Tracking Information Flow by Mapping Broadcast Encryption Subgroups to Security Lattices

Mohamad El Laz, Alejandro Hevia, Tamara Rezk

Tracking Information Flow by Mapping Broadcast Encryption Subgroups to Security Lattices

El Laz Mohamad[1], Hevia Alejandro[2], and Rezk Tamara[1]

[1] Inria, Sophia-Antipolis, France
[2] University of Chile, Santiago, Chile

Abstract. In this paper 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. 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.

Keywords: Secure information flow · Broadcast encryption schemes · Static analysis

1 Introduction

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 [22], to prevent leakage of information to attackers with polynomial computational power. However, even with plain encryption, the confidentiality of keys, plaintexts, and ciphertexts are interdependent [32]: encryption with untrusted keys is clearly dangerous, and plaintexts should never be more secret than their decryption keys.

We consider general scenarios where a server wishes to broadcast messages to nodes that have different confidentiality clearances. Broadcasted messages should only be visible to nodes with the appropriate confidentiality clearance.

Broadcast encryption schemes [7, 18, 26] offer an efficient solution to protect confidentiality over a network. These schemes allow a server to securely broadcast a message to a subgroup of nodes. They provide cryptographic primitives to generate a common cryptographic key for a subgroup of nodes to ensure that only members of a privileged subgroup can decrypt a message.

Yet, even by using IND-CPA broadcast encryption schemes (BES), server programs might encrypt a message with the wrong keys, thus, sending it to nodes without the appropriate clearance or send a plaintext which contains more confidential information than the one desired.

Our proposal is mapping broadcast subgroups of nodes to levels in information flow security lattices [14] to verify secure information flow in the server code. We implement this idea via a type system. Our type system checks that messages intended for a specific subgroup of nodes are not leaked to nodes in security levels with less privileges. It also checks that important variables used in the server such as cryptographic keys variables and variables that record nodes according to their security class are not erroneously or maliciously modified. We formally prove that our type system complies with a new security property based on the information flow literature [33].

In summary, **our contributions** are:

- By mapping BES subgroups to security classes, we show how to control the flow of information from the server to nodes in different classes.
- A type system for server code featuring a cryptographic operation of broadcast encryption and an algorithm for generation of cryptographic keys to broadcast to group of nodes with different security clearances.
- A soundness proof with respect to a new information flow security property.

2 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 [31] 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.

3 BES and Security Classes

3.1 Broadcast Encryption Schemes

Broadcast encryption schemes [18] allow a sender (e.g. a server) to transmit a message to a privileged group of receivers (e.g. nodes) such that only the receivers in the privileged group can decrypt it. We first a general definition of broadcast encryption scheme, then we present an instantiation based on ElGamal encryption.

Definition 1. *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*, heasers* $H_1,\ldots,H_n$ *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.*

We denote as $\mathsf{BEnc}(\mathsf{L},\mathsf{K_L},\mathsf{m})$ *the primitive that broadcasts an encrypted message* $Encrypt(L,K_L,m)$ *to the network.*

Broadcast encryption schemes can offer strong security guarantees, such as *IND-CPA*, while adopting small key sizes [7, 26].

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 2 (*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.*

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

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 where $b \in \{0,1\}$ 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.

Example 1 (Boneh-Franklin Scheme).

In this section we provide an example of a broadcast encryption scheme. We choose to use the scheme that was proposed by A. Boneh and M. K. Franklin [6]; which is cryptographically equivalent to *ElGamal*. Note that any other broadcast encryption scheme may be used for the purpose of our work.

- 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 $\mathbb{Z}_1$. 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 in G_q, 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: $({h_1}^r, ..., {h_k}^r, m \cdot y^r) = (H_1, ..., H_k, C)$.
- Decryption. To decrypt, a node n_i computes m as:

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

3.2 Security Classes

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 [14]. 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 $(\mathcal{L}, \leq)$. 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 $\mathcal{L}$.

Example 2. 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 1.

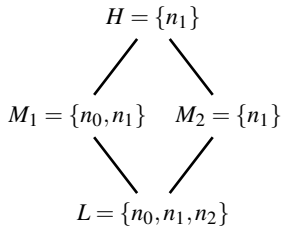

Fig. 1: Security lattice

Example 3. We present a numerical example where a server is willing to broadcast a message m of security clearance M_1 to a targeted subset of nodes (n_0, n_1). We assume to have the confidentiality lattice in Figure 1 and the broadcast encryption scheme presented in Example 1. We show how node n_2 (that belongs to a lower security level L) cannot decrypt the message since its security level is lower than the security clearance level of the message.
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 $\mathbb{Z}_{11}^*$ [16].
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.

Algorithm 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 $r_i \in \mathbb{Z}_5$ 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 $(\frac{\alpha_0}{\alpha_1}, \frac{\alpha_1}{\alpha_1}) = (2,1)$

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 : $({h_0}^r, {h_1}^r, m \cdot y^r) = (H_0, H_1, C)$.

Algorithm 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 he does not possess its decryption key.

Algorithm 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 $U = H_0^{\gamma_0^{(0)}} \cdot H_1^{\gamma_1^{(0)}} = 4$.

4 Type System

In this section, we present a language with syntax and a formal semantics for server code featuring broadcast encryption primitives. We also define a type system and a security property for secure information flow and state a theorem that says that typable server programs are guaranteed to provide secure information flow in the system.

4.1 Syntax

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

$$
\begin{aligned}
(\textit{Programs}) \ p &::= e \mid c \\
(\textit{Expressions}) \ e &::= x \mid n \mid v \mid k \mid e \circ e' \\
(\textit{Commands}) \ c &::= e := e' \mid c; c' \mid k := \mathsf{KG}(\{\mathsf{n_1}, ..., \mathsf{n_i}\}, \mathsf{k'}) \\
&\quad \mid \mathsf{while\ e\ do\ c} \mid \mathsf{if\ e\ then\ c\ else\ c'} \\
&\quad \mid \mathsf{sbroadcast}(\{\mathsf{n_1}, ..., \mathsf{n_j}\}, \mathsf{k}, \mathsf{e})
\end{aligned}
$$

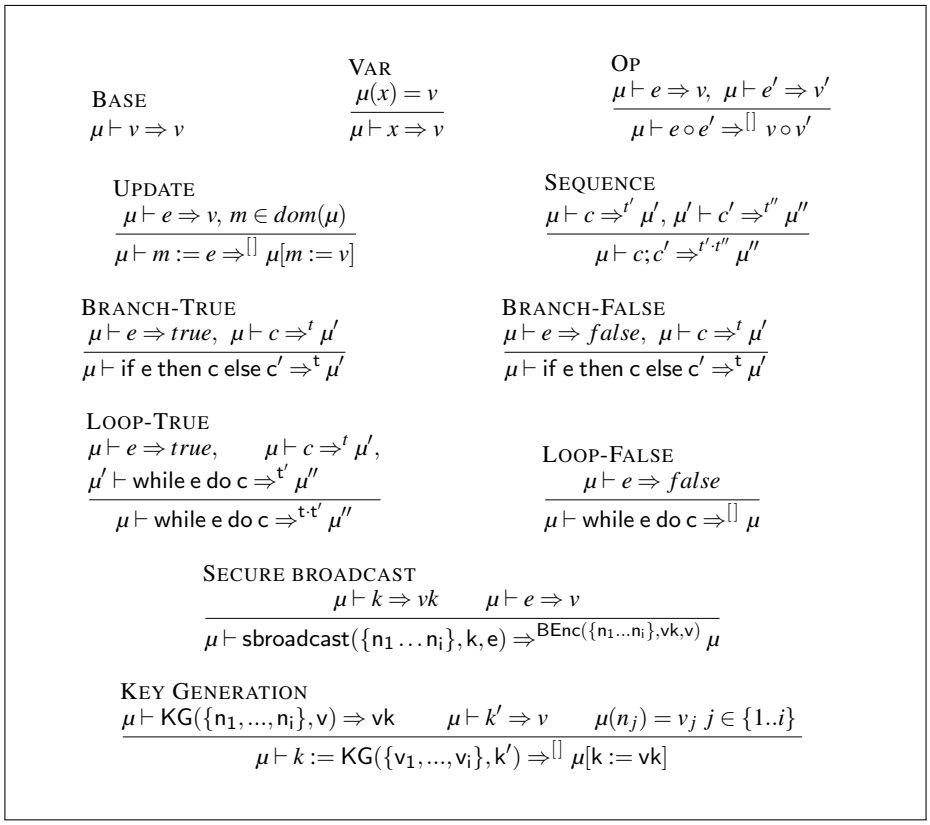

Fig. 2: Semantics

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 $\mathsf{sbroadcast}(\{\mathsf{n_1}, ..., \mathsf{n_j}\}, \mathsf{k}, \mathsf{e})$, 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 $k := \mathsf{KG}(\{\mathsf{n_1}, ..., \mathsf{n_i}\}, \mathsf{k}')$, the server uses a master key k' to generate the encryption key k for a set of nodes $\{n_1, ..., n_i\}$.

4.2 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 2. We only highlight the non-standard rules Secure broadcast and Key generation. 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.

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 $\mathsf{BEnc}(\{\mathsf{n}_1,...,\mathsf{n}_i\},\mathsf{vk},"\mathsf{m}")$.

4.3 Typing Rules

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

$$\begin{aligned}(\textit{Programs types})\ \rho\ ::=\ & \tau \mid \tau\ \textit{var} \mid \tau\ \textit{cmd} \mid \tau\ \textit{Nvar}(n) \\ & \mid \tau\ \textit{Kvar}\ (n_1,\ldots,n_i) \mid \top\ \textit{MKvar}\end{aligned}$$

Type τ *var* is the type of a variable, τ *cmd* is the type of a command, and type $\top$ *MKvar* the type of master keys. Type τ *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 τ *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 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.

Our typing system includes standard rules (*Var, Assign, Sequence, If, While*) for secure information flow control [36] and special rules (*Nvar, MKvar, Kvar-Assign, SBroadcast*).

The *Var* rule binds the type τ *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 τ *Nvar*(n_i) and e (the message to broadcast) of type τ. Moreover, it binds the key variable k to be of type τ *Kvar*$(n_1 \ldots n_j)$. The *Kvar-Assign* rule binds each node n_j to be of type τ *Nvar*(n_j), k to be of type τ *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 τ *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 3. The *Base* rule states that τ is a subset of τ' if $\tau \leq \tau'$. The *Cmd* rule states that τ' *cmd* is a subset of τ *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 τ *var* can be bound to type τ, and the *S-NVar* rule

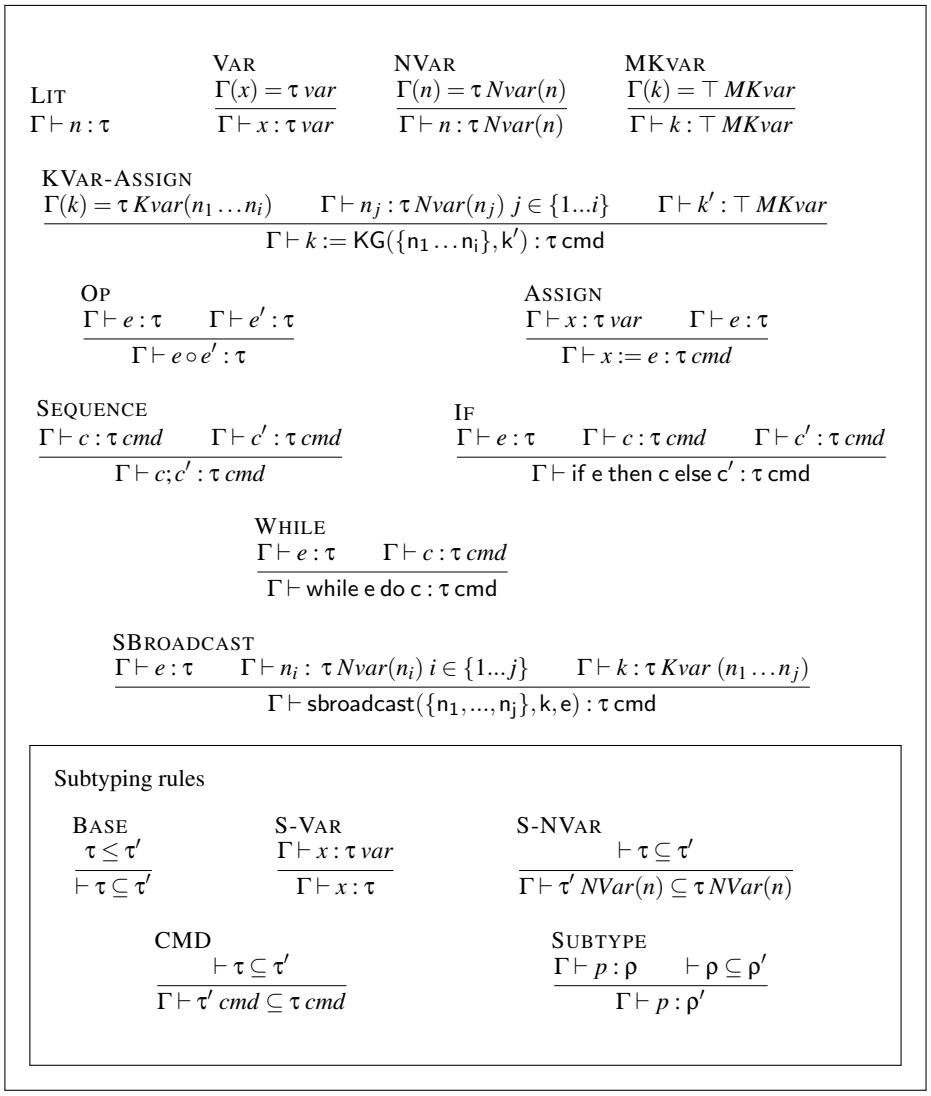

Fig. 3: Typing rules

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 1, where $L \leq M_1 \leq H$ and $L \leq M_2 \leq H$.

Example 4. We consider a server willing to send a message m: $\Gamma(m) = M_1\ var$ to a set of nodes in $\{n_0, n_1\}$: $\Gamma(n_0) = M_1\ NVar(n_0)$ 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:

$$\boxed{k := \mathsf{KG}(\{\mathsf{n_0}, \mathsf{n_1}\}, \mathsf{m}, \mathsf{k'}); \mathsf{sbroadcast}(\{\mathsf{n_0}, \mathsf{n_1}\}, \mathsf{k}, \mathsf{m})}$$

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 $\Gamma(n_1) = M_1\ NVar(n_1)$.

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

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 $M_1\ KVar(n_0, n_1)$.

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

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

Example 5. In contrast with Example 4, 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:

$$\boxed{k := \mathsf{KG}(\{\mathsf{n_0}, \mathsf{n_1}, \mathsf{n_2}\}, \mathsf{k'}); \mathsf{sbroadcast}(\{\mathsf{n_0}, \mathsf{n_1}\}, \mathsf{k}, \mathsf{m})}$$

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

Example 6. 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:

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

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 7. 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:

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

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 Security Properties and Main Results

We define confidentiality (securely flowing between nodes of different security levels) as a new noninterference property [35].

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 3 ($\tau - Equal$ 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 $\Gamma(\mathsf{x}) = \tau'\ \mathsf{var} \wedge \tau' \leq \tau$, *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 4 (Filtering). *Let* $\mathsf{filter}_\tau(\mathsf{t}) = \mathsf{filter}'_\tau(\mathsf{t}, [\,])$ *and* $\mathsf{filter}'_\tau([\,], \mathsf{t}) = \mathsf{t}$.

$$\mathsf{filter}'_\tau(\mathsf{BEnc}(\{\mathsf{n}_1,...,\mathsf{n}_\mathsf{k}\},\mathsf{vk},\mathsf{v}')\cdot \mathsf{t}',\mathsf{t}) = \begin{cases} \mathsf{filter}'_\tau(\mathsf{t}', \mathsf{t}\cdot(\{\mathsf{n}_1,...,\mathsf{n}_\mathsf{k}\},\mathsf{v}')) & \text{if } \Gamma(\mathsf{n}_1)\sqcap\Gamma(\mathsf{n}_2)...\sqcap\Gamma(\mathsf{n}_\mathsf{k}) \leq \tau \\ \mathsf{filter}'_\tau(\mathsf{t}',\mathsf{t}) & \textit{Otherwise} \end{cases}$$

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 5 (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* $\mu'_0 =^\Gamma_\tau \mu'_1 \wedge \mathsf{filter}(\mathsf{t}_0) = \mathsf{filter}(\mathsf{t}_1)$.

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

Theorem 1. *Let Γ be a typing environment and $\tau \in \mathcal{L}$ 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 $\Gamma \vdash p$. The proof is available on the following anonymous url: `https://anonymous.4open.science/r/Typing-System-Proof`.

Example 8. Concerning the security property, we show that the program p of Example 4 is noninterferent at M_1. We let two memories $\mu_0 = \{k := vk_0, k' := vk'_0, n_0 = \text{"}n_0\text{"}, n_1 = \text{"}n1\text{"}, m := v\}$ and $\mu_1 = \{k := vk_1, k' = vk'_1, n_0 = \text{"}n_0\text{"}, n_1 = \text{"}n1\text{"}, m := v\}$. Notice that $\mu_0 =^\Gamma_{M_1} \mu_1$. By Definition 3, $\mu_0 =^\Gamma_{M_1} \mu_1$ 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 5, 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 2: $\mathsf{BEnc}(\{\mathsf{n}_1,...,\mathsf{n}_\mathsf{k}\},\mathsf{k},\mathsf{v})$. 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 9. We show that the program p of Example 7 does not comply with NI^Γ_L according to Definition 5. Let two memories $\mu_0 = \{k := vk_3, k' := vk'_3, n_0 = \text{"}n_0\text{"}, n_1 = \text{"}n1\text{"}, m := 4, m' := 2\}$ and $\mu_1 = \{k := vk_4, k' = vk'_4, n_0 = \text{"}n_0\text{"}, n_1 = \text{"}n1\text{"}, m := 5, m' := 2\}$. 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 $\mu'_0 = \{k := vk_3, k' := vk'_3, n_0 = \text{"}n_0\text{"}, n_1 = \text{"}n1\text{"}, m := 4, m' := 4\}$ and $\mu'_1 = \{k := vk_4, k' = vk'_4, n_0 = \text{"}n_0\text{"}, n_1 = \text{"}n1\text{"}, m := 5, m' := 5\}$. 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^Γ_L.

6 Related Work

Broadcast Encryption and security classes. The concept of broadcast encryption was proposed by Fiat and Naor [18] 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 tradeoffs between key storage cost and transmission cost [1, 7, 8, 13]. Further improvements were done in the context of multicast protocols [9]. 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 [18] provides a simple and efficient solution for the case of dynamic groups (ie. user revocation in [9]). In terms of security notions, several schemes have been shown to achieve IND-CPA and IND-CCA [7, 12, 37, 39].

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)* [21, 24], 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 [7, 30]). None of these constructions, however, seem to exploit the underlying hierarchy to BE schemes with interesting properties among groups of receivers.

Secure information flow. Information flow policies [15, 35] 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. To prevent insecure flows, one may use static information flow enforcement [36] or dynamic enforcement [3, 5]. Both static and dynamic enforcements can be applied to our architecture; we choose to apply an static mechanism to enforce secure information flow through a type system. Several works propose type systems for secure information flow in distributed systems [4, 10, 11, 17, 19, 20, 27–29, 34, 38, 40]. In the literature of secure information flow, several works consider combining cryptography to information flow [2, 11, 19, 20, 23, 25] 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.

7 Conclusion

By mapping subgroups of nodes to security classes, and associating broadcast encryption keys to these, we propose enforcement of secure information flow between a server broadcasting messages to a set of nodes. We build a type system for server code and prove its soundness with respect to a new secure information flow property. To the best of our knowledge, we are the first to propose the association of security classes to cryptographic keys in BES in order to verify secure information flow.

References

[1] Agrawal, S., Yamada, S.: Optimal broadcast encryption from pairings and LWE. In: Advances in Cryptology - EUROCRYPT '20. Lecture Notes in Computer Science, vol. 12105, pp. 13–43. Springer (2020)

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

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

[4] Bastys, I., Piessens, F., Sabelfeld, A.: Prudent design principles for information flow control. In: Proceedings of the 13th Workshop on PLAS@CCS '18,. pp. 17–23. ACM (2018)

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

[6] Boneh, D., Franklin, M.K.: 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. Lecture Notes in Computer Science, vol. 1666, pp. 338–353. Springer (1999)

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

[8] Boneh, D., Waters, B., Zhandry, M.: Low overhead broadcast encryption from multilinear maps. In: Advances in Cryptology - CRYPTO '14. Lecture Notes in Computer Science, vol. 8616, pp. 206–223. Springer (2014)

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

[10] Cecchetti, E., Myers, A.C., Arden, O.: Nonmalleable information flow control. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17. pp. 1875–1891. ACM (2017)

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

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

[13] Delerablée, C., Paillier, P., Pointcheval, D.: Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. In: Pairing-Based Cryptography - Pairing '07. Lecture Notes in Computer Science, vol. 4575, pp. 39–59. Springer (2007)

[14] Denning, D.E.: A lattice model of secure information flow. Commun. ACM **19**(5), 236–243 (1976)
[15] Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Commun. ACM **20**(7), 504–513 (1977)
[16] El Laz, M., Grégoire, B., Rezk, T.: 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. pp. 310–321. ScitePress (2020)
[17] Ferraiuolo, A., Hua, W., Myers, A.C., Suh, G.E.: Secure information flow verification with mutable dependent types. In: Proceedings of the 54th Annual Design Automation Conference, DAC '17. pp. 6:1–6:6. ACM (2017)
[18] Fiat, A., Naor, M.: Broadcast encryption. In: Advances in Cryptology - CRYPTO '93. vol. 773, pp. 480–491 (1993)
[19] Fournet, C., Guernic, G.L., Rezk, T.: 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,. pp. 432–441. ACM (2009)
[20] Fournet, C., Rezk, T.: Cryptographically sound implementations for typed information-flow security. In: Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '08. pp. 323–335. ACM (2008)
[21] Gentry, C., Silverberg, A.: Hierarchical id-based cryptography. In: Advances in Cryptology - ASIACRYPT '02. Lecture Notes in Computer Science, vol. 2501, pp. 548–566. Springer (2002)
[22] Goldwasser, S., Micali, S.: Probabilistic encryption and how to play mental poker keeping secret all partial information. In: Lewis, H.R., Simons, B.B., Burkhard, W.A., Landweber, L.H. (eds.) Proceedings of the 14th Annual ACM Symposium on Theory of Computing. ACM (1982)
[23] Gregersen, S.O., Thomsen, S.E., Askarov, A.: A dependently typed library for static information-flow control in idris. CoRR **abs/1902.06590** (2019)
[24] Horwitz, J., Lynn, B.: Toward hierarchical identity-based encryption. In: Advances in Cryptology - EUROCRYPT '02. Lecture Notes in Computer Science, vol. 2332, pp. 466–481. Springer (2002)
[25] Laud, P.: Semantics and program analysis of computationally secure information flow. In: Programming Languages and Systems, 10th ESOP. Lecture Notes in Computer Science, vol. 2028 (2001)
[26] Lee, J., Kim, J., Oh, H.: BESTIE: broadcast encryption scheme for tiny iot equipments. IACR Cryptol. ePrint Arch. (2019)
[27] Magazinius, J., Askarov, A., Sabelfeld, A.: A lattice-based approach to mashup security. In: Proceedings of the 5th ACM Symposium on Information, Computer anD Communications Security, ASIACCS. pp. 15–23. ACM (2010)
[28] Matos, A.A., Cederquist, J.: Information flow in a distributed security setting. CoRR **abs/1901.01111** (2019)
[29] Murray, T.C., Sabelfeld, A., Bauer, L.: Special issue on verified information flow security. J. Comput. Secur. **25**(4-5), 319–321 (2017)
[30] Naor, D., Naor, M., Lotspiech, J.B.: Revocation and tracing schemes for stateless receivers. In: Advances in Cryptology - CRYPTO '01. p. 41–62. Lecture Notes in Computer Science, Springer (2001)
[31] Rackoff, C., Simon, D.R.: Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Advances in Cryptology - CRYPTO '91. Lecture Notes in Computer Science, vol. 576, pp. 433–444. Springer (1991)

[32] Rezk, T.: Secure Programming, HdR Thesis. Ph.D. thesis, University of Nice-Sophia Antipolis, France (April, 2018)
[33] Rezk, T.: Verification of Confidentiality Policies for Mobile Code, PhD Thesis. Ph.D. thesis, University of Nice-Sophia Antipolis, France (November, 2006)
[34] Russo, A., Sabelfeld, A.: Dynamic vs. static flow-sensitive security analysis. In: Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF '10. pp. 186–199. IEEE Computer Society (2010)
[35] Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE J. Sel. Areas Commun. **21**(1) (2003)
[36] Volpano, D.M., Irvine, C.E., Smith, G.: A sound type system for secure flow analysis. J. Comput. Secur. **4**(2/3), 167–188 (1996)
[37] Yang, Y.: Broadcast encryption based non-interactive key distribution in manets. J. Comput. Syst. Sci. **80**(3), 533–545 (2014)
[38] Zdancewic, S., Myers, A.C.: Secure information flow and CPS. In: Programming Languages and Systems, 10th European Symposium on Programming, ESOP '01. Lecture Notes in Computer Science, vol. 2028, pp. 46–61. Springer (2001)
[39] Zhang, L., Hu, Y., Wu, Q.: Adaptively secure identity-based broadcast encryption with constant size private keys and ciphertexts from the subgroups. Math. Comput. Model. **55**(1-2), 12–18 (2012)
[40] Zheng, L., Myers, A.C.: Dynamic security labels and static information flow control. Int. J. Inf. Sec. **6**(2-3), 67–84 (2007)