \vdash c' : \tau'\ cmd.$

(H_2) $\Gamma \vdash c'' : \tau'\ cmd.$

By (IH_2) and the semantics rule Sequence, we have that:

(H_3) $\mu \vdash c' \Rightarrow^{t'} \mu'.$

(H_4) $\mu' \vdash c'' \Rightarrow^{t''} \mu''.$

(H_5) $\mu \vdash c'; c'' \Rightarrow^{t' \cdot t''} \mu''.$

Concerning c', from the inductive hypothesis on c, it follows that:

(H_6) $\mu =_\tau^\Gamma \mu'$.

(H_7) $\mathsf{filter}_\tau(\mathsf{t}') = [\,]$.

Concerning c", from the inductive hypothesis on c, it follows that:

(H_8) $\mu' =_\tau^\Gamma \mu''$.

(H_9) $\mathsf{filter}_\tau(\mathsf{t}') = [\,]$.

By the transitive property of $=_\tau^\Gamma$ *and since* $\mu =_\tau^\Gamma \mu'$ *and* $\mu' =_\tau^\Gamma \mu''$*, we can conclude* $\mu =_\tau^\Gamma \mu''$.

In the semantics rule Sequence, $t' \cdot t''$ *denotes the concatenation of two traces* t' *and* t''. *Since* $\mathsf{filter}_\tau(\mathsf{t}') = [\,]$ *by (H_7) and* $\mathsf{filter}_\tau(\mathsf{t}'') = [\,]$ *by (H_9), we conclude that* $\mathsf{filter}_\tau(\mathsf{t}' \cdot \mathsf{t}'') = [\,]$.

While: while e do c**.** *For the typing rule While, we have that:*

(H_1) $\Gamma \vdash c : \tau'\ cmd$.

(H_2) $\Gamma \vdash e : \tau'$.

By (H_1) and because the height of $\Gamma \vdash$ while e do c *is* $n+1$*, we know that the height of the typing derivation tree of (H_1) is* $\leq n$*. By applying the inductive hypothesis on c, we get that:*

(H_3) $\mu =_\tau^\Gamma \mu'$.

(H_4) $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

We want to prove that the inductive hypothesis works for while e do c *also. By the hypothesis of Lemma 2, we have that if:*

(H_0') $\Gamma \vdash$ while e do c $: \tau'$ cmd.

(H_1') $\mu \vdash$ while e do c $\Rightarrow^{\mathsf{t}\cdot\mathsf{t}'} \mu''$.

(H_2') $\tau' \not\leq \tau$.

Then we want to prove that:

(G_1*)* $\mu =_\tau^\Gamma \mu''$.

(G_2*)* $\mathsf{filter}_\tau(\mathsf{t} \cdot \mathsf{t}') = [\,]$.

By the semantics rule of Loop, we have that:

(H_3'*)* $\mu \vdash e \Rightarrow v$.

(H_4'*)* $\mu \vdash c \Rightarrow^t \mu'$.

(H_5'*)* $\mu' \vdash \mathsf{while\ e\ do\ c} \Rightarrow^{\mathsf{t}'} \mu''$.

Base case. *The only possibility for the semantics tree to be of height* $= 2$ *is when* $v = False$.

$$\frac{\mu \vdash e \Rightarrow False}{\mu \vdash \mathsf{while\ e\ do\ c} \Rightarrow^{[\,]} \mu}$$

By (H_3*) we conclude that* $\mu =_\tau^\Gamma \mu$. *We also conclude that* $\mathsf{filter}_\tau([\,]) = [\,]$.

Assuming that our inductive hypothesis holds for the case of while when evaluating the height of the semantics tree $\leq m$, *we want to prove the case of While with height* $= m + 1$.

$$\frac{\mu \vdash e \Rightarrow True \qquad (1)\ \mu \vdash c \Rightarrow^t \mu'' \qquad (H_6')\ \mu'' \vdash \mathsf{while\ e\ do\ c'} \Rightarrow^{\mathsf{t}'} \mu'}{\mu \vdash \mathsf{while\ e\ do\ c'} \Rightarrow^{\mathsf{t} \cdot \mathsf{t}'} \mu'}$$

By applying the structural induction to (1), *we have that* $\mu =_\tau^\Gamma \mu''$ *and* $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$. *By applying the inductive hypothesis on* while e do c *by (*H_6'*), we can conclude that* $\mu'' =_\tau^\Gamma \mu'$ *and* $\mathsf{filter}_\tau(\mathsf{t}') = [\,]$. *By applying the transitive property on* $=_\tau^\Gamma$, *we can conclude that* $\mu'' =_\tau^\Gamma \mu'$ *which is* (G_1). *Moreover, since* $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$ *and* $\mathsf{filter}_\tau(\mathsf{t}') = [\,]$, *and* $t \cdot t'$ *is a concatenation, then* $\mathsf{filter}_\tau(\mathsf{t} \cdot \mathsf{t}') = [\,]$ *which is* (G_2).

If: if e then c′ else c″**.** *For the typing rule If, we have that:*

(H_1*)* $\Gamma \vdash c' : \tau'\ cmd.$

(H_2*)* $\Gamma \vdash c'' : \tau'\ cmd.$

(H_3*)* $\Gamma \vdash e : \tau'.$

By (H_1*) and the inductive hypothesis on c′, we get that:*

(H_4*)* $\mu =_\tau^\Gamma \mu'.$

(H_5*)* $\mathsf{filter}_\tau(\mathsf{t}) = [\,].$

By (H_1*) and the inductive hypothesis on c″, we get that:*

(H_6*)* $\mu =_\tau^\Gamma \mu'.$

(H_7*)* $\mathsf{filter}_\tau(\mathsf{t}') = [\,].$

Concerning if e then c′ else c″*, by the hypothesis of Lemma 2, we have that if:*

(H_8*)* $\Gamma \vdash$ if e then c′ else c″ $: \tau'$ cmd.

(H_9*)* $\mu \vdash$ if e then c′ else c″ $\Rightarrow^{\mathsf{t}} \mu'.$

(H_{10}*)* $\mu \vdash$ if e then c′ else c″ $\Rightarrow^{\mathsf{t}} \mu'.$

(H_{11}*)* $\tau' \not\leq \tau.$

Then we want to prove that:

(G_1*)* $\mu =_\tau^\Gamma \mu'.$

(G_2*)* $\mathsf{filter}_\tau(\mathsf{t}) = [\,].$

By the semantics rule of Branch- True, we have that:

(H_{12}*)* $\mu \vdash e \Rightarrow v.$

(H_{13}) $\mu \vdash c' \Rightarrow^{t} \mu'$.

(H_{14}) $t = [\,]$.

By (H_3) and (H_4) we can conclude that $\mu =_{\tau}^{\Gamma} \mu'$. *By ($H_5$) and ($H_{14}$) we conclude that* $\mathsf{filter}_{\tau}(\mathsf{t}) = [\,]$.

By the semantics rule of Branch- False, we have that:

(H_{15}) $\mu \vdash e \Rightarrow v$.

(H_{16}) $\mu \vdash c'' \Rightarrow^{t} \mu'$.

(H_{17}) $t = [\,]$.

By (H_3) and (H_5) we can conclude that $\mu =_{\tau}^{\Gamma} \mu'$. *By ($H_6$) and ($H_{17}$) we conclude that* $\mathsf{filter}_{\tau}(\mathsf{t}) = [\,]$. ■

A.3 Proof of Theorem 2

Proof. By Definition 26, for p to be NI_{τ}^{Γ}, we need to prove that:

(G_1) $\mu_0' =_{\tau}^{\Gamma} \mu_1'$.

(G_2) $\mathsf{filter}_{\tau}(\mathsf{t}_0) = \mathsf{filter}_{\tau}(\mathsf{t}_1)$.

We prove this theorem by induction on the height of typing derivation tree $\Gamma \vdash p$.

Case 1. *Base case: height = 2*

Subcase 1.1. $p \stackrel{\Delta}{=} x := e'$.

Concerning (G_1) and (G_2), by Definition 26, we have that:

(H_1) $\mu_0 =_{\tau}^{\Gamma} \mu_1$.

(H_2) $\mu_i \vdash x := e' \Rightarrow^{t_i} \mu_i'$.

Moreover, for the semantics rule Update we have that:

(H3) $t_i = [\,]$.

Hence, by the semantics rule Update and (H2), we have that:

(H4) $\mu_i \vdash e' \Rightarrow v_i$.

(H5) $\mu_i' = \mu_i[x := v_i]$.

By the hypothesis of Theorem 2, we have:

(H6) $\Gamma \vdash x := e' : \tau'\ cmd$.

By (H6) and the typing rule Assign, we have that:

(H7) $\Gamma \vdash x : \tau'\ var$.

(H8) $\Gamma \vdash e' : \tau'$.

By (H7) and the typing rule Var, we have:

(H9) $\Gamma(x) = \tau'\ var$.

Depending on τ', we have two cases:

(H10) $\tau' \leq \tau$.

By Lemma 1 that can be applied due to (H_1), (H_8), (H_{10}) *and* (H_4) *then*
$(H_{11}) : v_0 = v_1$.
To prove (G_1)*, we rely on the Definition 24 that states that* $\mu_0 =_\tau^\Gamma \mu_1$*, if* $\forall x \in \mu_0$ *such that* $\Gamma(x) = \tau'\ var$ *and* $\tau' \leq \tau$ *then* $\mu_0(x) = \mu_1(x)$.
Since (H_1) *holds and the only variable in which* μ_i *and* μ_i' *are different is x by* (H_5)*, then we need to prove that* $\mu_0(x) = \mu_1(x)$ *which holds by* (H_{11}).
To prove (G_2) *we rely on the Definition 25 and* (H_3)*. Since* $t_0 = t_1 = [\,]$ *then* $\mathsf{filter}_\tau(\mathsf{t_0}) = \mathsf{filter}_\tau(\mathsf{t_1})$.

Since we proved (G_1) *and* (G_2)*, then p is* NI_τ^Γ.

(H_{12}) $\tau' \not\leq \tau$.

To prove (G_1)*, we rely on the Definition 24 that states that* $\mu_0 =_\tau^\Gamma \mu_1$*, if* $\forall y \in \mu_0$ *such that* $\Gamma(y) = \tau'\ var$ *and* $\tau' \leq \tau$ *then* $\mu_0(s) = \mu_1(y)$*. For* (H_{12})*, we have that* $\tau' \not\leq \tau$*. Since we are only interested by variables with security level less or equal than* τ*, we can conclude by* (H_5) *and* (H_1) *that* $\mu_0' =_\tau^\Gamma \mu_1'$.

To prove (G_2) *we rely on the Definition 25 and* (H_3)*. Since* $t_0 = t_1 = [\,]$ *then* $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

Since we proved (G_1) *and* (G_2)*, then* p *is* NI_τ^Γ.

Subcase 1.2. $p \stackrel{\Delta}{=} k := KG(\{n_1 \ldots n_i\}, k')$.

Concerning (G_1) *and* (G_2)*, by Definition 26, we have that:*

(H_1) $\mu_0 =_\tau^\Gamma \mu_1$.

(H_2) $\mu_i \vdash k := KG(\{n_1 \ldots n_i\}, k') \Rightarrow^{t_i} \mu_i'$.

By the semantics rule Update we have that:

(H_3) $t_i = [\,]$.

Moreover, by the semantics rule Update and (H_2)*, we have that:*

(H_4) $\mu_i \vdash KG(\{n_1 \ldots n_i\}, k') \Rightarrow v_i$.

(H_5) $\mu_i' = \mu_i[x := v_i]$.

By the hypothesis of Theorem 2, we have:

(H_6) $\Gamma \vdash k := KG(\{n_1 \ldots n_i\}, k') : \tau'\ cmd$.

By the typing rule Kvar-Assign, we have that:

(H_7) $\Gamma(k) = \tau'\ Kvar(n_1, \ldots, n_i)$.

(H_8) $\Gamma \vdash n_j : \tau'\ Nvar(n_j)$.

(H_9) $\Gamma k' : \top\ MKvar$.

To prove (G_1), we rely on the Definition 24 that states that $\mu_0 =_\tau^\Gamma \mu_1$, *if* $\forall y \in \mu_0$ *such that* $\Gamma(y) = \tau'\ var$ *and* $\tau' \leq \tau$ *then* $\mu_0(y) = \mu_1(y)$. *For* (H_7), $\Gamma(k) = \tau'\ Kvar(n_1, ..., n_i)$. *Since we are only interested in variables of type* $\tau'\ var$, *we conclude by* (H_5) *and* (H_1) *that* $\mu_0' =_\tau^\Gamma \mu_1'$.

To prove (G_2) we rely on the Definition 25 and (H_3). *Since* $t_0 = t_1 = [\,]$ *then* $\mathsf{filter}_\tau(\mathsf{t_0}) = \mathsf{filter}_\tau(\mathsf{t_1})$.

Since we proved (G_1) and (G_2), then p *is* NI_τ^Γ.

Subcase 1.3. $p \stackrel{\Delta}{=} \mathsf{sbroadcast}(\{\mathsf{n_1}, ..., \mathsf{n_j}\}, \mathsf{k}, \mathsf{e})$.

Concerning (G_1) and (G_2), by Definition 26, we have that:

(H_1) $\mu_0 =_\tau^\Gamma \mu_1$.

(H_2) $\mu_i \vdash \mathsf{sbroadcast}(\{\mathsf{n_1}, ..., \mathsf{n_j}\}, \mathsf{k}, \mathsf{e}) \Rightarrow^{\mathsf{t_i}} \mu_\mathsf{i}$.

By the semantics rule Secure Broadcast, we have:

(H_3) $t_i = \mathsf{BEnc}(\{\mathsf{n_1}, ..., \mathsf{n_k}\}, \mathsf{vk}, \mathsf{v}')$.

(H_4) $\mu_i \vdash k \Rightarrow vk_i$.

(H_5) $\mu_i \vdash e \Rightarrow "m_i"$.

By the hypothesis of Theorem 2, we have:

(H_6) $\Gamma \vdash \mathsf{sbroadcast}(\{\mathsf{n_1}, ..., \mathsf{n_j}\}, \mathsf{k}, \mathsf{e}) : \tau'\ \mathsf{cmd}$.

By (H_6) *and the typing rule SBroadcast, we have:*

(H_7) $\Gamma \vdash e : \tau'$.

(H_8) $\Gamma \vdash n_i : \tau\ Nvar(n_i)$.

(H_9) $\Gamma \vdash K : \tau\ Kvar(n_1, ..., n_j)$.

To prove (G_1), we rely on the Definition 24 that states that $\mu_0 =_\tau^\Gamma \mu_1$, *if* $\forall y \in \mu_0$ *such that* $\Gamma(y) = \tau'\ var$ *and* $\tau' \leq \tau$ *then* $\mu_0(y) = \mu_1(y)$. *For* ($H_7 - H_9$), *all the variables types are*

different than τ' var. Since we are only interested in variables of type τ' var, (G_1) follows by (H_1) and (H_2).

To prove (G_2), we have two cases:

(H_{10}) $\Gamma(n_1) \sqcap \Gamma(n_2) ... \sqcap \Gamma(n_k) \leq \tau'$

Being $\mu'_0 =^{\Gamma}_{\tau} \mu'_1$ and by (H_2) and(H_3) we have that $t_0 = t_1$. For an execution that starts with μ_0, $t_0 = \mathsf{BEnc}(\{\mathsf{n}_1,...,\mathsf{n}_k\},\mathsf{vk}_0,\mathsf{v}')$ and for an execution that starts with μ_1, $t_1 = \mathsf{BEnc}(\{\mathsf{n}_1,...,\mathsf{n}_k\},\mathsf{vk}_1,\mathsf{v}')$. By ($H_{11}$), $\Gamma(n_1) \sqcap \Gamma(n_2) ... \sqcap \Gamma(n_k) \leq \tau'$ then $\mathsf{filter}(\mathsf{t}_0) = [\{\mathsf{n}_1,...,\mathsf{n}_k\},\mathsf{v}']$ and $\mathsf{filter}(\mathsf{t}_1) = [\{\mathsf{n}_1,...,\mathsf{n}_k\},\mathsf{v}']$. It results that $\mathsf{filter}(\mathsf{t}_0) = \mathsf{filter}(\mathsf{t}_1)$ and therefore we prove (G_2).

Since we proved (G_1) and (G_2), then p is NI^{Γ}_{τ}.

(H_{11}) $\Gamma(n_1) \sqcap \Gamma(n_2) ... \sqcap \Gamma(n_k) \not\leq \tau'$

Being $\mu'_0 =^{\Gamma}_{\tau} \mu'_1$ and by (H_2) and(H_3) we have that $t_0 = t_1$. For an execution that starts with μ_0, $t_0 = \mathsf{BEnc}(\{\mathsf{n}_1,...,\mathsf{n}_k\},\mathsf{vk}_0,\mathsf{v}')$ and for an execution that starts with μ_1, $t_1 = \mathsf{BEnc}(\{\mathsf{n}_1,...,\mathsf{n}_k\},\mathsf{vk}_1,\mathsf{v}')$. By ($H_{11}$), $\Gamma(n_1) \sqcap \Gamma(n_2) ... \sqcap \Gamma(n_k) \not\leq \tau'$, then $\mathsf{filter}_{\tau}(\mathsf{t}_0) = [\,]$ and $\mathsf{filter}_{\tau}(\mathsf{t}_1) = [\,]$. It results that $\mathsf{filter}_{\tau}(\mathsf{t}_0) = \mathsf{filter}_{\tau}(\mathsf{t}_1)$ and therefore we prove (G_2).

Since we proved (G_1) and (G_2), then p is NI^{Γ}_{τ}.

Case 2. *Inductive case: height= $n+1$*

Subcase 2.1. $p \stackrel{\Delta}{=} c';c''$.

We want to prove that $\Gamma \vdash c';c''$ is NI^{Γ}_{τ}. We assume that c' and c'' are of height $\leq n$, and $c';c''$ of height $n+1$.

For the typing rule Sequence, we have that:

(H_1) $\Gamma \vdash c' : \tau'\ cmd$.

(H_2) $\Gamma \vdash c'' : \tau'\ cmd$.

For the semantics rule Sequence, we have that:

(H_3) $\mu_i \vdash c' \Rightarrow^{t'_i} \mu'_i$.

(H4) $\mu'_i \vdash c'' \Rightarrow^{t''_i} \mu''_i$.

Concerning c', from the inductive hypotheses, it follows that:

(H5) $\mu_{c'_0} =^{\Gamma}_{\tau} \mu_{c'_1}$.

(H6) $\mu_{c'_0} \vdash c \Rightarrow^{t'_0} \mu'_{c'_0}$

(H7) $\mu_{c'_1} \vdash c \Rightarrow^{t'_1} \mu'_{c'_1}$

(H8) $\mu'_{c'_0} =^{\Gamma}_{\tau} \mu'_{c'_1}$.

(H9) $\mathsf{filter}_\tau(\mathsf{t}'_0) = \mathsf{filter}_\tau(\mathsf{t}'_1)$.

It follows that c' is NI^{Γ}_{τ}.

Concerning c'', from the inductive hypotheses, it follows that:

(H10) $\mu_{c''_0} =^{\Gamma}_{\tau} \mu_{c''_1}$.

(H11) $\mu_{c''_0} \vdash c \Rightarrow^{t''_0} \mu'_{c''_0}$

(H12) $\mu_{c''_1} \vdash c \Rightarrow^{t''_1} \mu'_{c''_1}$

(H13) $\mu'_{c''_0} =^{\Gamma}_{\tau} \mu'_{c''_1}$.

(H14) $\mathsf{filter}_\tau(\mathsf{t}''_0) = \mathsf{filter}_\tau(\mathsf{t}''_1)$.

It follows that c'' is NI^{Γ}_{τ}.

In the semantics rule Sequence, we use $t' \cdot t''$ to denote the concatenation of two traces t' and t''. By (H9) and (H14) we have that $t'_0 = t'_1$ and $t''_0 = t''_1$, which implies that $t'_0 \cdot t''_0 = t'_1 \cdot t''_1$. We conclude that $\mathsf{filter}_\tau(\mathsf{t}'_0 \cdot \mathsf{t}''_0) = \mathsf{filter}_\tau(\mathsf{t}'_1 \cdot \mathsf{t}''_1)$.

Since c', c'' are NI^{Γ}_{τ} and $\mathsf{filter}_\tau(\mathsf{t}'_0 \cdot \mathsf{t}''_0) = \mathsf{filter}_\tau(\mathsf{t}'_1 \cdot \mathsf{t}''_1)$*, we conclude that $\Gamma \vdash c'; c''$ is NI^{Γ}_{τ}.*

Subcase 2.2. $p \stackrel{\Delta}{=} \mathsf{while\ e\ do\ c}$.

For the typing rule While, we have that:

(H1) $\Gamma \vdash c : \tau'\ cmd$.

(H₂) $\Gamma \vdash e : \tau'$.

In the following, we show by induction that c is NI_τ^Γ.

By (H_1) *and because the height of* $\Gamma \vdash$ while e do c *is* $n+1$*, we know that the height of the typing derivation tree of* (H_1) *is* $\leq n$*. Hence, we can apply the inductive hypothesis and get:*

(H₃) $\mu_0 =_\tau^\Gamma \mu_1$.

(H₄) $\mu_0 \vdash c \Rightarrow^{t_0} \mu_0'$.

(H₅) $\mu_1 \vdash c' \Rightarrow^{t_1} \mu_1'$.

Then,

(H₆) $\mu_0' =_\tau^\Gamma \mu_1'$.

(H₇) $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

We want to prove that while e do c *is* NI_τ^Γ*. By the hypothesis of the theorem, we have that if:*

(H'₀) $\Gamma \vdash$ while e do c $: \tau'$ cmd.

(H'₁) $\mu_0 =_\tau^\Gamma \mu_1$.

(H'₂) $\mu_0 \vdash$ while e do c $\Rightarrow^{\mathsf{t}_0} \mu_0'$.

(H'₃) $\mu_1 \vdash$ while e do c $\Rightarrow^{\mathsf{t}_1} \mu_1'$.

Then, we want to prove that:

(G₀) $\mu_0' =_\tau^\Gamma \mu_1'$.

(G₁) $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

By (H_2') *and the semantics rule of Loop, we have that:*

(H'₄) $\mu_0 \vdash e \Rightarrow v_0$.

By (H'_3) and the semantics rule of Loop, we have that:

(H'_5) $\mu_1 \vdash e \Rightarrow v_1$.

Depending on τ', we have two cases:

Subcase 2.2.1. $\Gamma \vdash e : \tau'$, $\tau' \leq \tau$.

(H'_6) $\tau' \leq \tau$.

In the case of (H'_6), we check if we can apply the Subtype rule on While command. We would like to check if the command While of type τ' cmd can be typable as τ'' cmd, where $\tau'' \not\leq \tau'$. If the Subtype rule can be applied, the the proof of this case is analogous to Subcase 3.2.2.

Otherwise, if the Subtype rule cannot be applied, then by Lemma 1 that can be applied on (H'_1), (H_2), (H'_6), (H'_4) and (H'_5), we conclude that $v_0 = v_1$.

We prove this case by induction on the height of the semantics tree of (H'_2). (We do not show this formally, but we rely on the fact that the height of (H'_2) is equal to the height of (H'_3) by Lemma 1).

Base case: height = 2. *The only possibility for the semantics tree to be of height = 2 is:*

- $v_0 = v_1 = False$.

$$\frac{\mu_0 \vdash e \Rightarrow False}{\mu_0 \vdash \text{while e do c} \Rightarrow^{t_0} \mu_0}$$

$$\frac{\mu_1 \vdash e \Rightarrow False}{\mu_1 \vdash \text{while e do c} \Rightarrow^{t_1} \mu_1}$$

We have that $\mu'_0 =^{\Gamma}_{\tau} \mu_0$ and $\mu'_1 =^{\Gamma}_{\tau} \mu_1$. We conclude by (H'_1) that $\mu'_0 =^{\Gamma}_{\tau} \mu'_1$. Moreover, since $t_0 = t_1 = [\,]$ then $\text{filter}_\tau(t_0) = \text{filter}_\tau(t_1)$.

Assuming that our inductive hypothesis holds for the case of While when evaluating the height of semantics tree ≤ m with $\Gamma \vdash e : \tau', \tau' \leq \tau$*, let us prove the case of While with height* $= m + 1$.

Inductive case: height = m+1.

$$\frac{\mu_0 \vdash e \Rightarrow True \qquad \mu_0 \vdash c \Rightarrow^{t_0} \mu_0'' \qquad (H_7') \ \mu_0'' \vdash \text{while e do c} \Rightarrow^{t_0'} \mu_0'}{\mu_0 \vdash \text{while e do c} \Rightarrow^{t_0 \cdot t_0'} \mu_0'}$$

The height of (H_2') *is* $m + 1$.

$$\frac{\mu_1 \vdash e \Rightarrow True \qquad \mu_1 \vdash c \Rightarrow^{t_1} \mu_1'' \qquad (H_8') \ \mu_1'' \vdash \text{while e do c} \Rightarrow^{t_1'} \mu_1'}{\mu_1 \vdash \text{while e do c} \Rightarrow^{t_1 \cdot t_1'} \mu_1'}$$

The height of (H_3') *is* $m + 1$.

By the previous induction on c, we have that $\mu_0'' =_\tau^\Gamma \mu_1''$ *and* $\text{filter}_\tau(t_0) = \text{filter}_\tau(t_1)$*, through* $\mu_0 =_\tau^\Gamma \mu_1$*,* $\mu_0 \vdash c' \Rightarrow^{t_0} \mu_0''$ *and* $\mu_1 \vdash c' \Rightarrow^{t_1} \mu_1''$.

Moreover, by induction on while e do c *by* (H_7') *and* (H_8')*, we can conclude that* $\mu_0' =_\tau^\Gamma \mu_1'$ *which is already our goal* (G_1) *and* $\text{filter}_\tau(t_0') = \text{filter}_\tau(t_1')$*. This is because we have that* $\mu_0'' =_\tau^\Gamma \mu_1''$ *and* $\mu_0'' \vdash \text{while e do c} \Rightarrow^{t_0'} \mu_0'$ *and* $\mu_1'' \vdash \text{while e do c} \Rightarrow^{t_1'} \mu_1'$.

Since $t_i \cdot t_i'$ *is the concatenation of two traces* t_i *and* t_i'*, and since* $\text{filter}_\tau(t_0) = \text{filter}_\tau(t_1)$ *and* $\text{filter}_\tau(t_0') = \text{filter}_\tau(t_1')$*, we conclude that* $\text{filter}_\tau(t_0 \cdot t_0') = \text{filter}_\tau(t_1 \cdot t_1')$ *from which* (G_2) *follows.*

Since (G_1) *and* (G_2) *are satisfied, then* while e do c *is* NI_τ^Γ.

Subcase 2.2.2. $\Gamma \vdash e : \tau', \ \tau' \not\leq \tau$.

(H_9') $\tau' \not\leq \tau$.

By Lemma 2 that can be applied on (H_1)*,* (H_4)*,* (H_5) *and* (H_9')*, we conclude that* $\mu_i =_\tau^\Gamma \mu_i'$.

We prove that $\mu_0' =_\tau^\Gamma \mu_1'$ *by applying the transitive property on* $=_\tau^\Gamma$*:*

- *By* (H'_1) *we have that* (1) $\mu_0 =_\tau^\Gamma \mu_1$.
- *By Lemma 2 we have that* (2) $\mu_0 =_\tau^\Gamma \mu'_0$ *and* (3) $\mu_1 =_\tau^\Gamma \mu'_1$.
- *By* (1) *and* (2) *we have that* (4) $\mu_1 =_\tau^\Gamma \mu'_0$.
- *By* (1) *and* (3) *we have that* (5) $\mu_0 =_\tau^\Gamma \mu'_1$.
- *By* (4) *and* (5) *we conclude that* (6) $\mu'_0 =_\tau^\Gamma \mu'_1$.

By applying structural induction, we have that $\mathsf{filter}_\tau(\mathsf{t}'_0) = \mathsf{filter}_\tau(\mathsf{t}'_1)$. *By previous induction on c we have that* $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$. *Since* $t_i \cdot t'_i$ *is the concatenation of two traces* t_i *and* t'_i, *and since* $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$ *and* $\mathsf{filter}_\tau(\mathsf{t}'_0) = \mathsf{filter}_\tau(\mathsf{t}'_1)$, *we conclude that* $\mathsf{filter}_\tau(\mathsf{t}_0 \cdot \mathsf{t}'_0) = \mathsf{filter}_\tau(\mathsf{t}_1 \cdot \mathsf{t}'_1)$ *from which* (G_2) *follows.*

Since (G_1) *and* (G_2) *are satisfied, then* while e do c′ *is* NI_τ^Γ.

Subcase 2.3. $p \stackrel{\Delta}{=} \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{c}'\ \mathsf{else}\ \mathsf{c}''$.

For the typing rule If, we have that:

(H_1) $\Gamma \vdash e : \tau'$.

(H_2) $\Gamma \vdash c' : \tau'\ cmd$.

(H_3) $\Gamma \vdash c'' : \tau'\ cmd$.

For the semantics rule Branch, we have that:

(H_4) $\mu_i \vdash e \Rightarrow v_i$.

(H_5) $\mu_i \vdash c' \Rightarrow_{t_i} \mu'_i$.

(H_6) $\mu_i \vdash c'' \Rightarrow_{t_i} \mu'_i$.

Concerning c′, by induction, we conclude that:

(H_7) $\mu'_{c'_0} =_\tau^\Gamma \mu'_{c'_1}$.

(H_8) $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

It follows that c' *is* NI_{τ}^{Γ}.

Concerning c''*, by induction, we conclude that:*

(H_8) $\mu'_{c''_0} =_{\tau}^{\Gamma} \mu'_{c''_1}$.

(H_9) $\mathsf{filter}_{\tau}(\mathsf{t}_0) = \mathsf{filter}_{\tau}(\mathsf{t}_1)$.

It follows that c'' *is* NI_{τ}^{Γ}.

Depending on τ'*, we have two cases:*

Subcase 2.3.1. $\Gamma \vdash e : \tau', \tau' \leq \tau$.

(H_{10}) $\tau' \leq \tau$.

In the case of (H_{10})*, we check if we can apply the Subtype rule on If command. We would like to check if the command If of type* τ' *cmd can be typable as* τ'' *cmd, where* $\tau'' \not\leq \tau'$*. If the Subtype rule can be applied, the the proof of this case is analogous to Subcase 3.2.5.*

Otherwise, if the Subtype rule cannot be applied, then by Lemma 1, we can conclude that:

(H_{11}) $v_0 = v_1$.

$$\frac{\mu_0 \vdash e \Rightarrow True \qquad \mu_0 \vdash c' \Rightarrow^{t_0} \mu'_0}{\mu_0 \vdash \quad If\ e\ \mathsf{thenc'\ else\ c''} \Rightarrow^{\mathsf{t}_0} \mu'_0}$$

$$\frac{\mu_1 \vdash e \Rightarrow True \qquad \mu_1 \vdash c' \Rightarrow^{t_1} \mu'_1}{\mu_1 \vdash \quad If\ e\ \mathsf{thenc'\ else\ c''} \Rightarrow^{\mathsf{t}_1} \mu'_1}$$

We have that $\mu'_0 =_{\tau}^{\Gamma} \mu_0$ *and* $\mu'_1 =_{\tau}^{\Gamma} \mu_1$*. Since* $\mu_0 =_{\tau}^{\Gamma} \mu_1$*, we conclude that* $\mu'_0 =_{\tau}^{\Gamma} \mu'_1$*. Since* $t_0 = t_1 = [\,]$*, then* $\mathsf{filter}_{\tau}(\mathsf{t}_0) = \mathsf{filter}_{\tau}(\mathsf{t}_1)$.

$$\frac{\mu_0 \vdash e \Rightarrow False \qquad \mu_0 \vdash c'' \Rightarrow^{t_0} \mu_0'}{\mu_0 \vdash \qquad If\ e\ \mathsf{thenc'\ else\ c''} \Rightarrow^{\mathsf{t_0}} \mu_0'}$$

$$\frac{\mu_1 \vdash e \Rightarrow false \qquad \mu_1 \vdash c'' \Rightarrow^{t_1} \mu_1'}{\mu_1 \vdash \qquad If\ e\ \mathsf{thenc'\ else\ c''} \Rightarrow^{\mathsf{t_1}} \mu_1'}$$

We have that $\mu_0' =_\tau^\Gamma \mu_0$ *and* $\mu_1' =_\tau^\Gamma \mu_1$. *Since* $\mu_0 =_\tau^\Gamma \mu_1$, *we conclude that* $\mu_0' =_\tau^\Gamma \mu_1'$. *Since* $t_0 = t_1 = [\,]$, *then* $\mathsf{filter}_\tau(\mathsf{t_0}) = \mathsf{filter}_\tau(\mathsf{t_1})$.

Subcase 2.3.2. $\Gamma \vdash e : \tau', \tau' \not\leq \tau$.

(H_{12}) $\tau' \not\leq \tau$.

By Lemma 2, we conclude that $\mu_i =_\tau^\Gamma \mu_i'$.
We prove that $\mu_0' =_\tau^\Gamma \mu_1'$ *by applying the transitive property on* $=_\tau^\Gamma$.

- *We have that (1)* $\mu_0' =_\tau^\Gamma \mu_0$.
- *We have that (2)* $\mu_1' =_\tau^\Gamma \mu_1$.
- *We have that (3)* $\mu_0 =_\tau^\Gamma \mu_1$.
- *By (1) and (3) we have that (4)* $\mu_1 =_\tau^\Gamma \mu_0'$.
- *By (2) and (3) we have that (5)* $\mu_0 =_\tau^\Gamma \mu_1'$.
- *By (4) and (5) we conclude that (6)* $\mu_0' =_\tau^\Gamma \mu_1'$.

Since $\mathsf{filter}_\tau(\mathsf{t_0}) = \mathsf{filter}_\tau(\mathsf{t_1})$ *and* $\mu_0' =_\tau^\Gamma \mu_1'$, *then p is* NI_τ^Γ.

■

APPENDIX B

PROOF OF THEOREM 3 OF SECTION 5.2

To prove Theorem 3, we rely on Lemma 1 and 2. Since Lemma 1 is analogous to the one in Appendix A, we only prove Lemma 2 and Theorem 3.

B.1 Proof of Lemma 2

Proof. We prove this lemma by structural induction on c with $P_{Struct}(c) = n$. It follows that:

- $P_{Struct}(x := e') = 1$.
- $P_{Struct}(\mathsf{pc} := \mathsf{pc} + 1) = 1$.
- $P_{Struct}(\mathsf{sbroadcast}(\mathsf{L}, \mathsf{e}||\mathsf{pc}, \mathsf{K})) = 1$.
- $P_{Struct}(\mathsf{for\ n} \in \mathsf{L\ Ra}(\mathsf{L}, \mathsf{L}', \mathsf{n})) = 1$.
- $P_{Struct}(c'; c'') = P_{Struct}(c') + P_{Struct}(c'')$.
- $P_{Struct}(\mathsf{while\ e\ do\ c}) = \mathsf{P_{Struct}(c)} + 1$.
- $P_{Struct}(\mathsf{if\ e\ then\ c'\ else\ c''} = \mathsf{MAX}(\mathsf{P_{Struct}(c')}, \mathsf{P_{Struct}(c'')} + 1$.

From the hypothesis of Lemma 2, we have that:

(H_1) $\Gamma \vdash c : \tau'\ cmd$.

(H_2) $\tau' \not\leq \tau$.

(H_3) $\mu \vdash c \Rightarrow^t \mu'$.

And we need to prove that:

(G_1) $\mu =_\tau^\Gamma \mu'$.

(G_2) $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

Case 1 (Base case)**.**

Subcase 1.1 (Assign: $x := e'$.)**.** *By the hypothesis of the lemma, we have that:*

(H_4) $\Gamma \vdash x := e' : \tau'\ cmd$.

By (H_4) and the typing rule Assign, we have that:

(H_5) $\Gamma \vdash x : \tau'\ var$.

(H_6) $\Gamma \vdash e : \tau'$.

By (H_5) and the typing rule Var, we have that:

(H_7) $\Gamma(x) = \tau'\ var$.

By (H_3) and the semantics rule Update, we have that:

(H_8) $\mu \vdash e \Rightarrow v$.

(H_9) $\mu' \vdash x := e \Rightarrow^t \mu[x := v]$.

By the semantics rule Update, we have that:

(H_{10}) $t = [\,]$.

For (H_2), we have that $\tau' \not\leq \tau$. Since the only variable in which μ and μ' are different is x by (H_9) and the security level of the variable x is $\tau' var$, we can conclude that $\mu =_\tau^\Gamma \mu'$ which is (G_1).

To prove (G_2), by (H_{10}), $t = [\,]$. Therefore $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

Subcase 1.2 (Assign-Counter: $\mathsf{pc} := \mathsf{pc} + 1$)**.** *By the hypothesis of the lemma, we have that:*

(H_4) $\Gamma \vdash \mathsf{pc} := \mathsf{pc} + 1 : \bot\ \mathsf{cmd}$.

By (H_4) and the typing rule Assign-Counter, we have that:

(H_5) $\Gamma \vdash \mathsf{pc} : \bot\ \mathsf{Cvar}$.

By (H_3) and the semantics rule Update, we have that:

(H_6) $\mu \vdash \mathsf{pc} + 1 \Rightarrow \mathsf{v}$.

(H_7) $\mu' \vdash \mathsf{pc} := \mathsf{pc} + 1 \Rightarrow^{\mathsf{t}} \mu[\mathsf{pc} := \mathsf{v}]$.

By the semantics rule Update, we have that:

(H_8) $t = [\,]$.

For (H_2), we have that $\tau' \not\leq \tau$. Since the only variable in which μ and μ' are different is pc *by (H_7) and the security level of the variable* pc *is $\bot' Cvar$, we can conclude that $\mu =_\tau^\Gamma \mu'$ which is (G_1).*

To prove (G_2), by (H_8), $t = [\,]$. Therefore $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

Subcase 1.3 (SBroadcast: $\mathsf{sbroadcast}(\mathsf{L}, \mathsf{e}||\mathsf{pc}, \mathsf{K})$)**.** *By the hypothesis of the lemma, we have that:*

(H_4) $\Gamma \vdash \mathsf{sbroadcast}(\mathsf{L}, \mathsf{e}||\mathsf{pc}, \mathsf{K}) : \tau'\ \mathsf{cmd}$.

By (H_4) and the typing rule SBroadcast, we have that:

(H_5) $\Gamma \vdash e : \tau'$.

(H_6) $\Gamma \vdash L : \tau'\ Lvar$.

(H_7) $\Gamma \vdash K : \top\ Kvar$.

(H_8) $\Gamma \vdash \mathsf{pc} : \bot\ \mathsf{Cvar}$.

By (H_3) and the semantics rule Secure Broadcast, we have that:

(H_9) $\mu \vdash \mathsf{sbroadcast}(\mathsf{L}, \mathsf{e}||\mathsf{pc}, \mathsf{K}) \Rightarrow^{\mathsf{t}} \mu$.

By the semantics rule Secure Broadcast, we have that:

(H_{10}) $t = \mathsf{BEnc}(\mathsf{L}, \mathsf{vk}, \text{"m"}||\text{"v"})$.

For (H_2), we have that $\tau' \not\leq \tau$. Since the initial memory μ and the final memory μ are equal by (H_9), we conclude that $\mu =_\tau^\Gamma \mu'$ which is (G_1).

To prove (G_2), by (H_{10}), $t = \mathsf{BEnc}(\mathsf{L}, \mathsf{vk}, \text{"m"}||\text{"v"})$. By Definition 27, and by (H_2) and (H_6), $\Gamma(L) \not\leq \tau$, therefore $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

Subcase 1.4 (Remote Attestation: for $\mathsf{n} \in \mathsf{L}\ \mathsf{Ra}(\mathsf{L}, \mathsf{L}', \mathsf{n})$)**.** *By the hypothesis of the lemma, we have that:*

(H_4) $\Gamma \vdash \mathsf{for}\ \mathsf{n} \in \mathsf{L}\ \mathsf{Ra}(\mathsf{L}, \mathsf{L}', \mathsf{n}) : \tau'\ \mathsf{cmd}$.

By (H_4) and the typing rule Remote Attestation, we have that:

(H_5) $\Gamma \vdash L : \tau'\ Lvar$.

(H_6) $\Gamma \vdash L' : \tau''\ Lvar$.

(H_7) $\Gamma \vdash K : \top\ Kvar$.

(H_8) $\Gamma \vdash \mathsf{pc} : \bot\ \mathsf{Cvar}$.

(H_9) $I(\tau'' \leq_I I(\tau')$.

(H_{10}) $C(\tau' \leq_C C(\tau''))$.

By (H_3) and the semantics rule Endorse-Ra, we have that:

(H_{11}) $\mu \vdash \text{for n} \in \text{L Ra}(\text{L}, \text{L}', \text{n}) \Rightarrow^{\text{t}} \mu[\text{L} := \text{L} \setminus \text{S}; \text{L}' := \text{L}' \cup \text{S}]$.

By the semantics rule Endorse-Ra, we have that:

(H_{12}) $t = [\,]$.

For (H_2), we have that $\tau' \not\leq \tau$. By (H_9) and (H_{10}) we have that $\tau' \leq \tau''$, therefore $\tau'' \not\leq \tau$. Since $\tau' \not\leq \tau$ and $\tau'' \not\leq \tau$ and because μ and μ' are different only for variables L of type τ' and L' of type τ'' by (H_9), we conclude that $\mu =_\tau^\Gamma \mu'$ which is (G_1).

To prove (G_2), by (H_{10}), $t = [\,]$. Therefore $\text{filter}_\tau(\text{t}) = [\,]$.

Inductive hypothesis for c *For $P_{Struct}(c) = n$, we have that:*

(IH_1) $\Gamma \vdash c : \tau'\ cmd$.

(IH_2) $\mu \vdash c :\Rightarrow^t \mu'$.

(IH_3) $\tau' \not\leq \tau$.

Then we conclude that:

(IH_4) $\mu =_\tau^\Gamma \mu'$.

(IH_5) $\text{filter}_\tau(\text{t}) = [\,]$.

Inductive case: height $= P_{Struct}(c) = n + 1$.

Sequence: $c';c''$**.** *For the typing rule Sequence, we have that:*

(H_1*)* $\Gamma \vdash c' : \tau'\ cmd.$

(H_2*)* $\Gamma \vdash c'' : \tau'\ cmd.$

By (IH_2*) and the semantics rule Sequence, we have that:*

(H_3*)* $\mu \vdash c' \Rightarrow^{t'} \mu'.$

(H_4*)* $\mu' \vdash c'' \Rightarrow^{t''} \mu''.$

(H_5*)* $\mu \vdash c';c'' \Rightarrow^{t' \cdot t''} \mu''.$

Concerning c', from the inductive hypothesis on c, it follows that:

(H_6*)* $\mu =_\tau^\Gamma \mu'.$

(H_7*)* $\mathsf{filter}_\tau(\mathsf{t}') = [\,].$

Concerning c", from the inductive hypothesis on c, it follows that:

(H_8*)* $\mu' =_\tau^\Gamma \mu''.$

(H_9*)* $\mathsf{filter}_\tau(\mathsf{t}') = [\,].$

By the transitive property of $=_\tau^\Gamma$ *and since* $\mu =_\tau^\Gamma \mu'$ *and* $\mu' =_\tau^\Gamma \mu''$*, we can conclude* $\mu =_\tau^\Gamma \mu''$*.*

In the semantics rule Sequence, $t' \cdot t''$ *denotes the concatenation of two traces* t' *and* t''*. Since* $\mathsf{filter}_\tau(\mathsf{t}') = [\,]$ *by (*H_7*) and* $\mathsf{filter}_\tau(\mathsf{t}'') = [\,]$ *by (*H_9*), we conclude that* $\mathsf{filter}_\tau(\mathsf{t}' \cdot \mathsf{t}'') = [\,]$*.*

While: while e do c**.** *For the typing rule While, we have that:*

(H_1*)* $\Gamma \vdash c : \tau'\ cmd.$

(H_2*)* $\Gamma \vdash e : \tau'.$

By (H_1) and because the height of $\Gamma \vdash$ while e do c *is* $n+1$*, we know that the height of the typing derivation tree of (H_1) is* $\leq n$*. By applying the inductive hypothesis on c, we get that:*

(H_3) $\mu =_\tau^\Gamma \mu'$.

(H_4) $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

We want to prove that the inductive hypothesis works for while e do c *also. By the hypothesis of Lemma 2, we have that if:*

(H'_0) $\Gamma \vdash \mathsf{while\ e\ do\ c} : \tau'\ \mathsf{cmd}$.

(H'_1) $\mu \vdash \mathsf{while\ e\ do\ c} \Rightarrow^{\mathsf{t}\cdot\mathsf{t}'} \mu''$.

(H'_2) $\tau' \not\leq \tau$.

Then we want to prove that:

(G_1) $\mu =_\tau^\Gamma \mu''$.

(G_2) $\mathsf{filter}_\tau(\mathsf{t}\cdot\mathsf{t}') = [\,]$.

By the semantics rule of Loop, we have that:

(H'_3) $\mu \vdash e \Rightarrow v$.

(H'_4) $\mu \vdash c \Rightarrow^t \mu'$.

(H'_5) $\mu' \vdash \mathsf{while\ e\ do\ c} \Rightarrow^{\mathsf{t}'} \mu''$.

Base case. *The only possibility for the semantics tree to be of height* $= 2$ *is when* $v = False$.

$$\frac{\mu \vdash e \Rightarrow False}{\mu \vdash \mathsf{while\ e\ do\ c} \Rightarrow^{[\,]} \mu}$$

By (H_3) *we conclude that* $\mu =_\tau^\Gamma \mu$. *We also conclude that* $\mathsf{filter}_\tau([\,]) = [\,]$.

Assuming that our inductive hypothesis holds for the case of while when evaluating the height of the semantics tree $\leq m$, *we want to prove the case of While with height* $= m + 1$.

$$\frac{\mu \vdash e \Rightarrow True \qquad (1)\ \mu \vdash c \Rightarrow^{t} \mu'' \qquad (H_6')\ \mu'' \vdash \mathsf{while\ e\ do\ c'} \Rightarrow^{\mathsf{t'}} \mu'}{\mu \vdash \mathsf{while\ e\ do\ c'} \Rightarrow^{\mathsf{t}\cdot\mathsf{t'}} \mu'}$$

By applying the structural induction to (1), *we have that* $\mu =_\tau^\Gamma \mu''$ *and* $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$. *By applying the inductive hypothesis on* while e do c *by* (H_6'), *we can conclude that* $\mu'' =_\tau^\Gamma \mu'$ *and* $\mathsf{filter}_\tau(\mathsf{t'}) = [\,]$. *By applying the transitive property on* $=_\tau^\Gamma$, *we can conclude that* $\mu'' =_\tau^\Gamma \mu'$ *which is* (G_1). *Moreover, since* $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$ *and* $\mathsf{filter}_\tau(\mathsf{t'}) = [\,]$, *and* $t \cdot t'$ *is a concatenation, then* $\mathsf{filter}_\tau(\mathsf{t}\cdot\mathsf{t'}) = [\,]$ *which is* (G_2).

If: if e then c′ else c″**.** *For the typing rule If, we have that:*

(H_1) $\Gamma \vdash c' : \tau'\ cmd$.

(H_2) $\Gamma \vdash c'' : \tau'\ cmd$.

(H_3) $\Gamma \vdash e : \tau'$.

By (H_1) *and the inductive hypothesis on* c', *we get that:*

(H_4) $\mu =_\tau^\Gamma \mu'$.

(H_5) $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

By (H_1) *and the inductive hypothesis on* c'', *we get that:*

(H_6) $\mu =_\tau^\Gamma \mu'$.

(H_7) $\mathsf{filter}_\tau(\mathsf{t'}) = [\,]$.

Concerning if e then c′ else c″, *by the hypothesis of Lemma 2, we have that if:*

(H_8) $\Gamma \vdash \mathsf{if\ e\ then\ c'\ else\ c''} : \tau' \mathsf{\ cmd}$.

(H_9) $\mu \vdash \mathsf{if\ e\ then\ c'\ else\ c''} \Rightarrow^{\mathsf{t}} \mu'$.

(H_{10}) $\mu \vdash \mathsf{if\ e\ then\ c'\ else\ c''} \Rightarrow^{\mathsf{t}} \mu'$.

(H_{11}) $\tau' \not\leq \tau$.

Then we want to prove that:

(G_1) $\mu =_\tau^\Gamma \mu'$.

(G_2) $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

By the semantics rule of Branch- True, we have that:

(H_{12}) $\mu \vdash e \Rightarrow v$.

(H_{13}) $\mu \vdash c' \Rightarrow^t \mu'$.

(H_{14}) $t = [\,]$.

By (H_3) *and* (H_4) *we can conclude that* $\mu =_\tau^\Gamma \mu'$. *By* (H_5) *and* (H_{14}) *we conclude that* $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

By the semantics rule of Branch- False, we have that:

(H_{15}) $\mu \vdash e \Rightarrow v$.

(H_{16}) $\mu \vdash c'' \Rightarrow^t \mu'$.

(H_{17}) $t = [\,]$.

By (H_3) *and* (H_5) *we can conclude that* $\mu =_\tau^\Gamma \mu'$. *By* (H_6) *and* (H_{17}) *we conclude that* $\mathsf{filter}_\tau(\mathsf{t}) = [\,]$.

■

B.2 Proof of Theorem 3

Proof. By Definition 26, for p to be NI_{τ}^{Γ}, we need to prove that:

(G_1) $\mu_0' =_{\tau}^{\Gamma} \mu_1'$.

(G_2) $\mathsf{filter}_{\tau}(\mathsf{t}_0) = \mathsf{filter}_{\tau}(\mathsf{t}_1)$.

We prove this theorem by induction on the height of typing derivation tree $\Gamma \vdash p$.

Case 1. *Base case: height = 2*

Subcase 1.1. $p \stackrel{\Delta}{=} x := e'$.

Concerning (G_1) and (G_2), by Definition 26, we have that:

(H_1) $\mu_0 =_{\tau}^{\Gamma} \mu_1$.

(H_2) $\mu_i \vdash x := e' \Rightarrow^{t_i} \mu_i'$.

Moreover, for the semantics rule Update we have that:

(H_3) $t_i = [\,]$.

Hence, by the semantics rule Update and (H_2), we have that:

(H_4) $\mu_i \vdash e' \Rightarrow v_i$.

(H_5) $\mu_i' = \mu_i[x := v_i]$.

By the hypothesis of Theorem 3, we have:

(H_6) $\Gamma \vdash x := e' : \tau'\ cmd$.

By (H_6) and the typing rule Assign, we have that:

(H_7) $\Gamma \vdash x : \tau'\ var$.

(H_8) $\Gamma \vdash e' : \tau'$.

By (H_7) and the typing rule Var, we have:

(H_9) $\Gamma(x) = \tau' \; var$.

Depending on τ', we have two cases:

(H_{10}) $\tau' \leq \tau$.

By Lemma 1 that can be applied due to (H_1), (H_8), (H_{10}) and (H_4) then
(H_{11}): $v_0 = v_1$.
To prove (G_1), we rely on the Definition 24 that states that $\mu_0 =_\tau^\Gamma \mu_1$, if $\forall x \in \mu_0$ such that $\Gamma(x) = \tau' \; var$ and $\tau' \leq \tau$ then $\mu_0(x) = \mu_1(x)$.
Since (H_1) holds and the only variable in which μ_i and μ_i' are different is x by (H_5), then we need to prove that $\mu_0(x) = \mu_1(x)$ which holds by (H_{11}).
To prove (G_2) we rely on the Definition 27 and (H_3). Since $t_0 = t_1 = [\,]$ then $\mathsf{filter}_\tau(\mathsf{t_0}) = \mathsf{filter}_\tau(\mathsf{t_1})$.

Since we proved (G_1) and (G_2), then p is NI_τ^Γ.

(H_{12}) $\tau' \not\leq \tau$.

To prove (G_1), we rely on the Definition 24 that states that $\mu_0 =_\tau^\Gamma \mu_1$, if $\forall y \in \mu_0$ such that $\Gamma(y) = \tau' \; var$ and $\tau' \leq \tau$ then $\mu_0(s) = \mu_1(y)$. For (H_{12}), we have that $\tau' \not\leq \tau$. Since we are only interested by variables with security level less or equal than τ, we can conclude by (H_5) and (H_1) that $\mu_0' =_\tau^\Gamma \mu_1'$.

To prove (G_2) we rely on the Definition 27 and (H_3). Since $t_0 = t_1 = [\,]$ then $\mathsf{filter}_\tau(\mathsf{t_0}) = \mathsf{filter}_\tau(\mathsf{t_1})$.

Since we proved (G_1) and (G_2), then p is NI_τ^Γ.

Subcase 1.2. $p \stackrel{\Delta}{=} \mathsf{pc} := \mathsf{pc} + 1$.

Concerning (G_1) and (G_2), by Definition 26, we have that:

(H_1) $\mu_0 =_\tau^\Gamma \mu_1$.

(H_2) $\mu_i \vdash \mathsf{pc} := \mathsf{pc} + 1 \Rightarrow^{\mathsf{t_i}} \mu_\mathsf{i}'$.

By the semantics rule Update we have that:

(H3) $t_i = [\,]$.

Moreover, by the semantics rule Update and (H2), we have that:

(H4) $\mu_i \vdash \mathsf{pc} + 1 \Rightarrow \mathsf{v_i}$.

(H5) $\mu'_i = \mu_i[x := v_i]$.

By the hypothesis of Theorem 3, we have:

(H6) $\Gamma \vdash \mathsf{pc} := \mathsf{pc} + 1 : \bot\ \mathsf{cmd}$.

By the typing rule Assign-Counter, we have that:

(H7) $\Gamma \vdash \mathsf{pc} : \bot\ \mathsf{Cvar}$.

To prove (G1), we rely on the Definition 24 that states that $\mu_0 =_\tau^\Gamma \mu_1$, *if* $\forall y \in \mu_0$ *such that* $\Gamma(y) = \tau'\ var$ *and* $\tau' \leq \tau$ *then* $\mu_0(y) = \mu_1(y)$. *For (H7),* $\Gamma(\mathsf{pc}) = \bot\ \mathsf{Cvar}$. *Since we are only interested in variables of type* $\tau'\ var$, *we conclude by (H5) and (H1) that* $\mu'_0 =_\tau^\Gamma \mu'_1$.

To prove (G2) we rely on the Definition 27 and (H3). Since $t_0 = t_1 = [\,]$ *then* $\mathsf{filter}_\tau(\mathsf{t_0}) = \mathsf{filter}_\tau(\mathsf{t_1})$.

Since we proved (G1) and (G2), then p is NI_τ^Γ.

Subcase 1.3. $p \triangleq \mathsf{sbroadcast(L, e||pc, K)}$.

Concerning (G1) and (G2), by Definition 26, we have that:

(H1) $\mu_0 =_\tau^\Gamma \mu_1$.

(H2) $\mu_i \vdash \mathsf{sbroadcast(L, e||pc, K)} \Rightarrow^{\mathsf{t_i}} \mu_\mathsf{i}$.

By the semantics rule Secure Broadcast, we have:

(H3) $t_i = \mathsf{BEnc(L, vk_i, "m"||"v")}$.

(H4) $\mu_i \vdash K \Rightarrow v'_i$.

(H5) $\mu_i \vdash e \Rightarrow "m_i"$.

(H6) $\mu_i \vdash \mathsf{pc} \Rightarrow \mathsf{v_i}$.

(H7) $\mu_i(L) = \{"n_0","n_1",...,"n_n"\}$.

By the hypothesis of Theorem 3, we have:

(H8) $\Gamma \vdash \mathsf{sbroadcast(L,e||pc,K)} : \tau'\ \mathsf{cmd}$.

By (H8) and the typing rule SBroadcast, we have:

(H9) $\Gamma \vdash e : \tau'$.

(H10) $\Gamma \vdash L : \tau'\ Lvar$.

(H11) $\Gamma \vdash K : \top\ Kvar$.

(H12) $\Gamma \vdash \mathsf{pc} : \bot\ \mathsf{Cvar}$.

To prove (G_1), we rely on the Definition 24 that states that $\mu_0 =_\tau^\Gamma \mu_1$, if $\forall y \in \mu_0$ such that $\Gamma(y) = \tau'\ var$ and $\tau' \leq \tau$ then $\mu_0(y) = \mu_1(y)$. For ($H_9 - H_{12}$), all the variables types are different than $\tau'\ var$. Since we are only interested in variables of type $\tau'\ var$, (G_1) follows by (H_1) and (H_2).

To prove (G_2), we have two cases:

(H_{13}) $\Gamma(L) \leq \tau'$

Being $\mu_0' =_\tau^\Gamma \mu_1'$ and by (H_2) and(H_3) we have that $t_0 = t_1$. For an execution that starts with μ_0, $t_0 = \mathsf{BEnc(L,vk_0,"m"||"v")}$ and for an execution that starts with μ_1, $t_1 = \mathsf{BEnc(L,vk_1,"m"||"v")}$. By ($H_{13}$), $\Gamma(L) \leq \tau'$ then $\mathsf{filter(t_0) = [L,m]}$ and $\mathsf{filter(t_1) = [L,m]}$. It results that $\mathsf{filter(t_0) = filter(t_1)}$ and therefore we prove (G_2).

Since we proved (G_1) and (G_2), then p is NI_τ^Γ.

(H_{14}) $\Gamma(L) \not\leq \tau'$

Being $\mu_0' =_\tau^\Gamma \mu_1'$ and by (H_2) and(H_3) we have that $t_0 = t_1$. For an execution that starts with μ_0, $t_0 = \mathsf{BEnc(L,vk_0,"m"||"v")}$ and for an execution that starts with μ_1, $t_1 = \mathsf{BEnc(L,vk_1,"m"||"v")}$. By ($H_{14}$), $\Gamma(L) \not\leq \tau'$, then $\mathsf{filter_\tau(t_0) = [\,]}$ and $\mathsf{filter_\tau(t_1) = [\,]}$. It results that $\mathsf{filter_\tau(t_0) = filter_\tau(t_1)}$ and therefore we prove (G_2).

Since we proved (G_1) and (G_2), then p is NI_τ^Γ.

Subcase 1.4. $p \stackrel{\Delta}{=} \mathsf{for\ n} \in \mathsf{L\ endorse\ Ra(L, L', n)}$.

Concerning (G_1) and (G_2), by Definition 26, we have that:

(H_1) $\mu_0 =_\tau^\Gamma \mu_1$.

(H_2) $\mu_i \vdash \mathsf{for\ n} \in \mathsf{L\ endorse\ Ra(L, L', n)} \Rightarrow^{t_i} \mu_i'$.

By the semantics rule Endorse-Ra we have that:

(H_3) $t_i = [\,]$.

(H_4) $\mu_i \vdash \mathsf{Ra(L)} \Rightarrow \mathsf{S}$.

(H_5) $\mu_i(L) = \{"n_0", "n_1", ..., "n_n"\}$.

(H_6) $\mu_i(L') = \{"n_0'", "n_1'", ..., "n_n'"\}$.

By (H_2) and the semantics rule Endorse-Ra we have that:

(H_7) $\mu_i' = \mu_i[L := L \setminus S; L' := L' \cup S]$.

By the hypothesis of Theorem 3, we have:

(H_8) $\Gamma \vdash \mathsf{for\ n} \in \mathsf{L\ endorse\ Ra(L, L', n)} : \tau'\ \mathsf{cmd}$.

By (H_8) and the typing rule Remote Attestation, we have that:

(H_9) $\Gamma \vdash L : \tau'\ Lvar$.

(H_{10}) $\Gamma \vdash L' : \tau''\ Lvar$.

(H_{11}) $\Gamma \vdash K : \top\ Kvar$.

(H_{12}) $\Gamma \vdash \mathsf{pc} : \bot\ \mathsf{Cvar}$.

To prove (G_1), we rely on the Definition 24 that states that $\mu_0 =_\tau^\Gamma \mu_1$, if $\forall y \in \mu_0$ such that $\Gamma(y) = \tau'\ var$ and $\tau' \leq \tau$ then $\mu_0(y) = \mu_1(y)$. For ($H_9 - H_{12}$), all the variables types are different than $\tau'\ var$. Since we are only interested in variables of type $\tau'\ var$, (G_1) follows by (H_1) and (H_2).

To prove (G_2) we rely on the Definition 27 and (H_3). Since $t_0 = t_1 = [\,]$ then $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

Since we proved (G_1) and (G_2), then p is NI^Γ_τ.

Case 2 (Case: height $\leq n$)**.** *We will state our inductive hypothesis for a program c:*

For the hypothesis of the theorem, we have that:

(IH_1) $\Gamma \vdash c : \tau'\ cmd$.

(IH_2) $\mu_0 =^\Gamma_\tau \mu_1$.

For the derivation tree of height $\leq n$, we have that:

(IH_3) $\mu_0 \vdash c \Rightarrow^{t_0} \mu'_0$.

(IH_4) $\mu_1 \vdash c \Rightarrow^{t_1} \mu'_1$.

Then we conclude that:

(IH_5) $\mu'_0 =^\Gamma_\tau \mu'_1$.

(IH_6) $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

We suppose that ($IH_1 - IH_6$) are valid for programs of height $\leq n$ of typing derivation tree $\Gamma \vdash p$.

Case 3. *Inductive case: height$= n+1$*

Subcase 3.1. $p \stackrel{\Delta}{=} c'; c''$.

We want to prove that $\Gamma \vdash c'; c''$ is NI^Γ_τ. We assume that c' and c'' are of height $\leq n$, and $c'; c''$ of height $n+1$.

For the typing rule Sequence, we have that:

(H_1) $\Gamma \vdash c' : \tau'\ cmd$.

(H_2) $\Gamma \vdash c'' : \tau'\ cmd$.

For the semantics rule Sequence, we have that:

(H3) $\mu_i \vdash c' \Rightarrow^{t'_i} \mu'_i$.

(H4) $\mu'_i \vdash c'' \Rightarrow^{t''_i} \mu''_i$.

Concerning c', from the inductive hypotheses, it follows that:

(H5) $\mu_{c'_0} =^{\Gamma}_{\tau} \mu_{c'_1}$.

(H6) $\mu_{c'_0} \vdash c \Rightarrow^{t'_0} \mu'_{c'_0}$

(H7) $\mu_{c'_1} \vdash c \Rightarrow^{t'_1} \mu'_{c'_1}$

(H8) $\mu'_{c'_0} =^{\Gamma}_{\tau} \mu'_{c'_1}$.

(H9) $\mathsf{filter}_\tau(\mathsf{t}'_0) = \mathsf{filter}_\tau(\mathsf{t}'_1)$.

It follows that c' is NI^{Γ}_{τ}.

Concerning c'', from the inductive hypotheses, it follows that:

(H10) $\mu_{c''_0} =^{\Gamma}_{\tau} \mu_{c''_1}$.

(H11) $\mu_{c''_0} \vdash c \Rightarrow^{t''_0} \mu'_{c''_0}$

(H12) $\mu_{c''_1} \vdash c \Rightarrow^{t''_1} \mu'_{c''_1}$

(H13) $\mu'_{c''_0} =^{\Gamma}_{\tau} \mu'_{c''_1}$.

(H14) $\mathsf{filter}_\tau(\mathsf{t}''_0) = \mathsf{filter}_\tau(\mathsf{t}''_1)$.

It follows that c'' is NI^{Γ}_{τ}.

In the semantics rule Sequence, we use $t' \cdot t''$ to denote the concatenation of two traces t' and t''. By (H_9) and (H_{14}) we have that $t'_0 = t'_1$ and $t''_0 = t''_1$, which implies that $t'_0 \cdot t''_0 = t'_1 \cdot t''_1$. We conclude that $\mathsf{filter}_\tau(\mathsf{t}'_0 \cdot \mathsf{t}''_0) = \mathsf{filter}_\tau(\mathsf{t}'_1 \cdot \mathsf{t}''_1)$.

Since c', c'' are NI^{Γ}_{τ} and $\mathsf{filter}_\tau(\mathsf{t}'_0 \cdot \mathsf{t}''_0) = \mathsf{filter}_\tau(\mathsf{t}'_1 \cdot \mathsf{t}''_1)$*, we conclude that $\Gamma \vdash c'; c''$ is NI^{Γ}_{τ}.*

Subcase 3.2. $p \stackrel{\Delta}{=} \mathsf{while\ e\ do\ c}$.

For the typing rule While, we have that:

(H₁) $\Gamma \vdash c : \tau'\ cmd$.

(H₂) $\Gamma \vdash e : \tau'$.

In the following, we show by induction that c is NI_τ^Γ.

By (H_1) *and because the height of* $\Gamma \vdash$ while e do c *is* $n+1$, *we know that the height of the typing derivation tree of* (H_1) *is* $\leq n$. *Hence, we can apply the inductive hypothesis and get:*

(H₃) $\mu_0 =_\tau^\Gamma \mu_1$.

(H₄) $\mu_0 \vdash c \Rightarrow^{t_0} \mu_0'$.

(H₅) $\mu_1 \vdash c' \Rightarrow^{t_1} \mu_1'$.

Then,

(H₆) $\mu_0' =_\tau^\Gamma \mu_1'$.

(H₇) $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

We want to prove that while e do c *is* NI_τ^Γ. *By the hypothesis of the theorem, we have that if:*

(H₀′) $\Gamma \vdash$ while e do c $: \tau'$ cmd.

(H₁′) $\mu_0 =_\tau^\Gamma \mu_1$.

(H₂′) $\mu_0 \vdash$ while e do c $\Rightarrow^{\mathsf{t}_0} \mu_0'$.

(H₃′) $\mu_1 \vdash$ while e do c $\Rightarrow^{\mathsf{t}_1} \mu_1'$.

Then, we want to prove that:

(G₀) $\mu_0' =_\tau^\Gamma \mu_1'$.

(G₁) $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

By (H_2') *and the semantics rule of Loop, we have that:*

(H'4) $\mu_0 \vdash e \Rightarrow v_0$.

By (H'3) and the semantics rule of Loop, we have that:

(H'5) $\mu_1 \vdash e \Rightarrow v_1$.

Depending on τ', we have two cases:

Subcase 3.2.1. $\Gamma \vdash e : \tau',\ \tau' \leq \tau$.

(H'6) $\tau' \leq \tau$.

In the case of (H'6), we check if we can apply the Subtype rule on While command. We would like to check if the command While of type τ' cmd can be typable as τ'' cmd, where $\tau'' \not\leq \tau'$. If the Subtype rule can be applied, the the proof of this case is analogous to Subcase 3.2.2.

Otherwise, if the Subtype rule cannot be applied, then by Lemma 1 that can be applied on (H'1), (H2), (H'6), (H'4) and (H'5), we conclude that $v_0 = v_1$.

We prove this case by induction on the height of the semantics tree of (H'2). (We do not show this formally, but we rely on the fact that the height of (H'2) is equal to the height of (H'3) by Lemma 1).

Base case: height = 2. *The only possibility for the semantics tree to be of height = 2 is:*

- $v_0 = v_1 = False$.

$$\frac{\mu_0 \vdash e \Rightarrow False}{\mu_0 \vdash \text{while e do c} \Rightarrow^{t_0} \mu_0}$$

$$\frac{\mu_1 \vdash e \Rightarrow False}{\mu_1 \vdash \text{while e do c} \Rightarrow^{t_1} \mu_1}$$

We have that $\mu'_0 =^{\Gamma}_{\tau} \mu_0$ *and* $\mu'_1 =^{\Gamma}_{\tau} \mu_1$ *. We conclude by* (H'_1) *that* $\mu'_0 =^{\Gamma}_{\tau} \mu'_1$*. Moreover, since* $t_0 = t_1 = [\,]$ *then* $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$*.*

Assuming that our inductive hypothesis holds for the case of While when evaluating the height of semantics tree $\leq m$ *with* $\Gamma \vdash e : \tau', \tau' \leq \tau$*, let us prove the case of While with height* $= m + 1$*.*

Inductive case: height = m+1.

$$\frac{\mu_0 \vdash e \Rightarrow True \qquad \mu_0 \vdash c \Rightarrow^{t_0} \mu''_0 \qquad (H'_7)\ \mu''_0 \vdash \mathsf{while\ e\ do\ c} \Rightarrow^{\mathsf{t}'_0} \mu'_0}{\mu_0 \vdash \mathsf{while\ e\ do\ c} \Rightarrow^{\mathsf{t}_0 \cdot \mathsf{t}'_0} \mu'_0}$$

The height of (H'_2) *is* $m + 1$*.*

$$\frac{\mu_1 \vdash e \Rightarrow True \qquad \mu_1 \vdash c \Rightarrow^{t_1} \mu''_1 \qquad (H'_8)\ \mu''_1 \vdash \mathsf{while\ e\ do\ c} \Rightarrow^{\mathsf{t}'_1} \mu'_1}{\mu_1 \vdash \mathsf{while\ e\ do\ c} \Rightarrow^{\mathsf{t}_1 \cdot \mathsf{t}'_1} \mu'_1}$$

The height of (H'_3) *is* $m + 1$*.*

By the previous induction on c, we have that $\mu''_0 =^{\Gamma}_{\tau} \mu''_1$ *and* $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$*, through* $\mu_0 =^{\Gamma}_{\tau} \mu_1$*,* $\mu_0 \vdash c' \Rightarrow^{t_0} \mu''_0$ *and* $\mu_1 \vdash c' \Rightarrow^{t_1} \mu''_1$*.*

Moreover, by induction on $\mathsf{while\ e\ do\ c}$ *by* (H'_7) *and* (H'_8)*, we can conclude that* $\mu'_0 =^{\Gamma}_{\tau} \mu'_1$ *which is already our goal* (G_1) *and* $\mathsf{filter}_\tau(\mathsf{t}'_0) = \mathsf{filter}_\tau(\mathsf{t}'_1)$*. This is because we have that* $\mu''_0 =^{\Gamma}_{\tau} \mu''_1$ *and* $\mu''_0 \vdash \mathsf{while\ e\ do\ c} \Rightarrow^{\mathsf{t}'_0} \mu'_0$ *and* $\mu''_1 \vdash \mathsf{while\ e\ do\ c} \Rightarrow^{\mathsf{t}'_1} \mu'_1$*.*

Since $t_i \cdot t'_i$ *is the concatenation of two traces* t_i *and* t'_i*, and since* $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$ *and* $\mathsf{filter}_\tau(\mathsf{t}'_0) = \mathsf{filter}_\tau(\mathsf{t}'_1)$*, we conclude that* $\mathsf{filter}_\tau(\mathsf{t}_0 \cdot \mathsf{t}'_0) = \mathsf{filter}_\tau(\mathsf{t}_1 \cdot \mathsf{t}'_1)$ *from which* (G_2) *follows.*

Since (G_1) *and* (G_2) *are satisfied, then* $\mathsf{while\ e\ do\ c}$ *is* NI^{Γ}_{τ}*.*

Subcase 3.2.2. $\Gamma \vdash e : \tau',\ \tau' \not\leq \tau$*.*

(H'_9) $\tau' \not\leq \tau$*.*

By Lemma 2 that can be applied on (H_1)*,* (H_4)*,* (H_5) *and* (H'_9)*, we conclude that* $\mu_i =^{\Gamma}_{\tau} \mu'_i$*.*

We prove that $\mu'_0 =^{\Gamma}_{\tau} \mu'_1$ *by applying the transitive property on* $=^{\Gamma}_{\tau}$*:*

- *By* (H'_1) *we have that* (1) $\mu_0 =^{\Gamma}_{\tau} \mu_1$.
- *By Lemma 2 we have that* (2) $\mu_0 =^{\Gamma}_{\tau} \mu'_0$ *and* (3) $\mu_1 =^{\Gamma}_{\tau} \mu'_1$.
- *By* (1) *and* (2) *we have that* (4) $\mu_1 =^{\Gamma}_{\tau} \mu'_0$.
- *By* (1) *and* (3) *we have that* (5) $\mu_0 =^{\Gamma}_{\tau} \mu'_1$.
- *By* (4) *and* (5) *we conclude that* (6) $\mu'_0 =^{\Gamma}_{\tau} \mu'_1$.

By applying structural induction, we have that $\mathsf{filter}_\tau(\mathsf{t}'_0) = \mathsf{filter}_\tau(\mathsf{t}'_1)$*. By previous induction on c we have that* $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$*. Since* $t_i \cdot t'_i$ *is the concatenation of two traces* t_i *and* t'_i*, and since* $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$ *and* $\mathsf{filter}_\tau(\mathsf{t}'_0) = \mathsf{filter}_\tau(\mathsf{t}'_1)$*, we conclude that* $\mathsf{filter}_\tau(\mathsf{t}_0 \cdot \mathsf{t}'_0) = \mathsf{filter}_\tau(\mathsf{t}_1 \cdot \mathsf{t}'_1)$ *from which* (G_2) *follows.*

Since (G_1) *and* (G_2) *are satisfied, then* while e do c′ *is* NI^{Γ}_{τ}.

Subcase 3.2.3. $p \stackrel{\Delta}{=} \mathsf{if\ e\ then\ c'\ else\ c''}$.

For the typing rule If, we have that:

(H_1) $\Gamma \vdash e : \tau'$.

(H_2) $\Gamma \vdash c' : \tau'\ cmd$.

(H_3) $\Gamma \vdash c'' : \tau'\ cmd$.

For the semantics rule Branch, we have that:

(H_4) $\mu_i \vdash e \Rightarrow v_i$.

(H_5) $\mu_i \vdash c' \Rightarrow_{t_i} \mu'_i$.

(H_6) $\mu_i \vdash c'' \Rightarrow_{t_i} \mu'_i$.

Concerning c′, by induction, we conclude that:

(H_7) $\mu'_{c'_0} =^{\Gamma}_{\tau} \mu'_{c'_1}$.

(H_8) $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

It follows that c' *is* NI^{Γ}_{τ}.

Concerning c''*, by induction, we conclude that:*

(H_8) $\mu'_{c''_0} =^{\Gamma}_{\tau} \mu'_{c''_1}$.

(H_9) $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

It follows that c'' *is* NI^{Γ}_{τ}.

Depending on τ'*, we have two cases:*

Subcase 3.2.4. $\Gamma \vdash e : \tau', \tau' \leq \tau$.

(H_{10}) $\tau' \leq \tau$.

In the case of (H_{10})*, we check if we can apply the Subtype rule on If command. We would like to check if the command If of type* τ' *cmd can be typable as* τ'' *cmd, where* $\tau'' \not\leq \tau'$*. If the Subtype rule can be applied, the the proof of this case is analogous to Subcase 3.2.5.*

Otherwise, if the Subtype rule cannot be applied, then by Lemma 1, we can conclude that:

(H_{11}) $v_0 = v_1$.

$$\frac{\mu_0 \vdash e \Rightarrow True \qquad \mu_0 \vdash c' \Rightarrow^{t_0} \mu'_0}{\mu_0 \vdash \quad If\ e\ \mathsf{then} c'\ \mathsf{else}\ c'' \Rightarrow^{\mathsf{t}_0} \mu'_0}$$

$$\frac{\mu_1 \vdash e \Rightarrow True \qquad \mu_1 \vdash c' \Rightarrow^{t_1} \mu'_1}{\mu_1 \vdash \quad If\ e\ \mathsf{then} c'\ \mathsf{else}\ c'' \Rightarrow^{\mathsf{t}_1} \mu'_1}$$

We have that $\mu_0' =_\tau^\Gamma \mu_0$ *and* $\mu_1' =_\tau^\Gamma \mu_1$. *Since* $\mu_0 =_\tau^\Gamma \mu_1$, *we conclude that* $\mu_0' =_\tau^\Gamma \mu_1'$. *Since* $t_0 = t_1 = [\,]$, *then* $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

$$\frac{\mu_0 \vdash e \Rightarrow False \qquad \mu_0 \vdash c'' \Rightarrow^{t_0} \mu_0'}{\mu_0 \vdash \quad If\ e\ \mathsf{thenc'}\ \mathsf{else}\ \mathsf{c''} \Rightarrow^{\mathsf{t}_0} \mu_0'}$$

$$\frac{\mu_1 \vdash e \Rightarrow false \qquad \mu_1 \vdash c'' \Rightarrow^{t_1} \mu_1'}{\mu_1 \vdash \quad If\ e\ \mathsf{thenc'}\ \mathsf{else}\ \mathsf{c''} \Rightarrow^{\mathsf{t}_1} \mu_1'}$$

We have that $\mu_0' =_\tau^\Gamma \mu_0$ *and* $\mu_1' =_\tau^\Gamma \mu_1$. *Since* $\mu_0 =_\tau^\Gamma \mu_1$, *we conclude that* $\mu_0' =_\tau^\Gamma \mu_1'$. *Since* $t_0 = t_1 = [\,]$, *then* $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$.

Subcase 3.2.5. $\Gamma \vdash e : \tau', \tau' \not\leq \tau$.

(H_{12}) $\tau' \not\leq \tau$.

By Lemma 2, we conclude that $\mu_i =_\tau^\Gamma \mu_i'$.
We prove that $\mu_0' =_\tau^\Gamma \mu_1'$ *by applying the transitive property on* $=_\tau^\Gamma$.

- *We have that (1)* $\mu_0' =_\tau^\Gamma \mu_0$.
- *We have that (2)* $\mu_1' =_\tau^\Gamma \mu_1$.
- *We have that (3)* $\mu_0 =_\tau^\Gamma \mu_1$.
- *By (1) and (3) we have that (4)* $\mu_1 =_\tau^\Gamma \mu_0'$.
- *By (2) and (3) we have that (5)* $\mu_0 =_\tau^\Gamma \mu_1'$.
- *By (4) and (5) we conclude that (6)* $\mu_0' =_\tau^\Gamma \mu_1'$.

Since $\mathsf{filter}_\tau(\mathsf{t}_0) = \mathsf{filter}_\tau(\mathsf{t}_1)$ *and* $\mu_0' =_\tau^\Gamma \mu_1'$, *then* p *is* NI_τ^Γ.

■

APPENDIX C

IMPLEMENTATIONS OF THE SCHEMES IN CHAPTER 4

In this appendix, we provide the Ocaml implementations of the three schemes compared in Chapter 4. We only provide the source code of the key generation, the encryption and the decryption. The full source code can found here.

C.1 Implementation of ElGamal Baseline 4.1.2

```
let keygen n gr =
  let pbits = gr.pbits in
  let p = gr.p in
  let q = q gr.p in
  let x_list = List.init n (fun _ -> sample_le (pbits - 1) q) in
  let y_list = List.map (fun x -> Z.powm gr.g x p) x_list in
  { skey = { group = gr; key = x_list };
    pkey = { group = gr; key = y_list };
  }
```

Listing C.1: Key generation algorithm

```
let safe_encrypt pkey m =
  let gr = pkey.group in
  let r = sample_le (gr.pbits − 1) (q gr.p) in
  let u = Z.powm gr.g r gr.p in
  let encodem = Z.powm (Z.succ m) (Z.of_int 2) gr.p in
  let v = List.map (fun y −> mulm gr (Z.powm y r gr.p) encodem) pkey.key in
  (u,v)
```

Listing C.2: Encryption algorithm

```
let decrypt skey x (u,v) =
  let gr = skey.group in
  let denom = Z.powm u (Z.neg x) gr.p in
  let v_0 = List.nth v 0 in
  let decmsg= mulm gr v_0 denom in
  decmsg


let decode gr m =
  let p = gr.p in
  let q = q gr.p in
  let r = Z.powm m (Z.shift_right (Z.succ q) 1) p in
  let m = if Z.leq r q then r else (Z.sub p r) in
  (Z.pred m)
```

Listing C.3: Decrypting and decoding algorithm

```
let safe_decrypt skey x_0 (u,v) = decode skey.group ( decrypt skey x_0 (u,v))
```

Listing C.4: Safe decrypting algorithm

C.2 Implementation of Boneh-Franklin 4.1.3

```
let keygen n gr =
  let pbits = gr.pbits in
  let p = gr.p in
  let q = q gr.p in
  let g = gr.g in
  let alfa_list = List.init n (fun _ -> sample_le (pbits - 1) q) in
  let r_list = List.init n ( fun _ -> sample_le (pbits - 1) q) in
  let h_list = List.map (fun r -> Z.powm g r p) r_list in
  let y = List.fold_left2 (fun acc h alfa -> mulm gr acc (Z.powm h alfa p)) Z.one
      h_list alfa_list in
  let alfa0 = List.nth alfa_list 0 in
  let inv_alfa0 = Z.powm alfa0 (Z.pred (Z.pred q) ) q in
  let gamma0_list = List.map (fun alfa -> mulmq q alfa inv_alfa0) alfa_list in
  { skey = { group = gr; key = alfa_list };
    pkey = { group = gr; key = y };
    hkey = { group = gr; key = h_list};
    gkey = {group = gr; key = gamma0_list};
  }
```

Listing C.5: Key generation algorithm

```
let safe_encrypt (pk : pkey) h_list m =
  let gr = pk.group in
  let r = sample_le (gr.pbits - 1) (q gr.p) in
  let encodem = Z.powm (Z.succ m) (Z.of_int 2) gr.p in
  ((List.map (fun h -> Z.powm h r gr.p) h_list), mulm gr (Z.powm pk.key r gr.p)
      encodem)
```

Listing C.6: Encryption algorithm

```
let decrypt gkey alfa (u,v) =
  let gr = gkey.group in
  let bigu = List.fold_left2 (fun acc hri gammai -> mulm gr acc (Z.powm hri
      gammai gr.p)) Z.one u gkey.key in
  let bigualfa = Z.powm bigu (Z.neg alfa) gr.p in
  let decmsg= mulm gr v bigualfa in
  decmsg

let decode gr m =
  let p = gr.p in
  let q = q gr.p in
  let r = Z.powm m (Z.shift_right (Z.succ q) 1) p in
  let m = if Z.leq r q then r else (Z.sub p r) in
  (Z.pred m)
```

Listing C.7: Decrypting and decoding algorithm

```
let safe_decrypt gamma0_list alfa_0 (u,v) = decode gamma0_list.group (decrypt
    gamma0_list alfa_0 (u,v))
```

Listing C.8: Safe decrypting algorithm

C.3 Implementation of the proposed scheme 4.1.4

```
let keygen list_coprime grnn =
  let nbits = grnn.nbits in
  let nn = grnn.nn in
  let h = sample1_le (nbits - 1) (Z.sub nn Z.one) in
  let y_list = List.map (fun x -> Z.powm h x nn) list_coprime in
```

```
  let h0 = List.nth y_list 0 in
  let n0 = List.nth list_coprime 0 in
  { skey1 = { group1 = grnn; key1 = y_list};
    pkey1 = { group1 = grnn; key1 = list_coprime};
  }, h0, n0, h
```

Listing C.9: Key generation algorithm

```
let safe_encrypt gr h pkey1 m =
  let grnn = pkey1.group1 in
  let r = sample_le (gr.pbits − 1) (q gr.p) in
  let z = List. fold_left (fun acc x −> mulmn grnn.lam acc x ) Z.one pkey1.key1 in
  let x = Z.powm h z grnn.nn in
  let u = Z.powm gr.g r gr.p in
  let y = Z.powm gr.g x gr.p in
  let encodem = Z.powm (Z.succ m) (Z.of_int 2) gr.p in
  let v = mulm gr (Z.powm y r gr.p) encodem in
  (z,u,v)
```

Listing C.10: Encryption algorithm

```
let decrypt gr grnn n0 h0 (z,u,v) =
  let inv_n0 = Z.powm n0 (Z.neg Z.one ) grnn.lam in
  let n = mulmn grnn.lam z inv_n0 in
  let deckey = Z.powm h0 n grnn.nn in
  let ukey = Z.powm u (deckey) gr.p in
  let invukey = Z.powm ukey (Z.neg Z.one) gr.p in
  let decmsg = mulm gr v invukey in
  decmsg


let decode gr m =
```

```
let  p = gr.p in
let  q = q gr.p in
let  r  = Z.powm m (Z.shift_right (Z.succ q) 1) p in
let  m = if Z.leq  r  q then r else  (Z.sub p r)  in
(Z.pred m)
```

Listing C.11: Decrypting and decoding algorithm

```
let  safe_decrypt gr grnn n0 h0 (z,u,v) =  decode gr (  decrypt  gr  grnn n0 h0 (z,u,v))
```

Listing C.12: Safe decrypting algorithm