Formal Verification of Advanced Families of Security Protocols: E-Voting and APIs

Cyrille Wiedling

▶ **To cite this version:**

Cyrille Wiedling. Formal Verification of Advanced Families of Security Protocols: E-Voting and APIs. Cryptography and Security [cs.CR]. Université de Lorraine, 2014. English. NNT : 2014LORR0199 . tel-01107718v2

HAL Id: tel-01107718

https://theses.hal.science/tel-01107718v2

Submitted on 21 Jan 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Formal Verification of Advanced Families of Security Protocols: E-Voting and APIs

THÈSE

présentée et soutenue publiquement le 21 Novembre 2014

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Cyrille Wiedling

Composition du jury

Président :	Frédéric Cuppens	Télécom Bretagne, Brest.
Rapporteurs :	Bruno Blanchet	INRIA, Paris.
	Cas Cremers	Université d'Oxford, Oxford.
Examinateurs :	Véronique Cortier	CNRS, Nancy.
	Ralf Küsters	Université de Trèves, Trèves.
	Yassine Lakhnech	Université Joseph Fourier, Grenoble.
	Pierre-Etienne Moreau	Université de Lorraine, Nancy.
	Benjamin Morin	ANSSI, Paris.

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Remerciements

"You're only given one little spark of madness.
You mustn't lose it."
Robin Williams

En tout premier lieu, j'aimerai remercier tous les membres du jury. Un grand merci à Bruno Blanchet et Cas Cremers d'avoir accepté cette lourde tâche d'être rapporteurs et pour leur relecture attentive de ce manuscrit. Merci également à Ralf Küsters, Yassine Lakhnech et Benjamin Morin pour avoir accepté de participer à ce jury, et à Frédéric Cuppens pour l'avoir présidé. Je remercie également Pierre-Etienne Moreau pour avoir été un référent consciencieux pendant ces trois années. Enfin, je remercie Véronique Cortier, ma directrice de thèse, pour m'avoir fait découvrir la recherche lors de mon stage de Master et pour avoir fait de ces trois années de thèse, une expérience enrichissante, autant sur le plan scientifique que sur le plan humain; pour son soutien (probablement monnayé contre quelques gâteaux au chocolat), sa disponibilité et ses conseils; et, surtout, je salue son courage pour avoir, sans faillir, relu en détails de longues preuves jusqu'à ce que celles-ci soient parfaites.

Je tiens également à remercier l'ensemble de mes camarades de l'équipe CASSIS du LORIA à Nancy. Un remerciement particulier à Abdessamad IMINE, qui a été mon tuteur pour mon premier stage de Master. Merci à toute l'équipe pour l'ambiance et la bonne humeur qui rendaient le travail plus qu'agréable. Je remercie également Martine, pour son efficacité redoutable sur toutes mes questions administratives. Un merci particulier à l'ensemble de mes cobureaux : Mounira, Mathilde, Malika, David et Eric, avec lesquels il y avait toujours matière à se détendre lorsque cela devenait nécessaire et qui ont du composer avecmon goût prononcé pour le frais. Un merci particulier pour Eric qui m'a fait découvrir le Shogi, même si je n'ai pas été un élève très assidu; et pour Ludovic qui, je le sais, me remplacera avec brio en tant que Maître-Pâtissier de l'équipe. (Pour la plus grande joie de Véronique !)

Je voudrais aussi remercier mes amis, qui, même s'ils n'ont pas directement contribués à cette thèse, ont été un soutien moral important durant ces trois années. Merci à Anne pour avoir été une oreille attentive et avoir accepté de tenir compagnie à Réglisse lorsque je devais m'absenter à l'étranger. Merci à Mathieu, Jérôme et Arnaud, trio inspérable depuis la fac malgré la distance, pour nos soirées jeux de plateau et cette bulle d'air quasi-hebdomadaire. Un grand merci à Aurélie dont la relecture impitoyable d'une partie de ce manuscrit m'aura évité bon nombre de typos. Et, même si elle ne le lira probablement jamais, un merci à Réglisse, qui, malgré son comportement ronchon et parfois agaçant, a éclairé bon nombre de mes soirées.

Enfin, je terminerais simplement en remerciant mes parents et mes grands-parents, pour leur soutien constant, leur indéfectible confiance, et ce malgré les centaines de kilomètres qui nous séparent. C'est aussi grâce à eux que j'en suis ici aujourd'hui et je leur dédie ce manuscrit.

Contents

Introduction

"The multiple human needs and desires that demand privacy among two or more people in the midst of social life must inevitably lead to cryptology wherever men thrive and wherever they write."

David Kahn – *The Codebreakers*

For several decades, IT has gained a prominent place in most domains of our society and especially in our communications. Computers, smartphones or electronic tablets have become essential intermediates allowing people to overcome thousands miles of distance to talk to each other or to gain time by performing administrative procedures online without having to queue in official buildings. They also tend to replace human interaction in quite sensitive operations like bank transfers or payments. Whether by their speed, ease of use, or the fact that they allow people to stay connected to the whole world regardless of where they are, these devices have become, sometimes without questioning, true confidants entrusted with secrets such as bank account numbers, credit card information, private details, etc.

But even if this technology is well known for many years by the public, people are not always aware of the risks. One of them is identity theft. In France, according to a CREDOC study[1] in 2009, the number of reported identity thefts was about 250 thousands (overcoming the number of robberies by more that 25%). It mainly relies on the possibility for a malicious person to retrieve personal and useful information about someone on numerous documents that could be thrown away without precautions. Thus, although this problem is not inherent to this new technology, the increasing number of official documents, banking or personal information, sent over the Internet provides new opportunities for malicious individuals to achieve their purposes. Indeed, the CIFAS[2], the UK's Fraud Prevention Service, reports an always-increasing number of identity frauds from 2007 to 2012. It also gives a lot of advices to avoid these thefts and most of them concern electronic matters: limitation of private data published on social networks, protection of computers against viruses and malwares, etc.

These threats raise some questions: How do these systems work? How are they secured against potential attacks? How can we trust the security guarantees claimed by the system? To ensure privacy and security over the different networks, devices use *cryptographic protocols* (or *security protocols*) which are programs designed to ensure the security of communications (like the secret of a private information), often involving, as the name says, cryptography. Of course, it is not enough to claim the security of such protocols and there is a crucial need of verifying them before using them. If one protocol is found to be flawed, it can have a substantial impact depending on where it is used. Moreover, flaws need to be patched, either, in the best case, by

[1] A presentation of this report can be found at http://protegezvotreidentite.files.wordpress.com/2009/11/credoc-usurpation-didentite-en-france.pdf.

[2] Figures available at http://www.cifas.org.uk/identity_fraud.

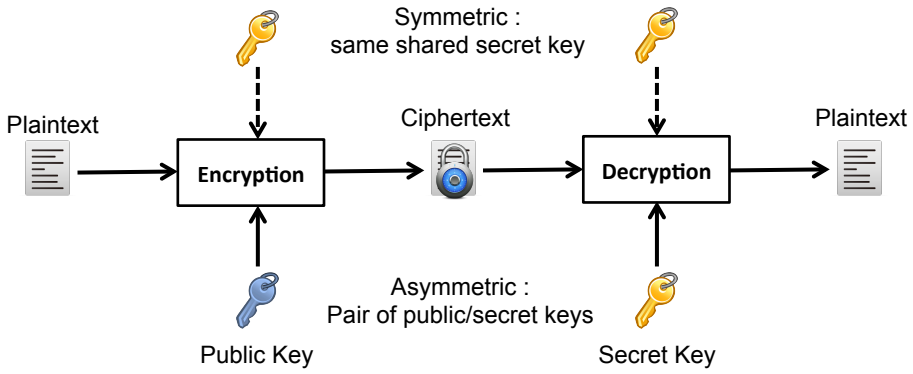


Figure 1: Symmetric Vs. Asymmetric Encryption Schemes.

a software update, or, in the worst case, by replacing defective hardwares, implying money and time costs.

1 Cryptographic Protocols

In the physical world, safely exchanging information between two or more participants requires particular layouts such as sealed envelopes, padlocked boxes or soundproof rooms. When it comes to the digital world, we use cryptographic protocols to ensure the security of our communications.

1.1 Cryptography

Based on two millennia of history and practice, cryptography aims at designing schemes which allow to hide, and therefore protect, some information from an adversary. Starting with Caesar and his first mono-alphabetical transposition (called *Caeser's cipher*), it has evolved along with technology, using the best of it, often for military purposes like the *Enigma Machine* during World War II. Nowadays, cryptography is not limited to military uses as the public needs for privacy and confidentiality make it essential for almost everyone around the globe.

Encryption schemes are functions, typically relying on a key k, designed to modify a *plaintext* into a *ciphertext* in such a way that it is not possible to retrieve the former using the latter without knowing the key. There are two main types of them: symmetric and asymmetric (see Fig.1). Symmetric schemes are designed in a way that the same key k is used for encryption (modification of the plaintext into a ciphertext) and decryption (reverse operation). DES (Data Encryption Standard), invented in 1977, and its successor, AES (Advanced Encryption Standard), are the most common symmetric schemes. On the asymmetric side, schemes involves a pair of *public* and *secret* keys. The public one, known by everyone on the network is used for encryption while the secret one, known only by its owner, is used for decryption. RSA (for Rivest, Shamir and Adlemann, its creators) is a widely used asymmetric scheme since 1978.

Curious readers may refer to the books of B. Schneier [Sch95] and D. Kahn [Kah96] for, respectively, a better overview of the different cryptographic schemes that have been or are currently used and a more detailed history of the early ages of this ever-changing science.

1.2 Protocols

Through decades, cryptographers worked hard to shape a perfect (i.e. unbreakable) cryptographic algorithm. But even if such algorithm may exist, that would not guarantee security in communications once and for all. Cryptography is merely a tool that needs to be used correctly to provide security. A *protocol* is a kind of manual that describes the way cryptographic algorithms should be used in order to get a specific task done. In communications, it is a set of operations designed, for example, to authenticate users together, to establish a secure channel over an insecure one, etc. Obviously, there are different protocols for each different goal and often several protocols proposed for each particular task (for efficiency, server load optimization, memory issues, etc.). Once again, curious readers may refer to [Sch95] for detailed examples. In this thesis, we are going to focus on two specific families of cryptographic protocols: electronic-voting protocols and APIs.

2 Electronic Voting

Voting is not a benign action. It often allows people to elect their representatives, expressing the fundamental principle of a republic. Elections can have serious political and economical impacts and, therefore, there is a crucial need to ensure security of the whole process, allowing people to express their own choices without any pressure, threats or risk of retaliation. In that way, paper-voting, using paper ballots, enveloppes and transparent ballot boxes, is one of the most secure system that can be used to provide strong guarantees to voters but it also have some drawbacks especially in the counting phase. Thus, a question raises: can we use current technology to upgrade this voting process and make it better, faster, stronger?

2.1 E-Voting Protocols

That question already raised about hundred years ago when the first lever-voting machines were introduced in the United States around 1990 [JS12]. These mechanical devices were designed to replace paper-ballot voting and speeding up both voting and counting processes. But these machines were also easy to tamper with, since the inner-mechanic could be modified by a specialist without arousing suspicion. More recently, mechanical technology has been replaced with electronic means with the first EVMs (Electronic Voting Machine) used in elections in 1996 in the United States. During the two last decades, electronic voting (or e-voting) spread out around the world, first with only EVMs and, in 2005, Estonia became the first country to offer Internet voting nationally in local elections.

There are two big families of e-voting protocols: direct-recording e-voting and remote e-voting. The former is physically supervised by representatives of governmental or independent electoral authorities directly in a polling station. The latter allows voters to vote from basically everywhere in the world, as soon as they get a connexion to the Internet through a computer. This thesis mainly focuses on remote e-voting protocols. To have some insights on how these protocols work, let us consider some small toy example. During the voting phase, voters cast their votes to the ballot box through their web-browsers after encrypting them using the public key of the election. Once this phase is closed, the ballot box (basically a program running on a server) decrypts the ballots using the secret key and publishes the results publicly. We can also describe this protocol in terms of exchange of messages between the participants (see Fig.2) to describe more precisely what is sent over the network during the execution of the protocol:

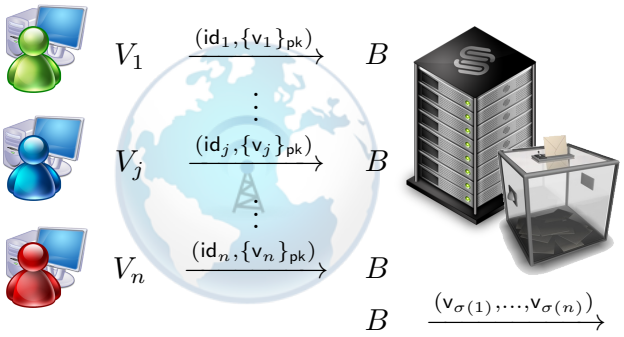


Figure 2: Messages Exchange in E_{toy}.

here the ballot box receives all pairs corresponding to all voters' ballots and then publishes the outcome which is a permutation of all decrypted ciphertexts.

Even if this toy-example looks quite simple, the principle of having a "black-box" collecting all different votes from players and then decrypting and publishing the outcome of the election is close to the current systems in use in France. Of course, such design provides very few guarantees in terms of security because the ballot box needs to be trusted to ensure anything. That is why several e-voting designs involves different election authorities with specific roles (collecting votes, decrypting votes, etc.) to weaken trust hypotheses.

2.2 Security Properties

E-voting protocols are designed to provide the same security as paper-ballot voting and the range of the security properties granted by the latter is large. During the analysis of a protocol, one major difficulty is to properly define which property will be proven. This section provides an overview of different security properties that are relevant in the e-voting field.

The first property reflects the major and must-have security guarantee someone should provide in any e-voting protocol: the absence of capabilities, for a malicious person, to discover how a person voted during an election. It is called *ballot-secrecy*. A formal definition of this property, based on equivalences, has been given in [DKR09]. Intuitively, it considers two executions of the protocol where two honest voters swap their votes. If an adversary cannot make a difference between the two executions, then, the protocol satisfies ballot-secrecy. Other definitions of ballot-secrecy exist, in different models, and have been used [KTV11, BPW12] for protocol analysis. In this thesis, the definition of [DKR09] will be retained.

Two enhancements of this property are *receipt-freeness* [DKR09] and *coercion-resistance* [ALBD04, DKR09]. While the former refers to the absence of possibilities, for the voter, to prove how he voted during the election, the latter denotes the inability, of a malicious person, to perform vote-buying or to threaten people in order to make them vote a specific option. In [DKR09], these three notions are shown to be related; coercion-resistance implies receipt-freeness which itself implies ballot-secrecy. In a more quantitative setting, [KTV11] shows that these implications do not necessarily hold.

Other security properties are desirable like *verifiability* and *accountability*. There exist several kind of verifiability. *Individual verifiability* states that a voter is able to check that his own ballot has been correctly included in the list of counted ballots; *universal verifiability* corresponds to the

fact that everyone can check if the result of the election corresponds to all the ballots submitted by the voters. Formal definitions of these properties can be found in [KRS10] and [KTV11]. *Eligibility verifiability* [FOO92, JLY98] denotes the fact that only registered voters can vote in an election. Finally, *accountability* [KTV10] provides the possibility to detect misbehaviors and impute liability to the responsible election authority. Stronger that verifiability, accountability reports that something went wrong during the process but also points who is responsible. We can also mention *correctness*, introduced in [JCJ05], which refers to the fact that the published result corresponds to the (sum of) intended votes of the voters.

2.3 Attacks

An e-voting protocol often involves numerous cryptographic primitives but using cryptography does not mean it will lead to a safe protocol. For example, in our protocol E_{toy} we use an asymmetric encryption scheme to protect the voters' votes but there is still a way for an attacker to break the ballot-secrecy property. Let us suppose, for simplicity, that we only have three voters, V_{A} and V_{B}, respectively Alice and Bob, two honest voters, and V_{C}, Charlie, a malicious one whose goal is to know how the others voted. We will also suppose that Charlie has control of the network (this hypothesis is usual in protocol analysis).

The scenario of the attack is as follows. During the voting phase, Alice and Bob submit their ballots $(id_{\mathsf{A}}, \{v_{\mathsf{A}}\}_{\mathsf{pk}})$ and $(id_{\mathsf{B}}, \{v_{\mathsf{B}}\}_{\mathsf{pk}})$ respectively. Charlie, who is listening to the network will see these ballots without being able to decrypt them because he is not in possession of the secret key only held by the Ballot Box. But he can retrieve one ciphertext (either $\{v_{\mathsf{A}}\}_{\mathsf{pk}}$ or $\{v_{\mathsf{B}}\}_{\mathsf{pk}}$) and use it as if it was his vote. Thus, by sending $(id_{\mathsf{C}}, \{v_{\mathsf{A}}\}_{\mathsf{pk}})$ to the Ballot Box, he would vote as Alice did. During the tallying phase, the Ballot Box will output the result $\{v_{\mathsf{A}}, v_{\mathsf{A}}, v_{\mathsf{B}}\}$, then Charlie will now that Alice voted for the candidate with two voices and Bob for the other one.

One can also note that this protocol does not provide any verifiability, since the ballot box behaves as a black box taking some ballots as inputs and returning the outcome of the election. There is no way to ensure, without trusting the ballot box, that the outcome corresponds to the submitted ballots sent by the voters. Nevertheless, this very basic attack illustrates that cryptography does not prevent attacks by itself. Moreover, the more complex is a protocol specification, the more difficult it is to see immediately if there are flaws. That explains why there is a real need of analyzing cryptographic protocols in order to ensure that they satisfy the desired security properties.

2.4 Existing Protocols

Since e-voting is a quite hot topic since around two decades, literature provides several propositions of e-voting protocols. Based on different settings, they achieve different security properties. Here is a non-exhaustive list of existing e-voting protocols. We provide a small description of how they work and which security properties they provide.

FOO This is a scheme proposed in [FOO92]. It relies on two authorities: an administrator and a collector and requires blind signatures and zero-knowledge proofs. This protocol assumes anonymous channels between the voters and the collector. It is composed of four steps: a setup phase, a registration phase where each voter gets a token from the administrator (this step relies on the blind signature scheme), a voting phase where voters send their encrypted vote, including the token, to the collector and a counting phase where the collector decrypts, counts

and publishes the result. It has been showed to satisfy ballot-secrecy [KR05] and it claims individual verifiability but it is also vulnerable to some attacks [KR05].

Okamoto et al. This system [OMA+99] is based on the FOO protocol. It requires an administrator, several mixers and talliers and is based on blind signatures and a threshold encryption scheme. The scheme is designed in three phases: during the registration, the voter prepares a ballot and gets an authorization from the administrator by obtaining a blind signature from the administrator. Then, during voting phase, the voter sends his ballot anonymously, using a mix-net. Finally, during the counting phase, talliers co-operatively open the ballots and count the votes. This protocol, widely inspired from FOO, is meant to satisfy the same security properties but suffers from the same attacks.

Civitas Inspired from a previous cryptographic voting scheme developed in [JCJ05], this protocol is described in [CCM08]. It uses a public encryption scheme supporting homomorphic properties, random re-encryption and distributed decryption, but also requires distributed plaintext equivalence tests (PETs), non-interactive zero-knowledge proofs and mix networks. The protocol participants include a supervisor monitoring the election, a registrar and registration tellers respectively responsible of authorizing eligible voters to participate to the election and jointly creating credentials for them, and tabulation tellers which will jointly decrypt and publish the outcome of the election. It is one of the strongest systems in terms of security. Indeed, it provides coercion-resistance (implying ballot-secrecy), individual and universal verifiability.

Helios & Belenios First web-based, open-audit voting system [Adi08], Helios involves an ElGamal homomorphic encryption scheme which allows re-encryption, zero-knowledge proofs and mix networks. The architecture relies on a public ballot box displaying information on its content allowing verifications from voters who use their web browser to perform cryptographic operations. This protocol aims to ensure ballot-secrecy when the Helios server is trusted and is fully verifiable by anyone, but attacks has been discovered and fixed in [CS13]. Based on the Helios system, Belenios has been developed to provide a system achieving ballot-secrecy, verifiability (individual and universal) and correctness [CGGI13] when the latter was not provided by the Helios system.

Caveat Coercitor This design [GRBR13] is mainly inspired from Civitas voting scheme and, therefore, it uses the cryptographic primitives involved in it and follows the same sequence of phases. Caveat Coercitor defines two new security properties *coercion-evidence* and *coercer independence*. The former is performed against the bulletin board to accurately determine the degree of coercion that has taken place in any given run while the latter is the ability of the voter to follow the protocol without being detected by the coercer. These new security properties verified by Caveat Coercitor implies less trust assumptions than other remote e-voting schemes like Civitas or Helios.

As mentioned at the beginning of this chapter, there also exist e-voting protocols that are used, in opposition with remote e-voting, directly in the polling station. This kind of voting systems are not the focus of this thesis but are interesting too. Here are some examples of such protocols.

Prêt-à-Voter This protocol presented in [CRS05] uses paper-ballots with two different parts: a randomized candidates list and corresponding empty boxes with a printed unique ID depending

on the permutation of choices and a secure probabilistic encryption scheme. This protocol satisfies receipt-freeness (thus ballot-secrecy), individual and universal verifiability.

Three-Ballots This voting scheme is paper-ballot based [Riv06] and has the particularity to involve no cryptography to ensure security properties. Each voter receives a paper which contains three identical ballots except their unique ID code printed on the bottom. Intuitively, as the name refers to it, each voter submits three ballots. (Since the system is quite visual, a curious reader should definitely have a look to [Riv06] for a more detailed presentation.) Three-Ballots ensure coercion-resistance, individual and universal verifiability.

Scantegrity This protocol is an open source election verification technology [SFCC11] for optical scan voting systems. It uses confirmation numbers to allow each voter to verify her vote is counted. The confirmation numbers also allow anyone to verify that all the votes were counted correctly. Scantegrity ensures ballot-secrecy, individual and universal verifiability.

3 Application Programming Interfaces

With the evolution of technology, embedded systems have increased and, in an attempt for security when they are deployed in hostile environments, they often include a dedicated temper-resistant and secure hardware. This item handles all cryptographic operations and/or is meant to keep all the different keys secure. Such systems exist in our mobile phones or smartphones (SIM card or secure elements), transports ticketing systems (smart cards), ATMs, etc. But it is not enough to use such hardware, we also need an interface to securely access to it. That is the purpose of *secure application programming interfaces* (APIs).

3.1 Security APIs

A *security API* is designed to allow an untrusted machine, or program, to access a trusted device in a secure way. It can be seen as an interface between two entities with different levels of trust. Intuitively, it is the interface between a credit-card (smart-card), which is a trusted device, and the ATM, obviously untrusted, only allowing to perform operation if the correct PIN is entered; or the interface between a cryptographic hardware security module and the client machine where it will be plugged. As a protocol, the security API aims to achieve a goal, which is to enforce security properties, no matter what sequence of commands are called. Basically, a security API should prevent any leak of critical data.

Let us consider a toy example on a tamper-resistant device (TRD) designed to store key values, used for encryption and decryption. The TRD's memory is represented by a table where each entry is labelled by a handle, which corresponds to the name of the entry, and contains a key value. Handles can be used by users to manipulate keys without knowing their actual value. For example, when a secure key is received (encrypted) by a token, it is stored inside the token and the user is only given a handle corresponding to that key. We consider the interface A_{toy} which allows a user to use two commands: encrypt and decrypt. As shown in Fig.3, the encryption call takes as arguments the data (a specified plaintext or a handle) that needs to be encrypted and the handle pointing to the key that must be used to perform the encryption. The decryption call works similarly with a ciphertext and a handle as arguments. Connecting the TRD to a host machine, a user can use the key stored inside the device to perform arbitrary encryptions and decryptions.

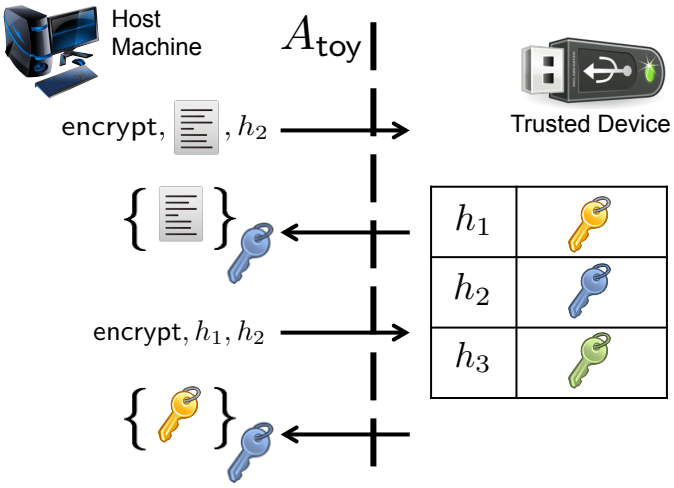


Figure 3: The command encrypt of A_{toy}, an API example.

Security Properties Unlike e-voting protocols where security properties are well identified like ballot-secrecy and verifiability, APIs have, in terms of security, quite specific needs depending on the purpose of the interface itself. Basically, an API is often designed to prevent any key leakage from a given cryptographic token and this will correspond to a first security property. For each design, other properties may be defined depending on its specificities.

3.2 Existing Designs

Security APIs can be used in a very wide range of possibilities. There exist different academic examples, open standards and proprietary solutions. This section provides a non-exhaustive list of them:

- C. Cachin and N. Chandran designed [CC09] a cryptographic token interface that supports multiple users and provides operations for key generation, encryption, authentication, and key wrapping. Their model defines an Access Control List (ACL) for every key. They also show how to implement their token securely from a set of cryptographic primitives and prove that it respects the security policy expressed by the ACL.

- The IBM Common Cryptographic Architecture [IBM06] is a proprietary security API supported by specific IBMs cryptographic coprocessors. It is widely used in ATM (cash machines) networks since it provides several functions that are useful in the context of banking like encryption, decryption and verification of Personal Identification Numbers (PINs).

- Trusted Platform Modules [Rya09] are hardware chips designed to store cryptographic keys or sensitive date in a shielded memory. It also provides an API to allow softwares to use these stored information. TPMs are widely present in current computers, especially laptops of several well-known manufacturers and allow applications like full disk encryption or platforms integrity and authentication.

- The Public Key Cryptographic Standard #11 [Inc04] (PKCS#11, also known as "Cryptoki") is a platform-independent API designed for cryptographic tokens like smart-cards and Hardware Security Modules. It defines most commonly used cryptographic object

types (RSA keys, Certificates, etc.) and all the functions needed to use, create/generate, modify and delete those objects.

Attacks Like other protocols, APIs are vulnerable to attacks. In a way, they may be more vulnerable since they only define the behavior of different functions where most of protocols specify the whole sequence of messages that will be exchanged during the execution. In the case of APIs, if a malicious user gets access to a secured device, he can use any sequence of any functions available in the interface. Thus, there is a need to ensure that it does not exist such sequence of commands that could lead to a violation of the desired security properties.

In the example of A_{toy} there is clearly an issue. Since there is no restriction on what can be encrypted or decrypted, we can easily construct a sequence of commands that would leak information on keys to any owner of a secure token. Indeed, a malicious user could just ask for an encryption of an handle by another, for example, (encrypt, h_1, h_2) and get an encryption C of the key stored in h_1. Then, he would ask to decrypt this just-created ciphertext using the command (decrypt, C, h_2) and get the key in clear.

Several attacks have been found on deployed APIs, mainly on implementations of PKCS#11 [Clu03, DKS08] since it is the most used and studied standard. More recently, R. Focardi and G. Steel developed a tool named Tookan [BCFS10] (TOOl for cryptoKi ANalysis) performing automated reverse-engineering of real cryptographic tokens implementing PKCS#11. On 18 studied tokens, 10 of them were found to be flawed while the 8 others were only working with really limited functionalities. Flaws have been also found on the IBM 4758 CCA. [BA01] shows that malicious users may get access to the PIN exploiting the properties of XOR operation.

4 Analysis of Cryptographic Protocols

As we have seen in previous sections, verifying protocols is crucial. To perform such analysis the first step is to represent the behavior of the system, including its environment and a potential attacker by a model.

4.1 Existing Models

There exist two different families of models. *Computational models* consider messages sent over the network as bitstrings. The attacker is abstracted as any probabilistic polynomial-time Turing machine and cryptographic primitives are represented by polynomial algorithms. The main advantage of computational models is that they give strong guarantees on security properties for the studied cryptographic protocols. On the other hand, proofs may be tedious and error prone, even if they have been some results leading to automation of computational verification like EasyCrypt [Com].

On the other side, there are the *symbolic models*. In those, messages sent over the network are abstracted by terms and cryptographic primitives are supposed perfect and modeled as function symbols. Then, the whole protocol is abstracted and the intruder's capabilities are fixed by a set of rules. In some ways, symbolic models are much simpler that computational models, and, naturally, proving security properties in the former becomes simpler as well, often leading to possibilities of automation. The literature provides numerous symbolic models like *strand spaces* [THG99], *constraint systems* [MS01, CLS03] or *Horn Clauses* [Wei99, Bla01, CLC03]. Each model usually fits better for a specific application. A large part of this thesis is based on another symbolic model, the *applied-pi calculus* [AF01].

Abstractions made by symbolic models lead to weaker guarantees on security properties verification than a computational model. Typically, the latter is more accurate, especially in terms of attacks found: if an attack in the symbolic model always has a counterpart in the computational one, the opposite is not always true. Several works attempt to bridge the gap between the two universes and identify when security in symbolic models implies security in computational models [AR00, BDK07, CLC08] especially in order to keep simplicity of the symbolic approach while gaining accuracy of the computational one.

In this thesis, all the different analyses have been performed using the symbolic approach.

4.2 Existing Results

Even with some advantages over the computational model, the analysis of cryptographic protocols in the symbolic model remains a difficult problem. Verifying security properties quickly becomes undecidable [DLMS99]. Decidability results require to consider restrictions on the protocol or on the attacker. One of the most common restrictions consists in bounding the number of sessions. This is sufficient to restore decidability [RT01]. Further solutions consist in restraining the set of considered cryptographic primitives or in abstracting the model.

Trace Properties Several results in symbolic approach concern trace properties which are used to define security properties like secrecy or authentication. Intuitively, these security properties can be seen as a reachability problem, where the analysis reveals if an execution of the protocol may contradict the security property (e.g. a secret information is leaked). There exist several decidability results on trace properties. Different tools have also been developed to automatically analyze such properties like ProVerif [Bla01] based on Horn clauses. Although it may sometimes not terminate, this tool can handle unbounded number of sessions of protocols using a wide class of cryptographic primitives. Other tools perform automatized analyses of cryptographic protocols like AVISPA [Vig06], using the High-Level Protocol Specification HLPSL Language, or Scyther [Cre08] based on a pattern refinement algorithm, providing concise representations of (infinite) sets of traces.

Equivalence Properties Nevertheless, as mentioned in Section 2.2, e-voting security properties are expressed through equivalence. For example, ballot-secrecy is defined by the absence of difference between two executions of the protocol, formally defined as an observational equivalence. First decidability results for such properties come with [Hüt02] in the *spi-calculus* model. Decision procedures for trace equivalence with bounded number of sessions are shown in [Bau05, CD09] for a specific class of protocols (determinate protocols), or in [CR05] for specific class of primitives. More recent results [CCD13, CCD14] provide first decision procedures for an unbounded number of sessions for a large class of determinate protocols. Several tools have also been designed to automatically analyze equivalence-based properties like SPEC [TD10] based on the spi-calculus model, AKiSs [cC01] using Horn clauses for trace equivalence or APTE [Che14], that implements a complete and sound algorithm based on constraint systems. Moreover, although it was first designed for trace properties, ProVerif has also been extended to be able to deal with some equivalences [BAF08].

Analysis of APIs As we mentioned it in Section 3.2, security APIs have been shown to be vulnerable to attacks. The first security models emerged using the Dolev-Yao's approach [CM06]. Typically "hand-made" from the API specification, they are written using Horn clauses or rewriting rules, representing the APIs by an non-monotonic evolving state. Finding an attack usually

consists in testing the reachability of specific state or derivability of a given secret. Such analysis can be automatized using model checkers or theorem provers [LR92, CKS07, KT08]. Nevertheless, mechanized analysis of APIs still remains a difficult issue. For example, existing tools, using protocols analysis like ProVerif, are not working well with an unbounded number of keys. Different approaches still exist using Horn clause resolution like StatVerif [ARR11] or multiset rewriting rules like Tamarin [MSCB13] which is still in an early development stage.

4.3 Limitations

Despite all existing results and constant progresses in the domain of cryptographic protocols analysis, there are still some limitations, especially when it comes to e-voting schemes.

Voting Protocols In order to ensure security properties like ballot-secrecy or verifiability, remote e-voting protocols often involve several cryptographic primitives ranging from specific encryption schemes (homomorphic, distributed encryption) to malleable (or not) zero-knowledge proofs through signatures, mix networks and PETs. Even if these primitives can often be managed separately by tools like ProVerif, e-voting designs required the use of all these functions at the same time making things a lot more difficult because of all the different possible interactions. Thus, it is not rare that challenging automated tools with e-voting processes does not yield a successful result, implying that analysis of such protocols since remains to be done by hand.

Security APIs Providing security analysis of APIs designs is often more complicated than for usual protocols. Indeed, the former typically consist in dozens of functions, which may be called in any order, while the latter usually present an exchange of half a dozen messages in a given order. Thus, the abstractions, search techniques and tools are different. Moreover, APIs commands often share the same cryptographic keys, which limits the possibility of analyzing the different commands in a modular way before composing the results. Finally, security APIs models include, in general, a global non-monotonic mutable state, which needs to be modeled with precision in order to get accurate results and avoid false attacks. Such models can be difficult to make using classical approaches of protocol analysis and, therefore, special approaches need to be developed for APIs analysis.

5 Contributions

This thesis is divided in three distinct parts. Part I presents formal studies of examples of electronic voting protocols, based on a symbolic analysis using the applied pi-calculus. Part II is devoted to the presentation of a new design of security API, which supports a revocation functionality, for symmetric keys management. Finally, Part III describes a new method to apply type-systems in order to automate ballot-secrecy proofs for e-voting protocols. All the contributions of this thesis are detailed below.

5.1 Formal Analysis of E-Voting Protocols

In Chapter 2, we discuss the security of the Norwegian internet voting protocol, which has been deployed in Norway and has been used for local elections in more that ten municipalities by around 70'000 voters. After a formal presentation of the protocol itself, we analyse its security through different corruption scenarios. This formal study is performed using the applied pi-calculus [AF01]. Thus, we first provide a complete modeling of the protocol in this syntax.

Then, we show that this specification of the Norwegian e-voting protocol satisfies ballot-secrecy, according to the definition given by [DKR09], in different corruption cases. Moreover, we complete this analysis by displaying some attacks on the protocol in particular corruption cases. At our knowledge, it is the first formal proof of security of this protocol.

In Chapter 3, we provide a formal analysis of another protocol: the CNRS boardroom voting protocol. This is a voting protocol, which aims to ease the voting sessions during boardroom meetings. The interest of this system remains in its simplicity since it does not involve cryptography. Nevertheless, we prove that this protocol still can preserve ballot-secrecy and correctness. In this chapter, we provide three different implementations of this protocol and perform a full analysis, using the applied pi-calculus model, of them - regarding ballot-secrecy and correctness.

5.2 A Secure API with Revocation

In Chapter 4, we present a new design for security APIs, which includes a revocation functionality, for symmetric keys management. Building upon a previous work from G. Steel and V. Cortier [CS09], which do not support revocation, we propose a new set of commands that allow to manage symmetric keys and, especially, the possibility to revoke specific keys in the case where they may have been corrupted. In this chapter, we detail the requirements and the different commands of our API. Then, we describe the formal model we used to analyze this design, based on global states and transitions.

We provide a full formal study of our API, ensuring three different security properties. The first one states that the keys stored in the trusted devices remain secure even if the attacker manages to corrupt a few of them (e.g. using side-channels attacks). The second one implies that the overall system is able to repair itself after a certain time. Basically, even if an attacker manages to break into a device through a corrupted key, he will not be able to use it to gain more power in the device. Finally, we show how our revocation functionality helps to immediately secure devices against corrupted keys.

5.3 Type-Systems for Verification of E-Voting Protocols

The last chapter presents a new method for automatic verification of ballot-secrecy for electronic voting protocols. To conduct this approach, we based our work on a new type-system presented by C. Fournet and *al.* [BFG+14], RF*. Indeed, this type-system is able to tackle equivalence-based properties. Therefore, we propose, in this chapter, a formalization of ballot-secrecy based on the type-checking approach. Moreover, we provide an example based on the Helios protocol. We describe how to implement this voting system into the RF* syntax, and we detail a cryptographic library to handle the primitives used in the Helios protocol, especially the homomorphic encryption property that is not well handled by tools like ProVerif [Bla01].

5.4 Research Publications

The contribution presented in Chapter 2, 3 and 4 have been published in proceedings of the following conferences POST'12 [CW12], Vote-ID'13 [ACW13] and CCS'12 [CSW12] respectively. They also represent joint work with Mathilde Arnaud (for the second article) and Graham Steel (for the third article). The third part has not been yet been published but is a still ongoing work in collaboration with Matteo Maffei, Fabienne Eigner and Steve Kremer.

Part I

Formal Analysis of E-Voting Protocols

Chapter 1

Applied Pi-Calculus

Contents

We describe here the applied pi-calculus [AF01], introduced by M. Abadi et C. Fournet. The applied pi-calculus is a process algebra that is often used to model protocols. In particular, Chapter 2 presents a formal model of the Norwegian protocol using the applied pi-calculus while we propose, in Chapter 3, a model of the CNRS protocol. In this chapter, we briefly recall the notations and definitions of the applied pi-calculus, providing some examples.

1.1 Terms

Messages are represented by *terms* built upon an infinite set of *names* $\mathcal{N}$ (for communication channels or atomic data), a set of *variables* $\mathcal{X}$ and a *signature* Σ consisting of a finite set of *function symbols* (to represent cryptographic primitives). A function symbol f is assumed to be given with its arity $\mathsf{ar}(f)$. Then, the set of terms $T(\Sigma, \mathcal{X}, \mathcal{N})$ is formally defined by the following grammar:

$$
\begin{array}{ll}
t, t_1, t_2, \ldots ::= & \\
\quad x & x \in \mathcal{X} \\
\quad n & n \in \mathcal{N} \\
\quad f(t_1, \ldots, t_n) & f \in \Sigma,\ n = \mathsf{ar}(f)
\end{array}
$$

We write $\{^{M_1}/_{x_1}, \ldots, ^{M_n}/_{x_n}\}$ for the *substitution* that replaces the variables x_i with the terms M_i. $N\sigma$ refers to the result of applying substitution σ to the free variables of the term N. A term is called *ground* when it does not contain variables.

In order to represent the properties of the primitives, the signature Σ is equipped with an *equational theory* E that is a set of equations which hold on terms built from the signature. We denote by $=_E$ the smallest equivalence relation induced by E, closed under application of

function symbols, substitutions of terms for variables and bijective renaming of names. We write $M =_E N$ when the equation $M = N$ holds in the theory E.

Example 1. *A possible signature for representing symmetric, asymmetric encryption and signature is* $\Sigma = \{\mathsf{checksign}, \mathsf{dec}, \mathsf{pdec}, \mathsf{enc}, \mathsf{penc}, \mathsf{sign}\}$ *where* penc *and* pdec *represent resp. asymmetric (randomized) encryption and decryption,* enc *and* dec *stand resp. for symmetric (randomized) encryption and decryption,* sign *models signature, and* checksign *a function to check the validity of signature. For example, the term* $\mathsf{enc}(m, r, k)$ *represents the symmetric encryption of the message* m *with the key* k *and random factor* r*. The properties of the cryptographic functions are represented by the following equational theory* E_{enc}*:*

$$\begin{aligned} \mathsf{dec}(\mathsf{enc}(m, r, k), k) &= m \\ \mathsf{pdec}(\mathsf{penc}(m, r, \mathsf{pk}(i)), \mathsf{sk}(i)) &= m \\ \mathsf{checksign}(\mathsf{sign}(m, \mathsf{sk}(i)), \mathsf{pk}(i), m) &= \mathsf{ok}. \end{aligned}$$

The first equation models that symmetric decryption is only successful if the key used for decryption is the same as the one used for encryption; the second equation reflects that asymmetric decryption only succeeds when the corresponding secret key is used; and finally, the last equation checks that a signature corresponds to a given message and a given verification key.

Some cryptographic primitives, like homomorphic encryption, need the use of associative and commutative symbols in order to be fully described. Equational theories including such symbols are called *AC-theories*. For them, we define an equality modulo AC, noted $=_{AC}$, which denotes that two terms are syntactically equal modulo associative or commutative properties included in the equational theory.

Example 2. *To represent the homomorphic encryption, we can use the signature* $\Sigma = \{+, *, \mathsf{enc}\}$ *with* $+$ *and* $*$ *two associative and commutative (AC) symbols and the corresponding equational theory* E_{AC} *which includes the associative and commutative properties of* $+$ *and* $*$ *symbols, and the following equation:*

$$\mathsf{enc}(m, k) * \mathsf{enc}(n, k) = \mathsf{enc}(m + n, k).$$

This equation denotes that two ciphertexts encrypted with the same key can be merged together to create a single ciphertext whose corresponding plaintext is the sum of the two previous plaintexts. In this example, we have that $\mathsf{enc}(m + n, k) =_{AC} \mathsf{enc}(n + m, k)$ *and* $\mathsf{enc}(m, k_1) * \mathsf{enc}(n, k_2) =_{AC} \mathsf{enc}(n, k_2) * \mathsf{enc}(m, k_1)$ *but these two equalities only holds modulo AC.*

1.2 Rewriting System

It might be difficult to work modulo an equational theory. Instead, it is often possible (and more convenient) to reason with a rewriting system. Formally, a *rewriting system* $\mathcal{R}$, a set of *rewriting rules* of the form $l \to r$ with l and r two terms. We say that a term s is *rewritten* in t for rule $l \to r$, noted $s \to t$, if there exists a position p in s and a substitution θ such that $s|_p = l\theta$ and $t = s[r\theta]_p$.

Definition 1.1 (Convergence)**.** *A rewriting system* $\mathcal{R}$ *is said* convergent *if:*

- *for every closed term* U*, there exists no infinite sequence* $U \to U_1 \to \cdots \to U_k \to \ldots$*. (In this case, we say that* $\mathcal{R}$ *is terminating.)*

$P, Q, R ::=$	(plain) processes
0	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu n.P$	name restriction
if ϕ then P else Q	conditional
$u(x).P$	message input
$\overline{u}\langle M\rangle.P$	message output
$A, B, C ::=$	extended processes
P	plain process
$A \mid B$	parallel composition
$\nu n.A$	name restriction
$\nu x.A$	variable restriction
$\{^M/_x\}$	active substitution

Figure 1.1: Syntax for processes

- *for every closed terms U, U_1, and U_2 such that $U \to^* U_1$ and $U \to^* U_2$, there exists V such that $U_1 \to^* V$, and $U_2 \to V$. (In this case, we say that $\mathcal{R}$ is confluent.)*

Since it is impossible to associate a finite and convergent rewriting system to an equational theory including associative and commutative (AC) equations, we consider, for these AC-theories, a rewriting system modulo AC. A term s is rewritten modulo AC in t, noted $s \to_{AC} t$, for a rule $l \to r$ if there exist s' and t' two terms such that $s =_{AC} s'$, $t =_{AC} t'$ and s' is rewritten in t' for the rule $l \to r$. Then, it is possible to define the notion of AC-convergence.

Definition 1.2 (AC-Convergence)**.** *A rewriting system $\mathcal{R}$ is said AC-convergent if:*

- *for every closed term U, there is no infinite sequence $U \to_{AC} U_1 \to_{AC} \cdots \to_{AC} U_k \ldots$. (In this case, we say that $\mathcal{R}$ is AC-terminating.)*
- *for every closed terms U, U_1, and U_2 such that $U \to^*_{AC} U_1$ and $U \to^*_{AC} U_2$, there exists V such that $U_1 \to^*_{AC} V$, and $U_2 \to^*_{AC} V$. (In this case, we say that $\mathcal{R}$ is AC-confluent.)*

1.3 Processes

Processes and *extended processes* are defined in Figure 1.1. The process 0 represents the null process that does nothing. $P \mid Q$ denotes the parallel composition of P with Q while $!P$ denotes the unbounded replication of P (*i.e.* the unbounded parallel composition of P with itself). $\nu n.P$ creates a fresh name n and then behaves like P. The process if ϕ then P else Q behaves like P if ϕ holds and like Q otherwise. $u(x).P$ inputs some message in the variable x on channel u and then behaves like P while $\overline{u}\langle M\rangle.P$ outputs M on channel u and then behaves like P. We write $\nu\tilde{u}$ for the (possibly empty) series of pairwise-distinct binders $\nu u_1 \ldots . \nu u_n$. The active substitution $\{^M/_x\}$ can replace the variable x for the term M in every process it comes into contact with and this behaviour can be controlled by restriction, in particular, the process $\nu x\left(\{^M/_x\} \mid P\right)$ corresponds exactly to let $x = M$ in P.

As in [CS13], we slightly extend the applied pi-calculus by letting conditional branches now depend on formulae $\phi, \psi ::= M = N \mid M \neq N \mid \phi \wedge \psi$. If M and N are ground, we define

Par − 0	A	$\equiv$	$A \mid 0$	
Par-A	$A \mid (B \mid C)$	$\equiv$	$(A \mid B) \mid C$	
Par-C	$A \mid B$	$\equiv$	$B \mid A$	
Repl	$!P$	$\equiv$	$P \mid !P$	
New − 0	$\nu n.0$	$\equiv$	0	
New-C	$\nu u.\nu w.A$	$\equiv$	$\nu w.\nu u.A$	
New-Par	$A \mid \nu u.B$	$\equiv$	$\nu u.(A \mid B)$	if $u \notin \mathsf{fv}(A) \cup \mathsf{fn}(A)$
Alias	$\nu x.\{^M/_x\}$	$\equiv$	0	
Subst	$\{^M/_x\} \mid A$	$\equiv$	$\{^M/_x\} \mid A\{^M/_x\}$	
Rewrite	$\{^M/_x\}$	$\equiv$	$\{^N/_x\}$	if $M =_E N$

Figure 1.2: Structural equivalence.

$[\![M = N]\!]$ to be true if $M =_E N$ and false otherwise. The semantics of $[\![\ ]\!]$ is then extended to formulae as expected.

The *scope* of names and variables is delimited by binders $u(x)$ and νu. Sets of bound names, bound variables, free names and free variables are respectively written $\mathsf{bn}(A)$, $\mathsf{bv}(A)$, $\mathsf{fn}(A)$ and $\mathsf{fv}(A)$. Occasionally, we write $\mathsf{fn}(M)$ (respectively $\mathsf{fv}(M)$) for the set of names (respectively variables) which appear in term M. An extended process is *closed* if all its variables are either bound or defined by an active substitution.

An *context* $C[\_]$ is an extended process with a hole instead of an extended process. We obtain $C[A]$ as the result of filling $C[\_]$'s hole with the extended process A. An *evaluation context* is a context whose hole is not in the scope of a replication, a conditional, an input or an output. A context $C[\_]$ closes A when $C[A]$ is closed.

A *frame* is an extended process built up from the null process 0 and active substitutions composed by parallel composition and restriction. The *domain* of a frame φ, denoted $\mathsf{dom}(\varphi)$ is the set of variables for which φ contains an active substitution $\{^M/_x\}$ such that x is not under restriction. Every extended process A can be mapped to a frame $\varphi(A)$ by replacing every plain process in A with 0.

1.4 Operational Semantics

The operational semantics of processes in the applied pi-calculus is defined by three relations: *structural equivalence* ($\equiv$), *internal reduction* ($\rightarrow$) and *labelled reduction* ($\xrightarrow{\alpha}$).

Structural equivalence is defined in Figure 1.2. It is closed by α-conversion of both bound names and bound variables, and closed under application of evaluation contexts. The *internal reductions* and *labelled reductions* are defined in Figure 1.3. They are closed under structural equivalence and application of evaluation contexts. Internal reductions represent evaluation of condition and internal communication between processes. Labelled reductions represent communications with the environment.

1.5 Equivalences

Privacy properties are often stated as equivalence relations [DKR09]. Intuitively, if a protocol preserves ballot secrecy, an attacker should not be able to distinguish between a scenario where

$$\begin{array}{ll}
(\textsc{Comm}) & \overline{c}\langle M\rangle.P \mid c(x).Q \to P \mid Q\{^{M}/_{x}\} \\
(\textsc{Then}) & \text{if } \phi \text{ then } P \text{ else } Q \to P \quad \text{if } [\![\phi]\!] = \mathsf{true} \\
(\textsc{Else}) & \text{if } \phi \text{ then } P \text{ else } Q \to Q \quad \text{otherwise}
\end{array}$$

$$(\textsc{In}) \qquad c(x).P \xrightarrow{c(M)} P\{^{M}/_{x}\}$$

$$(\textsc{Out-Atom}) \qquad \overline{c}\langle u\rangle.P \xrightarrow{\overline{c}\langle u\rangle} P$$

$$(\textsc{Open-Atom}) \qquad \frac{A \xrightarrow{\overline{c}\langle u\rangle} A' \qquad u \neq c}{\nu u.A \xrightarrow{\nu u.\overline{c}\langle u\rangle} A'}$$

$$(\textsc{Scope}) \qquad \frac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}$$

$$(\textsc{Par}) \qquad \frac{A \xrightarrow{\alpha} A' \qquad \mathsf{bv}(\alpha) \cap \mathsf{fv}(B) = \mathsf{bn}(\alpha) \cap \mathsf{fn}(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

$$(\textsc{Struct}) \qquad \frac{A \equiv B \qquad B \xrightarrow{\alpha} B' \qquad B' \equiv A'}{A \xrightarrow{\alpha} A'}$$

where α is a *label* of the form $c(M)$, $\overline{c}\langle u\rangle$, or $\nu u.\overline{c}\langle u\rangle$ such that u is either a channel name or a variable of base type.

Figure 1.3: Semantics for processes

a voter votes 0 from a scenario where the voter votes 1. *Static equivalence* formally expresses indistinguishability of sequences of terms.

Definition 1.3 (Static equivalence)**.** *Two closed frames φ and ψ are statically equivalent, denoted $\varphi \approx_s \psi$, if $\mathsf{dom}(\varphi) = \mathsf{dom}(\psi)$ and there exists a set of names $\tilde{n}$ and substitutions σ, τ such that $\varphi \equiv \nu\tilde{n}.\sigma$ and $\psi \equiv \nu\tilde{n}.\tau$ and for all terms M, N such that $\tilde{n} \cap (\mathsf{fn}(M) \cup \mathsf{fn}(N)) = \emptyset$, we have $M\sigma =_E N\sigma$ holds if and only if $M\tau =_E N\tau$ holds.*

Two closed extended processes A, B are statically equivalent, written $A \approx_s B$, if their frames are statically equivalent; that is, $\varphi(A) \approx_s \varphi(B)$.

Intuitively, two sequences of messages φ and ψ are distinguishable to an attacker (*i.e.* they are not statically equivalent) if the attacker can build a public test $M = N$ that holds for φ but not for ψ (or the converse).

Example 3. *Consider the signature and equational theory E_{enc} defined in Example 1. Let $\varphi_1 = \nu k.\sigma_1$ and $\varphi_2 = \nu k.\sigma_2$ where $\sigma_1 = \{^{\mathsf{penc}(s_1,r_1,\mathsf{pk}(k))}/_{x_1}, {}^{\mathsf{pk}(k)}/_{x_2}\}$, $\sigma_2 = \{^{\mathsf{penc}(s_2,r_2,\mathsf{pk}(k))}/_{x_1}, {}^{\mathsf{pk}(k)}/_{x_2}\}$ and s_1, s_2, k are names. We have that $\varphi_1 \not\approx_s \varphi_2$. Indeed, we $\mathsf{penc}(s_1, r_1, x_2)\sigma_1 =_E x_1\sigma_1$ but $\mathsf{penc}(s_1, r_1, x_2)\sigma_2 \neq_E x_1\sigma_2$.*

Intuitively, since the randomness of the encryption is public, an attacker may reconstruct the ciphertexts and compare. The two messages become indistinguishable as soon as the randomness remain private. That is, we have that $\nu(k, r_1).\sigma_1 \approx_s \nu(k, r_2).\sigma_2$.

Observational equivalence is the active counterpart of static equivalence, where the attacker can actively interact with the processes. The definition of observational equivalence requires

to reason about all contexts (*i.e.* all adversaries), which renders the proofs difficult. Since observational equivalence has been shown to coincide [AF01, Liu11] with labelled bisimilarity, we adopt the latter in this part.

Definition 1.4 (Labelled bisimilarity)**.** *Labelled bisimilarity ($\approx_l$) is the largest symmetric relation $\mathcal{R}$ on closed extended processes such that $A\mathcal{R}B$ implies:*

1. $A \approx_s B$;
2. *if* $A \to A'$, *then* $B \to^* B'$ *and* $A'\mathcal{R}B'$ *for some* B';
3. *if* $A \xrightarrow{\alpha} A'$ *such that* $\mathsf{fv}(\alpha) \subseteq \mathsf{dom}(A)$ *and* $\mathsf{bn}(\alpha) \cap \mathsf{fn}(B) = \emptyset$, *then* $B \to^* \xrightarrow{\alpha} \to^* B'$ *and* $A'\mathcal{R}B'$ *for some* B'.

Intuitively, two processes A and B are labelled bisimilar if, anyhow the process A (resp. B) behaves, the process B (resp. A) may behave with the same visible actions and such that their resulting frames are statically equivalent, that is, an attacker cannot distinguish between them.

1.6 A Detailed Example

We provide a detailed example to illustrate the syntax and semantics of the applied pi-calculus. Readers already familiar with this formalism may skip this section. Let us consider a simple protocol proposed (for illustration purposes) by B. Blanchet [Bla08].

$$\text{Alice} \quad \xrightarrow{\{\mathsf{sign}(k,\mathsf{sk_A})\}_{\mathsf{pk_B}}} \quad \xleftarrow{\{s\}_k} \quad \text{Bob}$$

In this protocol, Alice sends a newly generated and signed key k to Bob, the whole message encrypted using his public key $\mathsf{pk_B}$. Then, the receiver (Bob) is able to check that the signature belongs to the intended sender (Alice). He then encrypts a fresh secret s with k using symmetric encryption and sends it to Alice.

To model this protocol in applied-pi calculus, we use the equational theory E_{enc} described in Example 1 that models the properties of the different primitives used in the protocol. The behavior of the sender and receiver are modeled in the applied pi-calculus as follows.

$$\begin{aligned}
A &:= \nu k \,.\, \nu r_a \,.\, \overline{c}\langle \mathsf{penc}(\mathsf{sign}(k, \mathsf{sk}(a)), r_a, \mathsf{pk_B})\rangle \,.\, c(x_1) \\
B(s) &:= c(x_2) \,.\, B'(s) \\
B'(s) &:= \text{let } y = \mathsf{pdec}(x_2, \mathsf{sk}(b)) \text{ in} \\
&\qquad \text{if } \mathsf{checksign}(y, \mathsf{pk_A}) = \mathsf{ok} \text{ then } B''(s) \\
B''(s) &:= \nu r_b \,.\, \overline{c}\langle \mathsf{enc}(s, r_b, k)\rangle
\end{aligned}$$

In these processes, a and b are secret values that correspond to the private key pair of Alice and Bob, respectively. Since Alice should not have a direct access to b, the secret key of Bob, A is given access to Bob's public key through a variable $\mathsf{pk_B}$ and similarly for B. The whole protocol can be easily expressed using Alice and Bob processes:

$$P(s) := \nu(a, b) \,.\, [A \mid B(s) \mid \Gamma]$$

where $\Gamma = \left\{ {}^{\mathsf{pk}(a)}/_{\mathsf{pk_B}}, {}^{\mathsf{pk}(b)}/_{\mathsf{pk_B}} \right\}$. As mentioned above, a and b are protected but since $\mathsf{pk}(a)$ and $\mathsf{pk}(b)$ are public keys, we published them in a frame implying that they are available to anyone

(including the attacker). We first illustrate the semantics of the internal reductions by simulating the normal execution of the protocol, without any interference.

$$\begin{aligned} P(s) &\xrightarrow{\text{(COMM)}} \nu(a,b,k,r_a).\left[c(x_1) \mid B'(s) \mid \left\{{}^{\mathsf{penc}(\mathsf{sign}(k,\mathsf{sk}(a)),r_a,\mathsf{pk_B})}/_{x_2}\right\} \mid \Gamma\right] \\ &\xrightarrow{\text{(THEN)}} \nu(a,b,k,r_a).\left[c(x_1) \mid B''(s) \mid \left\{{}^{\mathsf{penc}(\mathsf{sign}(k,\mathsf{sk}(a)),r_a,\mathsf{pk_B})}/_{x_2}\right\} \mid \Gamma\right] \\ &\xrightarrow{\text{(COMM)}} \nu\tilde{n}.\left[\left\{{}^{\mathsf{enc}(s,r_b,k)}/_{x_1}, {}^{\mathsf{penc}(\mathsf{sign}(k,\mathsf{sk}(a)),r_a,\mathsf{pk_B})}/_{x_2}\right\} \mid \Gamma\right], \text{ with } \tilde{n} = (a,b,k,r_a,r_b). \end{aligned}$$

Actually, this simple protocol is flawed. Indeed, an intruder may impersonate Alice's identity from Bob's point of view. This attack only requires that Alice had, once in the past, spoken to the intruder using this protocol. This can be showed using our applied-pi calculus model by adding $\mathsf{penc}(\mathsf{sign}(k_c,\mathsf{sk}(a)),r_c,\mathsf{pk_C})$ to the initial frame, which is a message that Alice would have sent to Charlie according to the protocol. Let us define this slightly new process:

$$P_c(s) := \nu(a,b,r_c,k_c)\ .[A \mid B(s) \mid \Gamma_c]\,,$$

where $\Gamma_c = \left\{{}^{\mathsf{penc}(\mathsf{sign}(k_c,\mathsf{sk}(a)),r_c,\mathsf{pk_C})}/_{x}, {}^{\mathsf{pk}(a)}/_{\mathsf{pk_A}}, {}^{\mathsf{pk}(b)}/_{\mathsf{pk_B}}, {}^{\mathsf{pk}(c)}/_{\mathsf{pk_C}}\right\}$. Let us consider now the following execution: Charlie can send a message $M = \mathsf{penc}(\mathsf{pdec}(x,\mathsf{sk}(c)),r',\mathsf{pk_B})$ to Bob, where x is the message previously received from Alice.

$$\begin{aligned} P_c(s) &\xrightarrow{c(M)} \nu(a,b,r_c,k_c).\left[A \mid B' \mid \Gamma_c\right] \\ &\xrightarrow{\text{(THEN)}} \nu(a,b,r_c,k_c).\left[A \mid B'' \mid \Gamma_c\right] \\ &\xrightarrow{\nu r_b.\bar{c}\langle \mathsf{enc}(s,r_b,k_c)\rangle} \nu(a,b,r_c,k_c,r_b).\left[A \mid \left\{{}^{\mathsf{enc}(s,r_b,k_c)}/_{x_c}\right\} \mid \Gamma_c\right]. \end{aligned}$$

Let us define the substitution $\sigma = \left\{{}^{\mathsf{enc}(s,r_b,k_c)}/_{x_c}, {}^{M}/_{x_2}\right\} \mid \Gamma_c$. In this execution, the test performed by B succeeds since, according to the equation theory, we have:

$$\begin{aligned} M\sigma &= \mathsf{penc}(\mathsf{pdec}(x,\mathsf{sk}(c)),r',\mathsf{pk_B})\sigma \\ &= \mathsf{penc}(\mathsf{pdec}(\mathsf{penc}(\mathsf{sign}(k_c,\mathsf{sk}(a)),r_c,\mathsf{pk}(c)),\mathsf{sk}(c)),r',\mathsf{pk}(b)) \\ &=_{E_{\mathsf{enc}}} \mathsf{penc}(\mathsf{sign}(k_c,\mathsf{sk}(a)),r',\mathsf{pk}(b)), \end{aligned}$$

thus,

$$\begin{aligned} &\mathsf{checksign}(\mathsf{pdec}(x_2,\mathsf{sk}(b)),\mathsf{pk_A})\sigma \\ &\quad =_{E_{\mathsf{enc}}} \mathsf{checksign}(\mathsf{pdec}(\mathsf{penc}(\mathsf{sign}(k_c,\mathsf{sk}(a)),r',\mathsf{pk}(b)),\mathsf{sk}(b)),\mathsf{pk}(a)) \\ &\quad =_{E_{\mathsf{enc}}} \mathsf{checksign}(\mathsf{sign}(k_c,\mathsf{sk}(a)),\mathsf{pk}(a)) \\ &\quad =_{E_{\mathsf{enc}}} \mathsf{ok} \end{aligned}$$

which implies that the message is accepted by B, who therefore believes that Alice just sent him the key k_c she had, in fact, sent to Charlie. At the end of this execution, one can see that s is not secret anymore since Charlie knows k_c from his previous exchange and can perform a decryption of x_c to get s.

We can also illustrate Definition 1.4 using this example. In fact, using this attack, we have that for any s_1 and s_2, $P(s_1) \not\approx_l P(s_2)$. This is due to the fact that the two processes may evolve in two states that are not statically equivalent ($\approx_s$):

$$P(s_1) \to^* P_1 = \nu(a,b,r_b).[A \mid \sigma_1] \text{ and } P(s_2) \to^* P_2 = \nu(a,b,r_b).[A \mid \sigma_2]$$

where σ_i is σ where s is simply replaced by s_i. We can show that $P_1 \not\approx_s P_2$, due to the fact that the intruder can, using $N_1 = \mathsf{dec}(x_c,k_c)$ and $N_2 = s_1$, see that $(N_1 =_E N_2)\sigma_1$ but $(N_1 \neq_E N_2)\sigma_2$.

Chapter 2

Norwegian E-Voting Protocol

Contents

Used in September 2011 and September 2013 for municipality and county elections in Norway [Gov], E-voting was tested in ten municipalities. During this nationwide local elections, more than 28,000 voters did use internet to actually cast their vote in 2011 and 70,000 in 2013. While many countries use black-box proprietary solutions, Norway made the protocol publicly available [Gjø10] and, in [Gjø10], Gjøsteen describes the protocol and discusses its security but without security proof. Based on the applied pi-calculus model [AF01], this chapter details a formal analysis of the e-voting protocol used during these trials. Several other e-voting protocols have been studying using formal methods. The FOO [FOO92], Okamoto [OMA+99] and Lee *et al.* [LBD+04] voting protocols have been analysed in [DKR09]. Similarly, Helios has been recently proved secure both in a formal [CS13] and a computational [BCP+11, BPW12] model.

Helios is actually an implementation of a voting system proposed and analyzed (for the available definitions at that time) by Cramer *et al* [CGS97]. All these protocols were significantly simpler to analyse symbolically due to the fact that the cryptographic primitives were easier to abstract as a term algebra and due to the fact that these protocols involve fewer steps. The more complex Civitas protocol was analyzed in [KT09] but Civitas is not yet deployed. In contrast, the Norwegian protocol is both complex and fully deployed.

First, we provide an informal description of the protocol in Section 2.1, including details about the different phases of the voting process. Then, we propose, in Section 2.2, a formal model of this protocol using applied pi-calculus. Since the protocol also makes use of signature, zero-knowledge proofs, blinding functions, and coding functions, we have therefore proposed a new equational theory reflecting the unusual behavior of the primitives. In Section 2.4, we gather lemmas we needed for the different proofs and we think are independent of interest in the sense they can be useful for further formal studies of different protocols. The main results and corresponding proofs are presented towards Section 2.3 while some results, obtained using ProVerif, are described in Section 2.7. The lemmas used in the different proofs that are not displayed in this chapter are provided in Appendix A.

2.1 The Norwegian E-Voting Protocol

The Norwegian protocol features four players representing the electronic poll's infrastructure: a Ballot box (B), a Receipt generator (R), a Decryption service (D) and an Auditor (A). Each voter (V) can log in using a computer (P) in order to submit his vote. Channels between computers (voters) and the Ballot box are considered as authenticated channels, channels between infrastructure's player are untappable, and channels between voters and receipt generator are unidirectional out-of-band hannels. (Example of SMS is given in [Gjø10].) The protocol can be divided in three phases: the setting phase, the submission phase, where voters submit their votes, and the counting phase, where ballots are counted and the auditor verifies the correctness of the election.

2.1.1 Setting Phase

Before the election, private keys a_1, a_2 and a_3 (such that $\mathsf{a}_1 + \mathsf{a}_2 = \mathsf{a}_3$) are distributed over respectively D, B, and R, while the corresponding public keys $\mathsf{y}_1 = \mathsf{g}^{\mathsf{a}_1}$, $\mathsf{y}_2 = \mathsf{g}^{\mathsf{a}_2}$ and $\mathsf{y}_3 = \mathsf{g}^{\mathsf{a}_3}$ are made publicly available. The Receipt generator R is assumed to have a signing key id_R which corresponding verification key is public. Each voter is also assumed to have a signing key id_V which corresponding verification key is public too. The Ballot box B is provided with a table $V \mapsto \mathsf{s}_\mathsf{V}$ with a blinding factor s_V for each voter V. The Receipt generator R is given a table $V \mapsto d_V$ with a permutation function d_V for each voter V. Finally, each voter V is assumed to have received by surface mail a table where, for each voting option o, corresponds a precomputed receipt code $d_V(\mathsf{f}(\mathsf{o})^{\mathsf{s}_\mathsf{V}})$ with f some encoding function for voting options.

2.1.2 Submission Phase

The submission phase is summarized in Figure 2.1. We detail in this section the expected behavior of each participant when they are supposed honest.

Voter (V). Each voter tells his computer what voting option o to submit and allows it to sign the corresponding ballot on his behalf. Then, he has to wait for an acceptance message

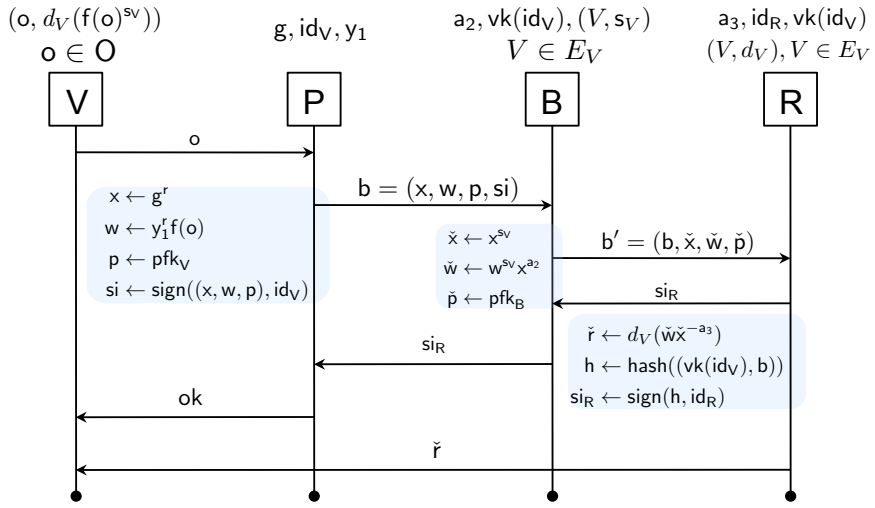


Figure 2.1: Submission of one vote.

coming from the computer and a receipt $\check{r}$ sent by the receipt generator through the out-of-band channel. Using the receipt, he verifies that the correct vote was submitted, that is, he checks that $\check{r} = d_V(f(o)^{s_V})$ by verifying that the receipt code $\check{r}$ indeed appears in the line associated to the voting option o he has chosen.

Computer (P). Voter's computer encrypts voter's ballot with the public key y_1 using standard El Gamal encryption. The resulting ballot is $\langle g^r, y_1^r f(o)\rangle$. P also proves that the resulting ciphertext corresponds to a valid voting option, by computing a standard proof of knowledge pfk_V. How pfk_V is computed exactly can be found in [Gjø10]. P also signs the ballot with id_V and sends it to the Ballot box. It then waits for a confirmation si_R coming from the latter, which is a hash of the initial encrypted ballot, signed by the Receipt generator. After checking this signature, the computer notifies the voter that his vote has been taken into account.

Ballot box (B). Upon receiving an encrypted and signed ballot b from a computer, the Ballot box first checks the correctness of signatures and proofs before re-encrypting the original encrypted ballot with a_2 and blinding it with s_V. B also generates a proof pfk_B, showing that its computation is correct. B then sends the new modified ballot b' to the Receipt generator. Once the Ballot box receives a message si_R from R, it simply checks that the Receipt generator's signature is valid, and sends it to the computer.

Receipt generator (R). Upon receiving an encrypted ballot $b' = \langle b, \tilde{x}, \tilde{w}, \tilde{p}\rangle$ from the Ballot box, the Receipt generator first checks signature and proofs (from the computer and the Ballot box). If the validity checks are successful, it generates:

- a receipt code $\check{r} = d_V(\tilde{w}\tilde{x}^{-a_3})$ sent by out-of-band channel directly to the voter. Intuitively, the Receipt generator decrypts the (blinded) ballot, applying the permutation function d_V associated to the voter. This gives assurance to the voter that the correct vote was submitted to the Ballot box.

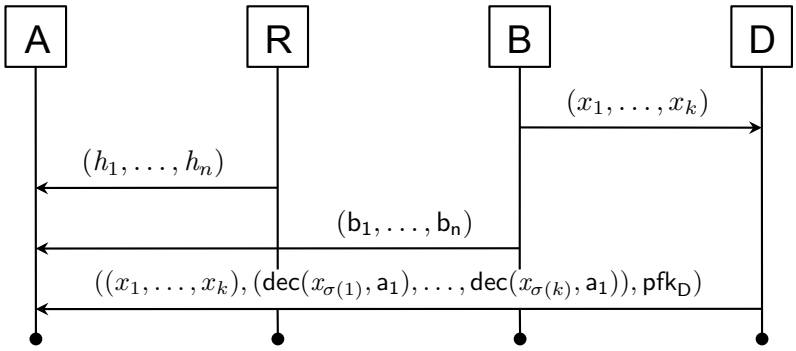


Figure 2.2: Counting phase.

- a signature on a hash of the original encrypted ballot for the Ballot box. Once transmitted by B, it allows the computer to inform the voter that his vote has been accepted.

2.1.3 Counting Phase

Once the voting phase is over, the counting phase begins (Figure 2.2). The Ballot box selects the encrypted votes $x_1, \ldots, x_k$ which need to be decrypted (if a voter has voted several times, all the submitted ballots remain in the memory of the Ballot box but only the last ballot should be sent) and sends them to the Decryption service. The whole content of the Ballot box $\mathsf{b_1}, \ldots, \mathsf{b_n}$ $(n \geq k)$ is revealed to the Auditor, including previous votes from re-voting voters. The Receipt generator sends to the Auditor the list of hashes of ballots it has seen during the submission phase. The Decryption service decrypts the incoming ciphertexts $x_1, \ldots, x_k$ received from the Ballot box and shuffles the decrypted votes before publishing them. It therefore outputs a message of the form $\mathsf{dec}(x_{\sigma(1)}, \mathsf{a_1}), \ldots, \mathsf{dec}(x_{\sigma(k)}, \mathsf{a_1})$ where σ denotes the permutation obtained by shuffling the votes. It also provides the Auditor with a proof $\mathsf{pfk_D}$ showing that the input ciphertexts and the outcoming decryption indeed match. Using the Ballot box content and the list of hashes from the Receipt generator, the Auditor verifies that no ballots have been inserted or lost and it computes its own list of encrypted ballots which should be counted. He compares this list with the one received from the Decryption service and checks the proof provided by the latter.

2.2 Modeling the Norwegian Protocol

We provide a formal specification of the Norwegian protocol, using the framework of the applied-pi calculus, defined in the previous chapter. We first model the cryptographic primitives used in the protocol (Section 2.2.1) and then the Norwegian protocol itself (Section 2.2.2).

2.2.1 Equational Theory

We adopt the following signature to capture the cryptographic primitives used by the protocol.

$$\begin{aligned}\Sigma_{\mathsf{sign}} = \{&\mathsf{ok}, \mathsf{fst}, \mathsf{hash}, \mathsf{p}, \mathsf{pk}, \mathsf{s}, \mathsf{snd}, \mathsf{vk}, \mathsf{blind}, \mathsf{d}, \mathsf{dec}, +, *, \circ, \diamond, \mathsf{pair},\\ &\mathsf{renc}, \mathsf{sign}, \mathsf{unblind}, \mathsf{checkpfk_1}, \mathsf{checkpfk_2}, \mathsf{checksign}, \mathsf{penc}, \mathsf{pfk_1}, \mathsf{pfk_2}\}\end{aligned}$$

$$\mathsf{fst}(\mathsf{pair}(x,y)) = x \tag{E-1}$$
$$\mathsf{snd}(\mathsf{pair}(x,y)) = y \tag{E-2}$$
$$\mathsf{dec}(\mathsf{penc}(x,r,\mathsf{pk}(k)),k) = x \tag{E-3}$$
$$\mathsf{dec}(\mathsf{blind}(\mathsf{penc}(x,r,\mathsf{pk}(k)),b),k) = \mathsf{blind}(x,b) \tag{E-4}$$
$$\mathsf{penc}(x,r_1,k_p) \circ \mathsf{penc}(y,r_2,k_p) = \mathsf{penc}(x \diamond y, r_1 * r_2, k_p) \tag{E-5}$$
$$\mathsf{renc}(\mathsf{penc}(x,r,\mathsf{pk}(k_1)),k_2) = \mathsf{penc}(x,r,\mathsf{pk}(k_1+k_2)) \tag{E-6}$$
$$\mathsf{unblind}(\mathsf{blind}(x,b),b) = x \tag{E-7}$$
$$\mathsf{checksign}(x,\mathsf{vk}(id),\mathsf{sign}(x,id)) = \mathsf{ok} \tag{E-8}$$
$$\mathsf{checkpfk}_1(\mathsf{vk}(id),\mathsf{ball},\mathsf{pfk}_1(id,r,x,\mathsf{ball})) = \mathsf{ok} \tag{E-9}$$
$$\text{where } \mathsf{ball} = \mathsf{penc}(x,r,k_p)$$
$$\mathsf{checkpfk}_2(\mathsf{vk}(id),x,\mathsf{ball},\mathsf{pfk}_2(\mathsf{vk}(id),k,b,x,\mathsf{ball})) = \mathsf{ok} \tag{E-10}$$
$$\text{where } \mathsf{ball} = \mathsf{blind}(\mathsf{renc}(x,k),b)$$

Figure 2.3: Equations for encryption, blinding, signature and proof of knowledge.

The function ok is a constant; fst, hash, p, pk, s, snd, vk are unary functions; blind, d, dec, $+$, $*$, $\circ$, $\diamond$, pair, renc, sign, $\mathsf{unblind}$ are binary functions; $\mathsf{checkpfk}_1$, $\mathsf{checksign}$, penc are ternary functions; pfk_1, $\mathsf{checkpfk}_2$ are quaternary functions and pfk_2 is a quinary function.

The term $\mathsf{pk}(K)$ denotes the public key corresponding to the secret key K in asymmetric encryption. Terms $\mathsf{s}(I)$, $\mathsf{p}(I)$, and $\mathsf{vk}(I)$ are respectively the blinding factor, the parameter that will help to consider a specific function d for each voter, and the verification key associated to a secret id I. The specific coding function used by the receipt generator for a voter with secret id I, applied to a message M is represented by $\mathsf{d}(\mathsf{p}(I),M)$. It corresponds to the function $d_V(M)$ explained in Section 2.1.2. The term $\mathsf{blind}(M,N)$ represents the message M blinded by N. Unblinding such a blinded term P, using the same blinding factor N is denoted by $\mathsf{unblind}(P,N)$. The term $\mathsf{penc}(M,N,P)$ refers to the encryption of plaintext M using randomness N and public key P. The term $M \circ N$ denotes the homomorphic combination of ciphertexts M and M' and the corresponding operation on plaintexts is written $P \diamond Q$ and $R * S$ on random factors. The decryption of ciphertext C using secret key K is denoted $\mathsf{dec}(C,K)$. The term $\mathsf{renc}(M,K)$ is the re-encryption of the ciphertext M using a secret key K. The addition of secret keys is denoted by $K+L$. The term $\mathsf{sign}(M,N)$ refers to the signature of the message M using secret id N. The term $\mathsf{pfk}_1(M,N,P,Q)$ represents a proof of knowledge that proves that Q is a ciphertext of the plaintext P using randomness N. The term $\mathsf{pfk}_2(M,N,P,Q,R)$ denotes a proof of knowledge that R is a blinding of a re-encryption of a term Q using the secret key N and the randomness P. $\mathsf{pair}(M,N)$ represents the tuple (M,N). For convenience, $\mathsf{pair}(M_1,\mathsf{pair}(\ldots,\mathsf{pair}(M_{n-1},M_n)))$ is abbreviated as $\langle M_1,\ldots,M_n\rangle$ and $\mathsf{fst}(\mathsf{snd}(M)^{i-1})$ is denoted $\Pi_i(M)$ with $i \in \mathbb{N}$.

The properties of the primitives are then modeled by equipping the signature with an equational theory E that asserts that functions $+$, $*$, $\circ$ and $\diamond$ are commutative and associative, and includes the equations defined in Figure 2.3.

The three first equations are quite standard. Equation (E-4) reflects that a blinded ciphertext can be decrypted, yielding the corresponding blinded plaintext. Equation (E-5) models the homomorphic combination of ciphertexts. Equation (E-6) represents the re-encryption of a ciphertext. The operation of unblinding is described with Equation (E-7). Equations (E-8), (E-9),

and (E-10) correspond to the verification of respectively signature and proofs of knowledge pfk_1 and to pfk_2.

The rewriting system corresponding to this equational theory is AC-convergent. This can be proved showing that the system is both AC-confluent and AC-terminating. The first property is true since there is no critical pairs. AC-termination can be shown through a special measure for length of terms $|.|$ defined as follows:

$$|M| = \begin{cases} 1 & \text{, if } M \text{ is a name or a variable,} \\ 2 + |M_1| + |M_2| + |M_3| & \text{, if } M = \mathsf{penc}(M_1, M_2, M_3), \\ 2 + |M_1| + |M_2| & \text{, if } M = \mathsf{renc}(M_1, M_2), \\ 1 + \sum_{i=1}^{k} |M_i| & \text{, otherwise, i.e. } M = f(M_1, \ldots, M_k). \end{cases}$$

Using that measure, it is easy to check that the length of terms is strictly decreasing at each step of the rewriting, which ensures AC-termination of the rewriting system.

2.2.2 Norwegian Protocol Process Specification

We chose not to model re-voting for simplicity and also since it is explicitly and strongly discouraged in [Gjø10], as it may allow an attacker to swap two votes (the initial casted one and its revoted one).

Voting process

The process $V(c^b_{auth}, c^r_{auth}, c_p, k_p, id_s, id_p, v)$ represents both the voter and his computer.

$V(c^b_{auth}, c^r_{auth}, c_p, k_p, id_s, id_p, v) = \nu r$.
let $e = \mathsf{penc}(v, r, k_p)$, $p = \mathsf{pfk}_1(id_s, t, v, e)$, $si = \mathsf{sign}(\langle e, p\rangle, id_s)$ in
$\overline{c_p}\langle\langle \mathsf{vk}(id_s), e, p, si\rangle\rangle$. % Public information.
$\overline{c^b_{auth}}\langle\langle \mathsf{vk}(id_s), e, p, si\rangle\rangle$. % Encrypted ballot sent to B.
$\overline{c^r_{auth}}\langle \mathsf{ok}\rangle$. % Synchronization for R.
$c^r_{auth}(x_r)$. $c^b_{auth}(x_b)$. % Wait for inputs from R and B.
if $\phi_{\mathsf{V}}(id_s, id_p, v, e, p, si, x_b, x_r)$ then $\overline{c_p}\langle \mathsf{ok}\rangle$. $\overline{c^r_{auth}}\langle \mathsf{ok}\rangle$

with:

$$\phi_{\mathsf{V}}(id_s, id_p, v, e, p, si, x_b, x_r) = (\mathsf{d}(\mathsf{p}(id_s), \mathsf{blind}(v, \mathsf{s}(id_s))) = x_r) \\ \wedge (\mathsf{checksign}(\mathsf{hash}(\langle \mathsf{vk}(id_s), e, p, si\rangle), id_p, x_b) = \mathsf{ok}).$$

Parameter v represents voter's vote and c^b_{auth}, c^r_{auth} denote the authenticated channels shared with, respectively, the ballot box and the receipt generator. k_p represents the public key of the election; id_s is the secret id of the voter and id_p is the verification key of the receipt generator. Note that messages sent over c^b_{auth} and c^r_{auth} are also sent on the public channel c_p. This simulates the fact that c^b_{auth} and c^r_{auth} are authenticated but not confidential channels. The synchronization step is only here to simplify the study in the case where B is corrupted.

The formula $\phi_{\mathsf{V}}(id_s, id_p, v, e, p, si, x_b, x_r)$ models all the checks performed by the voters: the message received from the ballot box should be properly signed and the message from the out-of-band channel should correspond to the right receipt code.

Ballot box

We represents by the process $B_n(c_{br}, c_{bd}, c_{ba}, c_p, k_s, id_p, c_b^1, id_v^1, s_v^1, \ldots, c_b^n, id_v^n, s_v^n)$ the program of a Ballot box ready to listen to n voters.

$B_n(c_{br}, c_{bd}, c_{ba}, c_p, k_s, id_p, c_b^1, id_v^1, s_v^1, \ldots, c_b^n, id_v^n, s_v^n) =$
$\ldots \,.\, c_b^i(x_i)\,.$
if $\phi_{\mathsf{B}}(id_v^i, x_i)$ then — % Checks ballot's validity.
let $b_i = \mathsf{blind}(\mathsf{renc}(\Pi_2(x_i), k_s), s_v^i)$ in — % Computes re-encrypted blinded
let $pfk_i = \mathsf{pfk}_2(id_v^i, k_s, s_v^i, \Pi_2(x_i), b_i)$ in — ballot and corresponding proof.
$\overline{c_{br}}\langle\langle x_i, b_i, pfk_i\rangle\rangle \,.\, c_{br}(y_i)\,.$ — % Message sent to R. Wait for R.
if $\phi_{\mathsf{S}}(id_p, x_i, y_i)$ then — % Checks confirmation's validity.
$\overline{c_p}\langle y_i\rangle \,.\, \overline{c_b^i}\langle y_i\rangle\,.$ — % Sends confirmation to voter.
$\ldots$
$\overline{c_p}\langle y_n\rangle \,.\, \overline{c_v^n}\langle y_n\rangle\,.$
$\overline{c_{bd}}\langle\Pi_2(x_1)\rangle \,.\, \ldots \,.\, \overline{c_{bd}}\langle\Pi_2(x_n)\rangle\,.$ — % Outputs encrypted votes to D.
$\overline{c_{ba}}\langle x_1\rangle \,.\, \ldots \,.\, \overline{c_{ba}}\langle x_n\rangle$ — % Outputs content to A.

with:

$$\phi_{\mathsf{S}}(id_p, x, y) = (\mathsf{checksign}(\mathsf{hash}(x), id_p, y) = \mathsf{ok})\,,$$
$$\phi_{\mathsf{B}}(id_v, x) = (x = \langle x_1, x_2, x_3, x_4\rangle) \wedge (\mathsf{checkpfk}_1(x_1, x_2, x_3) = \mathsf{ok}) \wedge (x_1 = id_v) \wedge (\mathsf{checksign}(\langle x_2, x_3\rangle, x_1, x_4) = \mathsf{ok})$$

where $(x = \langle x_1, \ldots, x_n\rangle)$ denotes the formula that holds only when x is a n-tuple. For example, $(x = \langle x_1, x_2\rangle)$ denotes the formula $x = \mathsf{pair}(\mathsf{fst}(x), \mathsf{snd}(x))$.

Intuitively, we assume the ballots to be received from the authenticated channels $c_b^1, \ldots, c_b^n$. Then the Ballot box sends messages to resp. the Receipt generator, the Decryption service, and the Auditor through secure channels c_{br}, c_{bd} and c_{ba}. k_s is the secret key known by B, $id_v^1, \ldots, id_v^n$ are the public ids of voters (*i.e.* their verification keys) and $s_v^1, \ldots, s_v^n$ the corresponding blinding factors.

Receipt generator

$R_n(c_{br}, c_{ra}, c_{rd}, c_p, id, k_s, c_v^1, id_v^1, p_v^1, \ldots, c_v^n, id_v^n, p_v^n)$ is the Receipt generator's process. It exchanges messages with the Ballot box, the Decryption service (only for an ad-hoc synchronization) and the Auditor through secure channels c_{br}, c_{rd}, and c_{ra} respectively. It also talks directly to voters through out-of-band channels $c_v^1, \ldots, c_v^n$. k_s is its secret key, $id_v^1, \ldots, id_v^n$ are the public ids of the voters and the corresponding receipt coding functions are $p_v^1, \ldots, p_v^n$.

$R_n(c_{br}, c_{ra}, c_{rd}, c_p, id, k_s, c_v^1, id_v^1, p_v^1, \ldots, c_v^n, id_v^n, p_v^n) =$
$\ldots \,.\, c_v^i(go_i^r) \,.\, c_{rb}(x_i)\,.$
if $\phi_{\mathsf{R}}(id_v^i, x_i)$ then — % Checks B's computations.
let $r_i = \mathsf{d}(p_v^i, \mathsf{dec}(\Pi_2(x_i), k_s))$ in — % Computes receipt for V.
let $hbr_i = \mathsf{hash}(\Pi_1(x_i))$ in
let $si_i = \mathsf{sign}(hbr_i, id)$ in — % Computes confirmations for B.
$\overline{c_{rb}}\langle si_i\rangle \,.\, \overline{c_p}\langle r_i\rangle \,.\, \overline{c_v^i}\langle r_i\rangle \,.\, c_v^i(sy_i)\,.$
$\ldots$

$\overline{c_{rb}}\langle si_n\rangle \,.\, \overline{c_p}\langle r_n\rangle \,.\, \overline{c_v^n}\langle r_n\rangle \,.\, c_v^n(sy_n)\,.$
$\overline{c_{ra}}\langle\langle id_v^1, hbr_1\rangle\rangle \,.\, \ldots \,.\, \overline{c_{ra}}\langle\langle id_v^n, hbr_n\rangle\rangle \,.$ % Outputs content to A.
$\overline{c_{rd}}\langle \mathsf{ok}\rangle$ % Ad-hoc synchronization for D.

with:

$$\begin{aligned}\phi_{\mathsf{R}}(id_v, x) = (x = \langle x_1, x_2, x_3\rangle) \wedge (x_1 = \langle y_1, y_2, y_3, y_4\rangle) \wedge (y_1 = id_v)\\ \wedge (\mathsf{checksign}(\langle y_2, y_3\rangle, y_1, y_4)) \wedge (\mathsf{checkpfk}_1(y_1, y_2, y_3))\\ \wedge (\mathsf{checkpfk}_2(y_1, y_2, x_2, x_3))\,.\end{aligned}$$

Decryption service

The Decryption service is represented by the process $D_n(c_{bd}, c_{rd}, c_{da}, c_p, k_s)$. It communicates securely with the Ballot box, the Receipt generator (waiting for synchronization), and the Auditor through respectively channels c_{bd}, c_{rd}, and c_{da}. The result is published on the public channel c_p. In order to decrypt ballots, it needs to know the secret key k_s. The parallelism at the end of the process models that the votes are shuffled.

$D_n(c_{bd}, c_{rd}, c_{da}, c_p, k_s) =$
$c_{rd}(go_1)\,.$ % Waits for R's signal.
$c_{bd}(d_1) \,.\, \ldots \,.\, c_{bd}(d_n)\,.$ % Inputs from B.
$\overline{c_{da}}\langle \mathsf{hash}(\langle d_1, \ldots, d_n\rangle)\rangle \,.\, c_{da}(go_2)\,.$ % Outputs for A and waits.
$(\overline{c_p}\langle \mathsf{dec}(d_1, a_1)\rangle \mid \cdots \mid \overline{c_p}\langle \mathsf{dec}(d_n, a_n)\rangle)$ % Publishes the results.

Auditor

Finally, the Auditor is modeled by the process $A_n(c_{ba}, c_{ra}, c_{da})$ which communicates with the other infrastructure players using secure channels c_{ba}, c_{ra}, and c_{da}.

$A_n(c_{ba}, c_{ra}, c_{da}) =$
$c_{ra}(h_1) \,.\, \ldots \,.\, c_{ra}(h_n)\,.$ % Inputs from R, D and B.
$c_{da}(h_d) \,.\, c_{ba}(x_1) \,.\, \ldots \,.\, c_{ba}(x_n)\,.$
if $\phi_{\mathsf{A}}(h_d, h_1, \ldots, h_n, x_1, \ldots, x_n)$ then $\overline{c_{da}}\langle \mathsf{ok}\rangle$ else 0 % Checks and sends approval.

with:

$$\begin{aligned}\phi_{\mathsf{A}}(h, h_1, \ldots, h_n, x_1, \ldots, x_n) = (\mathsf{hash}(\langle \Pi_2(x_1), \ldots, \Pi_2(x_n)\rangle) = h)\\ \bigwedge_{i=1}^{n} \Big[(x_i = \langle y_1, y_2, y_3, y_4\rangle) \wedge (h_i = \langle z_1, z_2\rangle) \wedge (y_1 = z_1)\\ \wedge (\mathsf{hash}(x_i) = z_2) \wedge (\mathsf{checksign}(\langle y_2, y_3\rangle, y_1, y_4) = \mathsf{ok}) \Big]\,.\end{aligned}$$

Norwegian protocol and corruption scenarios

The interaction of all the players is simply modeled by considering all the processes in parallel, with the correct instantiation and restriction of the parameters. In what follows, the restricted names a_1, a_2, a_3 model the private keys used in the protocol and the corresponding public keys $\mathsf{pk}(\mathsf{a}_1)$, $\mathsf{pk}(\mathsf{a}_2)$ and $\mathsf{pk}(\mathsf{a}_3)$ are added in the process frame. The restricted names c_1, c_2 (resp. $\mathsf{c}_{\mathsf{RV}_1}$ and $\mathsf{c}_{\mathsf{RV}_2}$) model authentic channels between the two honest voters and the Ballot box (resp. the Receipt generator). The restricted names id_1, id_2, id_{R} represent the secret ids of honest voters and of the Receipt generator. The corresponding public id's are added to the process frame.

Note. We consider that the two honest voters are the first ones. Enforcing the order is not a limitation. Indeed, it provides more information to the attacker without limiting his possibilities.

The process corresponding to the situation where all the authorities are honest is $P_n\,[\_]$, where n is the number of voters and the hole is the voter place, and is defined as follows:

$$\begin{aligned} P_n\,[\_] = \nu\tilde{n}.(\ \text{let}\ \mathsf{a_3} = \mathsf{a_1} + \mathsf{a_2}\ \text{in}\ ).\Big[\ \_ \\ \mid B_n(\mathsf{c_{BR}}, \mathsf{c_{BD}}, \mathsf{c_{BA}}, \mathsf{c_{out}}, \mathsf{a_2}, \mathsf{idp_R}, \mathsf{c_1}, \mathsf{idp_1}, \mathsf{s(id_1)}, \ldots, \mathsf{c_n}, \mathsf{idp_n}, \mathsf{s(id_n)}) \\ \mid R_n(\mathsf{c_{BR}}, \mathsf{c_{RA}}, \mathsf{c_{RD}}, \mathsf{c_{out}}, \mathsf{a_3}, \mathsf{id_R}, \mathsf{c_{RV_1}}, \mathsf{idp_1}, \mathsf{p(id_1)}, \ldots, \mathsf{c_{RV_n}}, \mathsf{idp_n}, \mathsf{p(id_n)}) \\ \mid D_n(\mathsf{c_{BD}}, \mathsf{c_{RD}}, \mathsf{c_{DA}}, \mathsf{c_{out}}, \mathsf{a_1}) \mid A_n(\mathsf{c_{BA}}, \mathsf{c_{RA}}, \mathsf{c_{DA}}) \mid \Gamma\Big] \end{aligned}$$

with $\tilde{n} = \mathsf{a_1}, \mathsf{a_2}, \mathsf{a_3}, \mathsf{id_1}, \mathsf{id_2}, \mathsf{r_1}, \mathsf{r_2}, \mathsf{id_R}, \mathsf{c_1}, \mathsf{c_2}, \mathsf{c_{RV_1}}, \mathsf{c_{RV_2}}, \mathsf{c_{BA}}, \mathsf{c_{RA}}, \mathsf{c_{DA}}, \mathsf{c_{BR}}, \mathsf{c_{BD}}, \mathsf{c_{RD}}$ the set of restricted names and the frame $\Gamma = \{^{\mathsf{pk(a_i)}}/_{\mathsf{g_i}} \mid \mathsf{i} = 1..3\} \mid \{^{\mathsf{vk(id_i)}}/_{\mathsf{idp_i}} \mid \mathsf{i} = 1,2\} \mid \{^{\mathsf{vk(id_R)}}/_{\mathsf{idp_R}}\}$. This frame represents the initial knowledge of the attacker: it has access to the public keys of the authorities and the verification keys of the voters. Moreover, since only the two first voters are assumed to be honest, only their two secret ids are restricted (in $\tilde{n}$). The attacker has therefore access to the secret ids of all the other voters.

The process where everyone but the Ballot box is honest:

$$\begin{aligned} P_n^b\,[\_] = \nu\tilde{n}_b.(\ \text{let}\ \mathsf{a_3} = \mathsf{a_1} + \mathsf{a_2}\ \text{in}\ ).\Big[\ \_ \\ \mid R_n(\mathsf{c_{BR}}, \mathsf{c_{RA}}, \mathsf{c_{RD}}, \mathsf{c_{out}}, \mathsf{a_3}, \mathsf{id_R}, \mathsf{c_{RV_1}}, \mathsf{idp_1}, \mathsf{p(id_1)}, \ldots, \mathsf{c_{RV_n}}, \mathsf{idp_n}, \mathsf{p(id_n)}) \\ \mid D_n(\mathsf{c_{BD}}, \mathsf{c_{RD}}, \mathsf{c_{DA}}, \mathsf{c_{out}}, \mathsf{a_1}) \mid A_n(\mathsf{c_{BA}}, \mathsf{c_{RA}}, \mathsf{c_{DA}}) \mid \Gamma_b\Big] \end{aligned}$$

with $\tilde{n}_b = \mathsf{a_1}, \mathsf{a_3}, \mathsf{id_1}, \mathsf{id_2}, \mathsf{r_1}, \mathsf{r_2}, \mathsf{id_R}, \mathsf{c_{RV_1}}, \mathsf{c_{RV_2}}, \mathsf{c_{RA}}, \mathsf{c_{DA}}, \mathsf{c_{RD}}$ the set of restricted names and $\Gamma_b = \{^{\mathsf{pk(a_i)}}/_{\mathsf{g_i}} \mid \mathsf{i} = 1..3\} \mid \{^{\mathsf{vk(id_i)}}/_{\mathsf{idp_i}}, ^{\mathsf{s(id_i)}}/_{\mathsf{s_i}} \mid \mathsf{i} = 1,2\} \mid \{^{\mathsf{vk(id_R)}}/_{\mathsf{idp_R}}\}$. Compared to P_n, we have simply removed B_n from the process and we have further removed the secret key $\mathsf{a_2}$ and the authenticated channels $\mathsf{c_1}, \mathsf{c_2}, \mathsf{c_{BA}}, \mathsf{c_{BR}}, \mathsf{c_{BD}}$ from the set of restricted names. Finally, we have added $\mathsf{s(id_1)}, \mathsf{s(id_2)}$ to the frame representing the initial knowledge. This models the fact that now, all the secrets of B_n are known to the attacker.

2.3 Formal Analysis of Ballot Secrecy

Our analysis shows that the Norwegian e-voting protocol preserves ballot secrecy, even when the Ballot box and all but two voters are corrupted, provided that the other components are honest. This of course implies ballot secrecy if all the authorities are honest and some voters are corrupted. Conversely, we identified several cases of corruption that are subject to attacks. Though not surprising, these cases were not explicitly mentioned in the literature.

Ballot secrecy has been formally defined in terms of equivalence by Delaune, Kremer, and Ryan in [DKR09]. A protocol with process $V(v, id)$ and authority process A preserves *ballot secrecy* if an attacker cannot distinguish when votes are swapped, *i.e.* it cannot distinguish when a voter a_1 votes v_1 and a_2 votes v_2 from the case where a_1 votes v_2 and a_2 votes v_1. This is formally specified by:

$$\nu\tilde{n}.\Big(A \mid V(v_1, a_1) \mid V(v_2, a_2)\Big) \approx_l \nu\tilde{n}.\Big(A \mid V(v_2, a_1) \mid V(v_1, a_2)\Big)$$

Proving ballot secrecy of the Norwegian protocol therefore amounts to prove equivalence of the corresponding processes we detailed in Section 2.2.

2.3.1 Corrupted Ballot Box and Corrupted Voters

Our main result states that the Norwegian protocol specification satisfies ballot secrecy even if the Ballot box and $n-2$ voters are corrupted, provided that the other components are honest.

Theorem 2.1. *Let n be an integer representing the number of voters. Let P_n^b be the process defined in Section 2.2.2, that corresponds to the voting process where all the authorities but the Ballot box are honest. Then,*

$$P_n^b\,[V_1(\mathsf{a}) \mid V_2(\mathsf{b})] \approx_l P_n^b\,[V_1(\mathsf{b}) \mid V_2(\mathsf{a})]$$

with $V_i(\mathsf{v_j}) = V(\mathsf{c_i}, \mathsf{c_{RV_i}}, \mathsf{c_{out}}, \mathsf{g_1}, \mathsf{id_i}, \mathsf{idp_R}, \mathsf{v_j})$ which corresponds to the i-th voter voting $\mathsf{v_j}$.

Proof of Theorem 2.1. In order to prove Theorem 2.1, we need to show equivalence as defined in Definition 1.4. To do so, we "guess" a symmetric relation $\mathcal{R}$ on closed processes satisfying the properties of a labelled bisimilarity. This amounts into describing symbolically all the possible (co-)evolutions of the two processes, depending on the actions of the adversary. The description of this relation is given in Section 2.5. The first step of the proof consists in showing that we did not miss any possible execution, which is ensured by the following proposition.

Proposition 2.1. *Let $\mathcal{R}$ be the relation defined in Definition 2.8 (Section 2.5). Then, $\mathcal{R}$ is verifying properties (2) and (3) of Definition 1.4.*

Proof. Given $(P, Q) \in \mathcal{R}$, we consider all possible evolutions of P in P' and show that there exists Q' such that (P', Q') remains in $\mathcal{R}$. In other words, we check that we did not forget any case when defining $\mathcal{R}$. The detailed proof is not really technical but is rather tedious and is therefore deferred to Section 2.5. □

It then remains to ensure that $\mathcal{R}$ satisfies property (1) of a bisimilarity relation, that is, we need to show that all pairs of frames obtained in the relation $\mathcal{R}$ are in static equivalence. In fact, all these frames are included in the final frames (modulo a "cleaning" step performed using Lemma A.1 presented in Appendix A) corresponding to the complete execution of the two processes. It is therefore sufficient to prove static equivalence of the two final frames.

We consider the following frames:

$$\begin{aligned}
\theta_{\mathsf{init}} &= \{{}^{\mathsf{vk}(\mathsf{id_k})}/_{\mathsf{idp_k}}, {}^{\mathsf{s}(\mathsf{id_k})}/_{\mathsf{s_k}} \mid \mathsf{k} = 1..n\} \mid \{{}^{\mathsf{vk}(\mathsf{id_R})}/_{\mathsf{idp_R}}\} \mid \{{}^{\mathsf{pk}(\mathsf{a_k})}/_{\mathsf{g_k}} \mid \mathsf{k} = 1..3\},\\
\theta_0 &= \theta_{\mathsf{init}} \mid \{{}^{\mathsf{penc}(\mathsf{v_k},\mathsf{r_k},\mathsf{g_1})}/_{\mathsf{e_k}}, {}^{\mathsf{pfk_1}(\mathsf{id_k},\mathsf{t_k},\mathsf{v_k},\mathsf{e_k})}/_{\mathsf{p_k}}, {}^{\mathsf{sign}(\langle \mathsf{e_k},\mathsf{p_k}\rangle,\mathsf{id_k})}/_{\mathsf{si_k}} \mid \mathsf{k} = 1..2\},\\
\theta_k &= \theta_{k-1} \mid \{{}^{\mathsf{sign}(\mathsf{hash}(\Pi_1(M_\mathsf{k})),\mathsf{id_R})}/_{sr_k}, {}^{\mathsf{d}(\mathsf{p}(\mathsf{id_k}),\mathsf{dec}(\Pi_2(M_\mathsf{k}),\mathsf{a_3}))}/_{rec_k}\},\\
\theta_\delta &= \theta_n \mid \{{}^{\mathsf{dec}(U_{\delta(k)},\mathsf{a_1})}/_{res_k} \mid \mathsf{k} = 1..n\}.
\end{aligned}$$

where M_i and U_k are free terms such that $\mathsf{fv}(M_{i+1}) \subseteq \mathsf{dom}(\theta_i)$ and $\mathsf{fv}(U_{k+1}) \subseteq \mathsf{dom}(\theta_n)$. Intuitively, the M_i and U_k are the recipes sent by the adversary. The restriction on the variables makes sure the adversary only use terms he has access to, at this step. δ is a substitution of $[\![1, n]\!]$ intuitively corresponding to the shuffling of the votes at the end of the election. Then each frame can be interpreted as follows:

- θ_{init} corresponds to the initial knowledge of the attacker. It contains the public data of the honest voters and the public keys of the election.

- θ_0 corresponds to the submission of ballots from the two honest voters. Note that our synchronization phase ensure that honest voters vote first. And we can show that the adversary cannot interfere with these two ballots.
- θ_k corresponds to the knowledge of the adversary once the k-th voter has voted. Intuitively, the adversary will submit any ballot he wishes (M_{k}) based on his prior knowledge and in return, he receives the receipt and the signature from the Receipt generator, that is, he receives $\mathsf{d}(\mathsf{p}(\mathsf{id_k}), \mathsf{dec}(\Pi_2(M_{\mathsf{k}}), \mathsf{a_3}))$ and $\mathsf{sign}(\mathsf{hash}(\Pi_1(M_{\mathsf{k}})), \mathsf{id_R})$.
- Then θ_δ corresponds to the frame with the final decryption of the votes, after shuffling, that is, the adversary can see the votes in clear after some permutation δ.

Proposition 2.2. *Let δ is a substitution of $[\![1, n]\!]$ and ${}^t\delta = \delta \circ [1 \mapsto 2, 2 \mapsto 1]$. Let θ_δ be the frame as defined above. Then we have:*

$$\nu\tilde{\omega}.\theta_\delta\sigma_{\mathsf{L}} \approx_s \nu\tilde{\omega}.\theta_{t\delta}\sigma_{\mathsf{R}}, \text{ with } \sigma_{\mathsf{L}} = \{{}^{\mathsf{a}}/_{\mathsf{v}_1}, {}^{\mathsf{b}}/_{\mathsf{v}_2}\} \text{ and } \sigma_{\mathsf{R}} = \{{}^{\mathsf{b}}/_{\mathsf{v}_1}, {}^{\mathsf{a}}/_{\mathsf{v}_2}\}.$$

Proof. The proof is done step by step. First, we show that $\nu\tilde{\omega}.\theta_0\sigma_L \approx_s \nu\tilde{\omega}.\theta_0\sigma_R$, that is, the frames are in static equivalence once the two honest voters have voted (Lemma A.3 detailed in Appendix A). We then show that the receipt sent by the receipt generator does not break static equivalence. This requires to prove in particular that adding the signature of a known term does preserve static equivalence (Lemma 2.8, one of our core lemma). Finally, we conclude by showing that the decryption of the (shuffled) vote only yields already known terms, built using the same recipes, in both frames.

The full proof of Proposition 2.2 can be found in Section 2.6. □

Now since all intermediates frames are sub-frames of the two, showed to be in equivalence by Proposition 2.2, we can deduce that $\mathcal{R}$ satisfies (1) and we conclude.

□

2.3.2 Honest Authorities and Corrupted Voters

The Norwegian E-Voting Protocol specification *a fortiori* satisfies ballot secrecy even if $n - 2$ voters are corrupted, provided that the other components are honest.

Theorem 2.2. *Let n be an integer, that corresponds to the number of voters. Let P_n be the process defined in Section 2.2.2, that corresponds to the voting process when all authorities are honest. Then,*

$$P_n\left[V_1(\mathsf{a}) \mid V_2(\mathsf{b})\right] \approx_l P_n\left[V_1(\mathsf{b}) \mid V_2(\mathsf{a})\right]$$

with $V_i(\mathsf{v_j}) = V(\mathsf{c_i}, \mathsf{c_{RV_i}}, \mathsf{c_{out}}, \mathsf{g_1}, \mathsf{id_i}, \mathsf{idp_R}, \mathsf{v_j})$ which corresponds to the i-th voter voting $\mathsf{v_j}$.

Theorem 2.2 is a corollary of Theorem 2.1. While this is intuitively obvious, the use of equivalence adds some technicalities to the proof. The key proposition is that extending the initial knowledge of the attacker can only help finding attacks.

Lemma 2.1. *Let $\tilde{n}$ be a set of names, P, Q, two processes. We assume that $x \notin \mathsf{fv}(P, Q)$. Then :*

$$\nu\tilde{n}.\left(P \mid \{{}^{M}/_{x}\}\right) \approx_l \nu\tilde{n}.\left(Q \mid \{{}^{M}/_{x}\}\right) \Longrightarrow \nu\tilde{n}.P \approx_l \nu\tilde{n}.Q.$$

Proof. We define a symmetric relation $\mathcal{R}$ on closed extended processes as follows:

$$\nu\tilde{n}.A \;\mathcal{R}\; \nu\tilde{n}.B \overset{\text{def}}{\iff} \nu\tilde{n}.\left(A \mid \{^M/_x\}\right) \approx_l \nu\tilde{n}.\left(B \mid \{^M/_x\}\right).$$

Let us show that $\mathcal{R}$ verifies the three properties of a bisimilarity relation (*cf* Definition 1.4).

1. This is straightforward since $\phi(\nu\tilde{n}.A) \subseteq \phi\left(\nu\tilde{n}.\left(A \mid \{^M/_x\}\right)\right)$.
2. Let $A \to A'$. Then $\nu\tilde{n}.\left(A \mid \{^M/_x\}\right) \to \nu\tilde{n}.\left(A' \mid \{^M/_x\}\right)$. Now, since $A \;\mathcal{R}\; B$, we have that $\nu\tilde{n}.\left(A \mid \{^M/_x\}\right) \approx_l \nu\tilde{n}.\left(B \mid \{^M/_x\}\right)$ thus there exists $\overline{B}$ such that $\nu\tilde{n}.\left(B \mid \{^M/_x\}\right) \to^* \overline{B}$ and $\overline{B} \approx_l \nu\tilde{n}.\left(A' \mid \{^M/_x\}\right)$. Since B is a closed process, it can not make use of x thus it must be the case that there exists B' such that $B \to^* B'$ and $\overline{B} \equiv \nu\tilde{n}.\left(B' \mid \{^M/_x\}\right)$. Moreover since $\nu\tilde{n}.\left(A' \mid \{^M/_x\}\right) \approx_l \overline{B} \equiv \nu\tilde{n}.\left(B' \mid \{^M/_x\}\right)$, we conclude that $A' \;\mathcal{R}\; B'$.
3. Let $A \xrightarrow{\alpha} A'$. Then $\nu\tilde{n}.\left(A \mid \{^M/_x\}\right) \xrightarrow{\alpha} \nu\tilde{n}.\left(A' \mid \{^M/_x\}\right)$. Now, since $A \;\mathcal{R}\; B$, we have that $\nu\tilde{n}.\left(A \mid \{^M/_x\}\right) \approx_l \nu\tilde{n}.\left(B \mid \{^M/_x\}\right)$ thus there exists $\overline{B}$ such that $\nu\tilde{n}.\left(B \mid \{^M/_x\}\right) \to^* \xrightarrow{\alpha} \to^* \overline{B}$ and $\overline{B} \approx_l \nu\tilde{n}.\left(A' \mid \{^M/_x\}\right)$. Since A is a closed process, it does not contain x, thus we must have that the label α does not contain x either. Thus, if $\nu\tilde{n}.\left(B \mid \{^M/_x\}\right) \to^* \xrightarrow{\alpha} \to^* \overline{B}$, we must have that there exists B' such that $B \to^* \xrightarrow{\alpha} \to^* B'$ and $\overline{B} \equiv \nu\tilde{n}.\left(B' \mid \{^M/_x\}\right)$. Moreover since $\nu\tilde{n}.\left(A' \mid \{^M/_x\}\right) \approx_l \overline{B} \equiv \nu\tilde{n}.\left(B' \mid \{^M/_x\}\right)$, we conclude that $A' \;\mathcal{R}\; B'$.

Now, since $\approx_l$ is the largest relation satisfying these three properties, we conclude.

□

We are now ready to prove Theorem 2.2.

Proof of Theorem 2.2. Let us define $A = P_n^b\left[V_1(\mathsf{a}) \mid V_2(\mathsf{b})\right]$ and $B = P_n^b\left[V_1(\mathsf{b}) \mid V_2(\mathsf{a})\right]$. According to Theorem 2.1, we have that: $A \approx_l B$. Thus, for all closing evaluation context $C[\_]$, we also have: $C[A] \approx_l C[B]$. Let us consider the following closing evaluation context:

$$C[\_] = \nu\tilde{m}.\left[\ \_ \mid B_n(\mathsf{c_{BR}}, \mathsf{c_{BD}}, \mathsf{c_{BA}}, \mathsf{c_{out}}, \mathsf{a}_2, \mathsf{idp_R}, \mathsf{c}_1, \mathsf{idp}_1, \mathsf{s}_1, \ldots, \mathsf{c}_n, \mathsf{idp}_n, \mathsf{s}_n)\right],$$

with $\tilde{m} = \mathsf{a}_2, \mathsf{c}_1, \mathsf{c}_2, \mathsf{c_{BR}}, \mathsf{c_{BD}}$ and B_n is defined as the honest one except that we replace arguments $\mathsf{s}(\mathsf{id}_1)$ and $\mathsf{s}(\mathsf{id}_2)$ by variables s_1 and s_2. Then:

$$C[A] = \nu\tilde{m}.\left[\nu\tilde{n}_b.(\text{ let } \mathsf{a}_3 = \mathsf{a}_1 + \mathsf{a}_2 \text{ in }).\left(V_1(\mathsf{a}) \mid V_2(\mathsf{b}) \mid R_n \mid D_n \mid \Gamma_b\right)\sigma_L \mid B_n\right]$$

where B_n, R_n and D_n are just short notations of the different processes with their attributes. Since B_n is such that $(\mathsf{fv}(B_n) \cup \mathsf{fn}(B_n)) \cap \tilde{n}_b = \emptyset$, we have that:

$$\begin{aligned} C[A] &= \nu\tilde{m}.\left[\nu\tilde{n}_b.(\text{ let } \mathsf{a}_3 = \mathsf{a}_1 + \mathsf{a}_2 \text{ in }).\left(V_1(\mathsf{a}) \mid V_2(\mathsf{b}) \mid B_n \mid R_n \mid D_n \mid \Gamma_b\right)\right] \\ &= \nu(\tilde{m}, \tilde{n}_b).(\text{ let } \mathsf{a}_3 = \mathsf{a}_1 + \mathsf{a}_2 \text{ in }).\left[V_1(\mathsf{a}) \mid V_2(\mathsf{b}) \mid B_n \mid R_n \mid D_n \mid \Gamma_b\right]. \end{aligned}$$

But $(\tilde{m}, \tilde{n}_b) = \tilde{n}$ and:

$$\begin{aligned} &B_n(\mathsf{c_{BR}}, \mathsf{c_{BD}}, \mathsf{c_{BA}}, \mathsf{c_{out}}, \mathsf{a}_2, \mathsf{idp_R}, \mathsf{c}_1, \mathsf{idp}_1, \mathsf{s}_1, \ldots, \mathsf{c}_n, \mathsf{idp}_n, \mathsf{s}_n)\Gamma_b \\ &\quad = B_n(\mathsf{c_{BR}}, \mathsf{c_{BD}}, \mathsf{c_{BA}}, \mathsf{c_{out}}, \mathsf{a}_2, \mathsf{idp_R}, \mathsf{c}_1, \mathsf{idp}_1, \mathsf{s}(\mathsf{id}_1), \ldots, \mathsf{c}_n, \mathsf{idp}_n, \mathsf{s}(\mathsf{id}_n))\Gamma_b. \end{aligned}$$

So, we get that $C[A] = \overline{P}_n[V_1(\mathsf{a}) \mid V_2(\mathsf{b})]$, where $\overline{P}_n^a$ is exactly the same as P_n except that Γ is replaced by Γ_b. Doing the same for $C[B]$, we deduce that $C[B] = \overline{P}_n[V_1(\mathsf{b}) \mid V_2(\mathsf{a})]$ and thus $\overline{P}_n[V_1(\mathsf{a}) \mid V_2(\mathsf{b})] \approx_l \overline{P}_n[V_1(\mathsf{b}) \mid V_2(\mathsf{a})]$. According to Lemma 2.1, this equivalence implies that:

$$P_n[V_1(\mathsf{a}) \mid V_2(\mathsf{b})] \approx_l P_n[V_1(\mathsf{b}) \mid V_2(\mathsf{a})].$$

□

2.3.3 Attacks

We have shown that the Norwegian protocol guarantees ballot privacy provided that the Receipt generator, the Decryption service, and the Auditor are honest. We review further cases of corruption where ballot secrecy is no longer guaranteed.

Dishonest Decryption service. The Decryption service is a very sensitive component since it has access to the decryption key a_1 of the public key used for the election. Therefore, a corrupted Decryption service can very easily decrypt all encrypted ballots and thus learns the votes as soon as he has access to the communication between the voters and the Ballot box (these communications being conducted on the *public* Internet network). Even if we did not find any explicit mention of this, we believe that the designers of the protocol implicitly assume that a corrupted Decryption service should not be able to control (some of) the communication over the Internet. It should also be noted that a corrupted Decryption service could learn the votes even *without access to Internet* if the Ballot box does not shuffle the ballots before sending them. How shuffling is performed is not completely clear in [Gjø10].

Dishonest Ballot box and Receipt generator. Clearly, if the Ballot box and the Receipt generator collude, they can compute $\mathsf{a}_1 = \mathsf{a}_3 - \mathsf{a}_2$ and they can then decrypt all incoming encrypted ballots. More interestingly, a corrupted Receipt generator does not need the full cooperation of the ballot box for breaking ballot secrecy. Indeed, assume that the Receipt generator has access, for some voter V, to the blinding factor $\mathsf{s_V}$ used by the Ballot box to blind the ballot. Recall that the Receipt generator retrieves $f(o)^{s_V}$ when generating the receipt codes (by computing $\check{\mathsf{w}}\check{\mathsf{x}}^{-\mathsf{a}_3}$). Therefore, the Receipt generator can compute $\mathsf{f}(\mathsf{o}')^{\mathsf{s_V}}$ for any possible voting option o'. Comparing with the obtained values with $\mathsf{f}(\mathsf{o})^{\mathsf{s_V}}$ it would easily deduce the chosen option o. Of course, the more blinding factors the Receipt generator can get, the more voters it can attack. Therefore, the security of the protocol strongly relies on the security of the blinding factors which generation and distribution are left unspecified in the documentation. The Ballot box can also perform a similar attack, provided it can learn some coding function d_V and additionally, provided that it has access to the SMS sent by the Receipt generator, which is probably a too strong corruption scenario.

Dishonest Ballot box and Auditor. Even if the Auditor does not hold any secret (besides the access to the output of both the Ballot box and the Receipt generator), it is still a key component in the voting process. Indeed, it ensures that the ballots sent to the Decryption service indeed correspond to the ballots sent by the voters (unless both the Ballot box and the Receipt generator are corrupted). Assume now that the Ballot box and the Auditor corrupted. Then the Ballot box can send any ballots it wants to the Decryption service (the - corrupted - Auditor would not complain). This is a clear breach of security in terms of the correctness of the result: indeed, the results would not correspond to the votes as casted by the voters. As a consequence, this is also a breach of ballot privacy. Indeed, the Ballot box may send the same ballot several times (possibly re-randomized) therefore obtaining a bias of information about the vote casted by the voter under attack.

2.4 Lemmas for Static Equivalence

The proof of our main result requires several intermediate lemmas. We believe that some of them are of independent of interest. We first state lemmas that are independent of the equational

theory (Section 2.4.1) and then one of them that relies on it but is still quite general and could be reused for other formal studies of protocols (Section 2.4.2).

2.4.1 Generic Lemmas

We use some arguments repetitively on our proofs and we found useful to state them separately. First, we notice that breaking pairs preserves static equivalence.

Lemma 2.2. *Let $\tilde{n}, \tilde{m}$ be names, θ_1, θ_2 substitutions and let us define $\phi = \nu\tilde{n}.\left(\theta_1 \mid \{^{\langle U_1,U_2\rangle}/_x\}\right)$ and $\psi = \nu\tilde{m}.\left(\theta_2 \mid \{^{\langle U_1,U_2\rangle}/_x\}\right)$ two frames with U_1, U_2 some terms. We have:*

$$\phi \approx_s \psi \iff \nu\tilde{n}.\left(\theta_1 \mid \{^{U_1}/_{x_1}, {}^{U_2}/_{x_2}\}\right) \approx_s \nu\tilde{m}.\left(\theta_2 \mid \{^{U_1}/_{x_1}, {}^{U_2}/_{x_2}\}\right).$$

Proof. Let $\phi' = \nu\tilde{n}.\left(\theta_1 \mid \{^{U_1}/_{x_1}, {}^{U_2}/_{x_2}\}\right)$ and $\psi' = \nu\tilde{m}.\left(\theta_2 \mid \{^{U_1}/_{x_1}, {}^{U_2}/_{x_2}\}\right)$.

$\underline{\Longrightarrow|}$ Let M, N be terms s.t. $(M =_E N)\phi'$. We consider $\delta = [x_1 \mapsto \Pi_1(x), x_2 \mapsto \Pi_2(x)]$ and we have, for all term P: $(P\delta)\phi =_E P\phi'$ and $(P\delta)\psi =_E P\psi'$. Thus:

$$\begin{aligned} M\phi' &=_E N\phi' \\ (M\delta)\phi &=_E (N\delta)\phi \\ (M\delta)\psi &=_E (N\delta)\psi, \text{ since } \phi \approx_s \psi \\ M\psi' &=_E N\psi'. \end{aligned}$$

$\underline{\Longleftarrow|}$ Let M, N be terms s.t. $(M =_E N)\phi$. We consider $\delta : x \mapsto \langle x_1, x_2\rangle$ and we have, for all term P: $(P\delta)\phi' =_E P\phi$ and $(P\delta)\psi' =_E P\psi$. Thus:

$$\begin{aligned} M\phi &=_E N\phi \\ (M\delta)\phi' &=_E (N\delta)\phi' \\ (M\delta)\psi' &=_E (N\delta)\psi', \text{ since } \phi \approx_s \psi \\ M\psi &=_E N\psi. \end{aligned}$$

□

Adding a deducible term to the frames preserves static equivalence, provided that the same recipe is used in the two frames.

Lemma 2.3. *Let $\tilde{n}, \tilde{m}$ be names, θ_1, θ_2 substitutions, and $\phi = \nu\tilde{n}.\theta_1$ and $\psi = \nu\tilde{m}.\theta_2$ frames. Let U be a free term, we have:*

$$\phi \approx_s \psi \iff \nu\tilde{n}.\left(\theta_1 \mid \{^{U\theta_1}/_x\}\right) \approx_s \nu\tilde{m}.\left(\theta_2 \mid \{^{U\theta_2}/_x\}\right).$$

Proof. Let $\phi' = \nu\tilde{n}.\left(\theta_1 \mid \{^{U\theta_1}/_x\}\right)$ and $\psi' = \nu\tilde{m}.\left(\theta_2 \mid \{^{U\theta_2}/_x\}\right)$.

$\underline{\Longleftarrow|}$ Straightforward.

$\underline{\Longrightarrow|}$ Let M, N be terms such that $(M =_E N)\psi'$. We consider $\delta : x \mapsto U$ and we have, for all term P: $(P\delta)\phi =_E P\phi'$ and $(P\delta)\psi =_E P\psi'$. Thus:

$$\begin{aligned} M\phi' &=_E N\phi' \\ (M\delta)\phi &=_E (N\delta)\phi \\ (M\delta)\psi &=_E (N\delta)\psi, \text{ since } \phi \approx_s \psi \\ M\psi' &=_E N\psi'. \end{aligned}$$

□

The next lemmas hold for equational theories for which *destructors* can be identified.

Definition 2.1. *Let E be an equational theory induced by an AC convergent rewriting system $\mathcal{R}$ and Σ a signature. We say that $f \in \Sigma$ is a* destructor in E *if for any rewrite rule $l \to r$ of $\mathcal{R}$*

$$f \notin r \text{ and } (f \notin l \text{ or } l = f(l_1, \ldots, l_n) \text{ and } \forall\, i \in [\![1, n]\!],\ f \notin l_i),$$

i.e. f can only appear in head in l and does not appear in r. Terms of the form $f(t_1, \ldots, t_n)$ with f destructor in E are called destructor terms.

The next lemma states that destructor terms cannot be introduced by rewriting rules.

Lemma 2.4. *Let E be an equational theory induced by an AC-convergent rewriting system $\mathcal{R}$, Σ a signature and $f \in \Sigma$ a destructor in E. Let $C[\_]$ be a context such that $f \notin C$ and let $T_1, \ldots, T_n$ be terms in normal form. Then there exists a context $C'[\_]$ such that $f \notin C'$ and $C[T_1, \ldots, T_n] \to C'[M_1, \ldots, M_p]$ where $M_1, \ldots, M_p$ are subterms of $T_1, \ldots, T_n$.*

Proof. Let us consider such context $C[\_]$ and such terms $T_1, \ldots, T_n$. For this proof, we also consider that we flatten AC symbols, i.e. a term $\oplus(x, \oplus(y, z))$ will be considered as $\oplus(x, y, z)$ for any AC symbol $\oplus$. Moreover, we can consider w.l.o.g. that each T_i does not start with an AC-symbol. If $T_i = \oplus(N_1, \ldots, N_q)$ for one $i \in [\![1, n]\!]$, then we can consider $\overline{C}[\_]$ a new context deduced from $C[\_]$ by adding a $\oplus$ symbol at each position T_i should be inserted in $C[\_]$. Then, we have:

$$C[T_1, \ldots, T_n] =_{AC} \overline{C}[T_1, \ldots, T_{i-1}, N_1, \ldots, N_q, T_{i+1}, \ldots, T_n].$$

Now, if $C[T_1, \ldots, T_n]$ is in normal form, the result is straightforward. If it is not, then we know that there exists a position s, a rule $l \to r$ from $\mathcal{R}$ and a substitution θ such that $C[T_1, \ldots, T_n]|_s =_{AC} l\theta$. Let x be a variable of l and p_x be a position such that $l|_{p_x} = x$. We know that p_x can't be the root, otherwise $l = x$ and $\mathcal{R}$ would contain a rule $x \to r$. Such a rule would allow infinite chains of reduction which is in contradiction with hypothesis on $\mathcal{R}$. Thus, we can move one step up and call p_s the position such that $p_x = p_s * j$ with $j \in \mathbb{N}^*$ and $l|_{p_s} = g \in \Sigma$. We consider two cases:

- g is not an AC-symbol. We consider two subcases:
  - p_x is a position of $C[\_]$. Then $x\theta =_{AC} C_x[T_1, \ldots, T_n]$ with $C_x[\_]$ a sub context of $C[\_]$ and we can replace the substitution of x in θ by $\{^{C_x[T_1,\ldots,T_n]}/_x\}$.
  - p_x is not a position of $C[\_]$, i.e. there exists p_i and p_2 two positions such that $p_x = p_i * p_2$ with $C[T_1, \ldots, T_n]|_{p_i} =_{AC} T_i$ and $T_i|_{p_2} =_{AC} x\theta =_{AC} W$ with $W \in \mathsf{St}(T_i)$. In that case, we can just replace the substitution of x in θ by $\{^W/_x\}$.
- $g = \oplus$, an AC-symbol. Then, we have $l|_{p_s} =_{AC} x \oplus l'$. We consider again two subcases:
  - Either p_s is a position of $C[\_]$. Then, $l|_{p_s}\theta =_{AC} (x \oplus l')\theta =_{AC} C_x[T_1, \ldots, T_n]$. Thus, $\mathsf{head}(C_x[T_1, \ldots, T_n)) = \oplus$ and we can expand it as follows:

    $$C_x[T_1, \ldots, T_n] =_{AC} \bigoplus_{i=1}^{m_1} C_x^i[T_1, \ldots, T_n] \bigoplus_{j=1}^{m_2} W_j$$

    with $C_x^i[\_]$ subcontexts of $C_x[\_]$ and W_j being one of $T_1, \ldots, T_n$. Then, each $x\theta$ is linked to a subset of this sum and we can replace the substitution of x in θ by $\{^{\bigoplus_{i=1}^{w_1} C_x^i[T_1,\ldots,T_n] \bigoplus_{j=1}^{w_2} W_j}/_x\}$ with $w_1 \leq m_1$ and $w_2 \leq m_2$.

– Or $l|_{p_s}\theta =_{AC} W$ with $W \in \mathsf{St}(T_i)$ for one $i \in [\![1, n]\!]$. Then, $x\theta$ is subterm of W and we conclude.

Finally, for each case, we have that $l\theta =_{AC} l\theta'$ where θ' contain only substitutions of variables of l by sub contexts of $C[\_]$ and subterms of $T_1, \ldots, T_n$. Thus, we have:

$$C[T_1, \ldots, T_n]|_s =_{AC} l\theta \to r\theta' =_{AC} C_r[R_1, \ldots, R_{q_r}]$$

where $C_r[\_]$ contains r and all subcontexts $C_x[\_]$, $C_x^k[\_]$. Since $f \notin C$ and $C_x[\_]$, $C_x^k[\_]$ are subcontexts of $C[\_]$, we know that $f \notin C_x$ and $f \notin C_x^k$. Moreover, since f is a destructor in E, $f \notin r$. Thus, $f \notin C_r$. Finally, $R_1, \ldots, R_{q_r}$ are subterms of $T_1, \ldots, T_n$, then:

$$C[T_1, \ldots, T_n] \to C[T_1, \ldots, T_n]\left[C_r[R_1, \ldots, R_{q_r}]\right]_p = C'[M_1, \ldots, M_q].$$

□

If a destructor term t is non deducible and does not appear as subterm of a frame then t cannot appear as subterm of any deducible term.

Lemma 2.5. *Let E be an equational theory induced by an AC convergent rewriting system $\mathcal{R}$, Σ a signature and $f \in \Sigma$ is a destructor in E. Let φ be a frame and $U = f(U_1, \ldots, U_n)$ a term in normal form such that $U \notin \mathsf{St}(\varphi)$ and $\varphi \not\vdash U$. Then:*

$$\textit{for all term } T \textit{ in normal form such that } \varphi \vdash T,\ U \notin \mathsf{St}(T).$$

Proof. Let us prove this by induction on the number of steps used to deduce T.

Base case: $T \in \varphi$. Then, since $U \notin \mathsf{St}(\varphi)$, we have that $U \notin \mathsf{St}(T)$.

Induction case: Now suppose that for any T in normal form such that $\varphi \vdash T$ in n steps, then $U \notin \mathsf{St}(T)$. Let T in normal form such that $\varphi \vdash T$ in $n+1$ steps. We have $\varphi \vdash T =_{AC} g(T_1, \ldots, T_n)\!\downarrow$ where $g \in \Sigma$ and $\varphi \vdash T_i$ in n steps with T_i in normal form for $i \in [\![1, n]\!]$.

- If $g(T_1, \ldots, T_n)$ is already in normal form, then we have two subcases:
  - If $g \neq f$, then according to the induction hypothesis and since $U \notin \mathsf{St}(T_i)$, we have $U \notin \mathsf{St}(T)$.
  - If $g = f$, then, if $U \in \mathsf{St}(T)$ and since $U \notin \mathsf{St}(T_i)$ according to the induction hypothesis, we must have $T = U$, which yields to a contradiction since $\varphi \vdash T$ and $\varphi \not\vdash U$.
- If $g(T_1, \ldots, T_n)$ is not in normal form. If $g = f$ then, we have that $f(T_1, \ldots, T_n) \to r[\widetilde{T_i}]$ for some rule $l \to r$. Since f is a destructor in E, we know that $f \notin r$. Thus, according to Lemma 2.4, we know that $r[\widetilde{T_i}] \to^* C[\widetilde{T_i}]$ such that $f \notin C$. In that case, if $U \in \mathsf{St}(C[\widetilde{T_i}])$, then, we must have $U \in \mathsf{St}(\widetilde{T_i})$ which is a contradiction. If $g \neq f$, then, using Lemma 2.4 with $C[T_1, \ldots, T_n] =_{AC} g(T_1, \ldots, T_n)$, we have that $\exists\, C'$ such that $f \notin C'$ and $g(T_1, \ldots, T_n) \to^* C'[\widetilde{T_i}]$ with $\widetilde{T_i}$ subterms of $T_1, \ldots, T_n$. Then $T =_{AC} C'[\widetilde{T_i}]$ but we know that $U \notin \mathsf{St}(\widetilde{T_i})$ according to induction hypothesis, thus $U \notin \mathsf{St}(T)$.

This concludes the induction. □

A context in normal form applied to a destructor term remains in normal form.

Lemma 2.6. *Let E be an equational theory induced by an AC convergent rewriting system $\mathcal{R}$, Σ a signature and $f \in \Sigma$ is a destructor in E. Let $U = f(U_1, \ldots, U_n)$ a term in normal form and $P[X]_p$ another term in normal form. Then $P[U]_p$ is also in normal form.*

Proof. Assume that $P[U]$ is not in normal form. Since $P[X]$ and U are in normal form, it implies that $\exists\, p$ a position of $P[\_]$ such that $P[U]|_p =_{AC} l\theta$ with some rule $l \to r$ and some substitution θ. But f is a destructor in E, thus, f must appear at the head of l, which is a contradiction with the position p.

□

As a consequence of the previous lemmas, we can state that adding non deducible destructor terms to the frames preserve static equivalence.

Lemma 2.7. *Let E be an equational theory, Σ a signature and $f, g \in \Sigma$ destructors in E. Let $\varphi_1 = \nu\tilde{\omega}_1.\theta_1$ and $\varphi_2 = \nu\tilde{\omega}_2.\theta_2$ be frames. Let $U_1 = f(U_1^1, \ldots, U_p^1)$ and $U_2 = g(U_1^2, \ldots, U_q^2)$ be terms in normal form such that, $\forall\ i \in \{1,2\}$, $\varphi_i \not\vdash U_i$ and $U_i \notin \mathsf{St}(\varphi_i)$. Assume that $x \notin \mathsf{dom}(\varphi_1, \varphi_2)$, then:*

$$\varphi_1 \approx_s \varphi_2 \iff \nu\tilde{\omega}_1.\left(\theta_1 \mid \{^{U_1}/_x\}\right) \approx_s \nu\tilde{\omega}_2.\left(\theta_2 \mid \{^{U_2}/_x\}\right).$$

Proof. Let $\varphi_i' = \nu\tilde{\omega}_i.\left(\theta_i \mid \{^{U_i}/_x\}\right) = \nu\tilde{\omega}_i.\theta_i'$ for $i \in \{1,2\}$.

$\underline{\Longleftarrow|}$ Straightforward.

$\underline{\Longrightarrow|}$ Let M and N be terms s.t. $\mathsf{fv}(M,N) \subseteq \mathsf{dom}(\varphi_1')$ and $\mathsf{bn}(M,N) \cap \mathsf{bn}(\varphi_1') = \emptyset$ with $(M =_E N)\varphi_1'$. Then:

$$\begin{array}{rcl} M\varphi_1' & =_E & N\varphi_1' \\ (M\varphi_1')\!\downarrow & =_{AC} & (N\varphi_1')\!\downarrow \\ (M\varphi_1\delta_1)\!\downarrow & =_{AC} & (N\varphi_1\delta)\!\downarrow \end{array}$$

with $\delta_1 : x \mapsto U_1$, according to φ_1' definition. Since U_1 is in normal form, we apply Lemma 2.6:

$$\begin{array}{rcl} (M\varphi_1)\!\downarrow \delta_1 & =_{AC} & (N\varphi_1)\!\downarrow \delta_1 \\ ((M\varphi_1)\!\downarrow \delta_1)\delta_1^{-1} & =_{AC} & ((N\varphi_1)\!\downarrow \delta_1)\delta_1^{-1} \end{array}$$

with $\delta_1^{-1} : U_1 \mapsto x$. We know that $U_1 = f(U_1^1, \ldots, U_p^1)$ with f destructor in E and U_1 in normal form and not subterm of φ_1. Thus, using Lemma 2.5, for any T in normal form s.t. $\varphi_1 \vdash T$, then $U_1 \notin \mathsf{St}(T)$. And, since $\varphi_1 \vdash (M\varphi_1)\!\downarrow$ (resp. $(N\varphi_1)\!\downarrow$) we have that $(M\varphi_1)\!\downarrow \delta_1^{-1} = (M\varphi_1)\!\downarrow$ (resp. $(N\varphi_1)\!\downarrow \delta_1^{-1} = (N\varphi_1)\!\downarrow$). Thus:

$$\begin{array}{rcl} (M\varphi_1)\!\downarrow \delta_1\delta_1^{-1} & =_{AC} & (N\varphi_1)\!\downarrow \delta_1\delta_1^{-1} \\ (M\varphi_1)\!\downarrow & =_{AC} & (N\varphi_1)\!\downarrow \end{array}$$

Since $\varphi_1 \approx_s \varphi_2$, we have (with $\delta_2 : x \mapsto U_2$):

$$\begin{array}{rcl} (M\varphi_2)\!\downarrow & =_{AC} & (N\varphi_2)\!\downarrow \\ (M\varphi_2)\!\downarrow \delta_2 & =_{AC} & (N\varphi_2)\!\downarrow \delta_2 \end{array}$$

Using Lemma 2.6, we deduce $(M\varphi_2)\downarrow \delta_2 =_{AC} (N\varphi_2)\downarrow$, implying that $(M =_E N)\varphi_1' \to (M =_E N)\varphi_2'$. Repeating the same reasoning, we can also prove that $(M =_E N)\varphi_2' \to (M =_E N)\varphi_1'$, and we conclude.

□

2.4.2 A More Specific Lemma

We show that adding the signature of a deducible term preserves static equivalence, when the same recipe is used in both frames. We believe that this lemma holds for other primitives provided the equations are similar to those for the "sign" symbol like in the case of zero-knowledge proofs.

Lemma 2.8. *Let $\varphi_1 = \nu\omega_1.\theta_1$ and $\varphi_2 = \nu\omega_2.\theta_2$ be two frames, x a fresh variable and a, a name such that $a \in \omega_1 \cap \omega_2$, $\{^{\mathsf{vk}(a)}/_{\mathsf{id_{P_a}}}\} \in \theta_1 \cap \theta_2$ and $\varphi_1, \varphi_2 \not\vdash a$. Let $U = \mathsf{sign}(U', a)$ in normal form with U' a free term and such that $(U\varphi_i)\!\downarrow \notin \mathsf{St}(\varphi_i)$. Then:*

$$\varphi_1 \approx_s \varphi_2 \iff \nu\omega_1.\left(\theta_1 \mid \{^{(U\theta_1)\downarrow}/_x\}\right) \approx_s \nu\omega_2.\left(\theta_2 \mid \{^{(U\theta_2)\downarrow}/_x\}\right).$$

Proof. Let $\overline{\varphi}_i = \nu\omega_1.\overline{\theta}_i$ with $\overline{\theta}_i = \theta_i \mid \{^{(U\varphi_i)\downarrow}/_x\}$ for $i \in \{1,2\}$.

$\underline{\Longleftarrow|}$ Straightforward.

$\underline{\Longrightarrow|}$ We introduce the following notion:

Definition 2.2. *Let us consider* $M_0 \xrightarrow{l_1 \to r_1} M_1 \longrightarrow \ldots \xrightarrow{l_i \to r_i} M_i \ldots \longrightarrow M_n = M_0\downarrow$ *a reduction strategy. This strategy is called* down-to-top*, noted* $\longrightarrow_{dt}$*, if, for all* $i \in [\![1, n]\!]$*,* $M_{i-1}|_{p_i} =_{AC} l_i\theta_i \to r_i\theta_i$ *and for all position* $q < p_i$*,* $M_{i-1}|_q$ *is in normal form.*

We also introduce the following claims that are proved below:

Claim 1. *Let* M *be a term s.t.* $\mathsf{fn}(M) \cap \omega_i = \emptyset$*. Then, for any term* T *s.t.* $M\varphi_i \longrightarrow_{dt}^* T$*, we have* $(U\varphi_i)\downarrow \notin \mathsf{St}(T)$*.*

Claim 2. *Let* M *be a term s.t.* $(U\varphi_i)\downarrow \notin \mathsf{St}(M)$ *and for all term* T *s.t.* $M \longrightarrow_{dt}^* T$*, then* $(U\varphi_i)\downarrow \notin \mathsf{St}(T)$*. Let* $\sigma = \{^{(U\varphi_i)\downarrow}/_x\}$ *a substitution. If* $M\sigma \longrightarrow_{dt}^* T$ *with no rule (8) involved in the reduction path, then there exists some term* T' *such that* $M \longrightarrow_{dt}^* T'$ *and* $T'\sigma =_{AC} T$*.*

We now suppose that $\varphi_1 \approx_s \varphi_2$ and we consider M and N two terms s.t. $\mathsf{fv}(M, N) \subseteq \mathsf{dom}(\overline{\varphi}_1)$ and $\mathsf{fn}(M, N) \cap \omega_1 = \emptyset$. (Since $\varphi_1 \approx_s \varphi_2$, we have $\omega_1 = \omega_2$ and $\mathsf{dom}(\overline{\varphi}_1) = \mathsf{dom}(\overline{\varphi}_2)$.) We are going to prove that $(M =_E N)\overline{\varphi}_1$ iff $(M =_E N)\overline{\varphi}_2$ by induction on the number of positions p in M and N s.t. $M|_p = \mathsf{checksign}(M_1, M_2, M_3)$ and $(M|_p\overline{\varphi}_i)\downarrow =_E \mathsf{ok}$ for $i = 1$ or $i = 2$. We also assume down-to-top reduction strategies only (noted $\longrightarrow_{dt}$).

Base case: There is no such position in M and N. Then, we can apply Claim 2 on $M\overline{\varphi}_i$ and $N\overline{\varphi}_i$. Indeed, let $\sigma_i = \{^{(U\varphi_i)\downarrow}/_x\}$. For $P \in \{M, N\}$, then $P\overline{\varphi}_i = (P\varphi_i)\sigma_i \longrightarrow_{dt}^* (P\overline{\varphi}_i)\downarrow$. If there exists a position p such that $(P\varphi_i)|_p =_{AC} (U\varphi_i)\downarrow$ then, according to the hypothesis on φ_i, p must be a position of P (and not at a leaf). This implies that $P|_{p*2} = a$ which would be a contradiction with the fact that a is restricted. Thus, we have $(U\varphi_i)\downarrow \notin \mathsf{St}(P\varphi_i)$. We also have $P\varphi_i \longrightarrow_{dt}^* (P\varphi_i)\downarrow$ and s.t. $\mathsf{fn}(P) \cap \omega_i = \emptyset$. Using Claim 1, we deduce that $(U\varphi_i)\downarrow$ does not occur in any of the terms in the reduction $P\varphi_i \longrightarrow_{dt}^* (P\varphi_i)\downarrow$. Applying Claim 2 leads to $(P\overline{\varphi}_i)\downarrow =_{AC} (P\varphi_i)\downarrow \sigma_i$. Then:

$$\begin{array}{rlrl}
(M\overline{\varphi}_i)\downarrow &=_{AC} (N\overline{\varphi}_i)\downarrow & (M\varphi_2)\downarrow &=_{AC} (N\varphi_2)\downarrow \quad (\varphi_1 \approx_s \varphi_2)\\
(M\varphi_1)\downarrow \sigma_1 &=_{AC} (N\varphi_1)\downarrow \sigma_1 \quad \text{(Claim 2)} & (M\varphi_2)\downarrow \sigma_2 &=_{AC} (N\varphi_2)\downarrow \sigma_2\\
(M\varphi_1)\downarrow &=_{AC} (N\varphi_1)\downarrow \quad \text{(Claim 1)} & (M\overline{\varphi}_2)\downarrow &=_{AC} (N\overline{\varphi}_2)\downarrow . \quad \text{(Claim 2)}
\end{array}$$

Induction step: We suppose that there exists at least one position p in M or N s.t. $M|_p = \mathsf{checksign}(M_1, M_2, M_3)$ (or $N|_p$) and $(M|_p\overline{\varphi}_i)\downarrow =_E \mathsf{ok}$ (resp. $N|_p$) for $i = 1$ or $i = 2$. Let us consider w.l.o.g. that this position is in M and that it reduces when $\overline{\varphi}_1$ is applied. (We note that if $(M|_p\overline{\varphi}_2)\downarrow =_E \mathsf{ok}$ too then we have $(M[\mathsf{ok}]_p =_E M)\overline{\varphi}_i$ for $i \in \{1,2\}$ and we can conclude using the induction hypothesis on $M[\mathsf{ok}]_p$ and N.) We consider the deepest position satisfying this in M. Thus, we know that M_1, M_2 and M_3 do not contain such a position.

Since $(M|_p\overline{\varphi}_1)\downarrow =_E \mathsf{ok}$, then, according to E, we have that $(M_3\overline{\varphi}_1)\downarrow =_{AC} \mathsf{sign}((M_1\overline{\varphi}_1)\downarrow, Q)$ and $(M_2\overline{\varphi}_1)\downarrow =_{AC} \mathsf{vk}(Q)$ for some term Q. According to the hypothesis on position p, we can apply the same reasoning as used in the base case on M_1, M_2 and M_3 separately and therefore we can apply Claim 2 on them, which leads to $(M_i\overline{\varphi}_1)\downarrow =_{AC} (M_i\varphi_1)\downarrow \sigma_1$ for $i \in \{1,2,3\}$. We have then the following equalities:

- $(M_1\varphi_1)\downarrow =_{AC} T_1$ and $T_1\sigma_1 =_{AC} (M_1\overline{\varphi}_1)\downarrow$.

- $(M_2\varphi_1)\downarrow =_{AC} T_2$ and $T_2\sigma_1 =_{AC} \mathsf{vk}(Q)$. According to definition of σ_1, which does not contain a term of this form, we must have $T_2 =_{AC} \mathsf{vk}(T_2')$ and $T_2'\sigma_1 =_{AC} Q$.

- $(M_3\varphi_1)\downarrow =_{AC} T_3$ and $T_3\sigma_1 =_{AC} \mathsf{sign}((M_1\overline{\varphi}_1)\downarrow, Q)$. Using the two previous equalities, we have $T_3\sigma_1 =_{AC} \mathsf{sign}(T_1\sigma_1, T_2'\sigma_1)$. Then, according to the definition of σ_1, we consider two subcases:

  - $T_3 =_{AC} \mathsf{sign}(T_1, T_2',)$ and then, we have:

$$\begin{aligned}(M|_p\varphi_1)\downarrow &=_E \mathsf{checksign}((M_1\varphi_1)\downarrow, (M_2\varphi_1)\downarrow, (M_3\varphi_1)\downarrow)\downarrow\\ &=_E \mathsf{checksign}(T_1, \mathsf{vk}(T_2'), \mathsf{sign}(T_1, T_2'))\downarrow\\ &=_E \mathsf{ok}.\end{aligned}$$

    Thus, $(M|_p =_E \mathsf{ok})\varphi_1$ and using $\varphi_1 \approx_s \varphi_2$, we have $(M|_p =_E \mathsf{ok})\varphi_2$ which also implies by extension $(M|_p =_E \mathsf{ok})\overline{\varphi}_2$ and we conclude.

  - $T_3 = x$. Then, we have $T_2\sigma_1 =_{AC} \mathsf{vk}(a)$, i.e. $T_2 =_{AC} \mathsf{vk}(a)$, which is $(M_2\varphi_1)\downarrow =_{AC} \mathsf{vk}(a)$. Thus, we have $(M_2 =_E \mathsf{idp_a})\varphi_1$ and, using $\varphi_1 \approx_s \varphi_2$ that leads us to $(M_2 =_E \mathsf{idp_a})\overline{\varphi}_2$. Moreover, $(M_1\overline{\varphi}_1)\downarrow =_{AC} T_1\sigma_1 =_{AC} (U'\varphi_1)\downarrow$, i.e. $T_1 =_{AC} (U'\varphi_1)\downarrow$ according to the definition of σ_1, which implies $(M_1\varphi_1)\downarrow =_{AC} (U'\varphi_1)\downarrow$. Using $\varphi_1 \approx_s \varphi_2$ again, we have $(M_1 =_E U')\overline{\varphi}_2$. Then:

$$\begin{aligned}(M|_p\overline{\varphi}_2)\downarrow &=_E \mathsf{checksign}((M_1\overline{\varphi}_2)\downarrow, (M_2\overline{\varphi}_2)\downarrow, (x\overline{\varphi}_2)\downarrow)\downarrow\\ &=_E \mathsf{checksign}((U'\varphi_2)\downarrow, (\mathsf{idp_a}\varphi_2)\downarrow, (U\varphi_2)\downarrow)\downarrow\\ &=_E \mathsf{ok}.\end{aligned}$$

    And we conclude.

We now prove that the two claims hold.

Proof of Claim 1. We prove this Claim by induction on the size of M. We suppose w.l.o.g. that M is in normal form.

Base case: $M = y$ is a variable (or a name). Since φ_i is in normal form, we have that for all T s.t. $M\varphi_i \longrightarrow_{dt}^* T$ then $M\varphi_i =_{AC} T$. If M is a name, result is straightforward. If M is a variable, then $(U\varphi_i)\downarrow \in \mathsf{St}(T)$ implies that $(U\varphi_i)\downarrow \in \mathsf{St}(\varphi_i)$ which is a contradiction.

Induction Step: We assume that for any term M of size m s.t. $\mathsf{fn}(M) \cap \omega_i = \emptyset$ we have, for all term T s.t. $M\varphi_i \longrightarrow_{dt}^* T$, $(U\varphi_i)\downarrow \notin \mathsf{St}(T)$. We now consider a term M of size $(m+1)$, i.e. $M = \mathsf{f}(M_1, \ldots, M_n)$ with M_i of size at most equal to m. Moreover, following a down-to-top reduction strategy, we have $M\varphi_i = \mathsf{f}(M_1\varphi_i, \ldots, M_n\varphi_i) \longrightarrow_{dt}^* \mathsf{f}(T_1, \ldots, T_n)$ with $T_j = (M_j\varphi_i)\downarrow$. According to the induction hypothesis, we now that for all term T s.t. $M_j\varphi_i \longrightarrow_{dt}^* T$, then $(U\varphi_i)\downarrow \notin \mathsf{St}(T)$. Let us consider now the down-to-top reduction strategy $M\varphi_i \longrightarrow_{dt}^* T = \mathsf{f}(T_1', \ldots, T_n') \longrightarrow_{dt}^* \mathsf{f}(T_1, \ldots, T_n)$. If $(U\varphi_i)\downarrow \in \mathsf{St}(T)$, then we have two possibilities. Either $(U\varphi_i)\downarrow \in \mathsf{St}(T_k')$ for some $k \in [\![1, n]\!]$ and we have a contradiction with the induction hypothesis; or $\mathsf{f}(T_1', \ldots, T_n') =_{AC} (U\varphi_i)\downarrow$ and $\mathsf{f} = \mathsf{sign}$, $n = 2$ and $T_2' = \mathsf{a}$ which implies that a is deducible by φ_i, contradiction. Thus, $(U\varphi_i)\downarrow$ does not occur in any of the terms in the reduction $(M\varphi_i)\downarrow \longrightarrow_{dt}^* \mathsf{f}(T_1, \ldots, T_n)$. We now consider the reduction of $\mathsf{f}(T_1, \ldots, T_n)$:

- It is in normal form, then, we have two subcases:

  - $\mathsf{f} \neq \mathsf{sign}$. Then, since $(U\varphi_i)\downarrow \notin \mathsf{St}(T_j)$ for $j \in [\![1, n]\!]$ by the induction hypothesis, we conclude.
  - $\mathsf{f} = \mathsf{sign}$. Using the induction hypothesis, if $(U\varphi_i)\downarrow \in \mathsf{St}(\mathsf{sign}(T_1, T_2))$, we must have $\mathsf{sign}(T_1, T_2) =_{AC} (U\varphi_i)\downarrow$ and a would be deducible by φ_i which is a contradiction.

- $\mathsf{f}(T_1, \ldots, T_n)$ is not in normal form. Then, we have $\mathsf{f} \neq \mathsf{sign}$ since there are no rule in E s.t. sign is in head. We have $\mathsf{f}(T_1, \ldots, T_n) \longrightarrow_{dt} T$ and we consider two subcases:
  - $T \in \mathsf{St}(T_1, \ldots, T_n)$, then T is in normal form and, using the induction hypothesis, we know that $(U\varphi_i)\downarrow \notin \mathsf{St}(T)$ and we conclude.
  - $T \notin \mathsf{St}(T_1, \ldots, T_n)$, then, according to E, we have used rule (4), (5), (6) or (8), (9), (10). The three last rules lead to $T = \mathsf{ok}$ which concludes straightforwardly and the three first rules lead to T in normal form with the property that if $(U\varphi_i)\downarrow \in \mathsf{St}(T)$, then $(U\varphi_i)\downarrow \in \bigcup \mathsf{St}(T_k)$, which would be a contradiction.

Proof of Claim 2. Let us prove this by induction on the number of reduction steps in $M\sigma \longrightarrow_{dt}^{*} T$.

Base case: If $M\sigma =_{AC} T$, the result is straightforward. If $M\sigma \longrightarrow_{dt} T$ in one step, then, there is a position p, a substitution θ and a rule $l \longrightarrow r$ (different from (8)) s.t. $(M\sigma)|_p =_{AC} l\theta \longrightarrow_{dt} r\theta$. Since σ is in normal form, p must be a position of M. In particular, if we consider $M' = M[z]_p$, we have:

$$M\sigma =_{AC} (M[z]_p\sigma)[l\theta]_p =_{AC} (M'\sigma)[l\theta]_p \text{ and } M\sigma \longrightarrow_{dt} (M'\sigma)[r\theta]_p.$$

We note $\delta : (U\varphi_i)\downarrow \mapsto x$. Then, we have $(M\sigma)\delta =_{AC} ((M'\sigma)[l\theta]_p)\delta$. Hypothesis on M states that there is no position q such that $M|_q =_{AC} (U\varphi_i)\downarrow$. Therefore, there is no position q such that $M'|_q =_{AC} (U\varphi_i)\downarrow$ and the following equalities hold: $(M\sigma)\delta =_{AC} M(\sigma\delta) =_{AC} M$ and $(M'\sigma)\delta =_{AC} M'(\sigma\delta) =_{AC} M'$. Moreover, we know that $(l\theta) \notin \mathsf{St}((U\varphi_i)\downarrow)$ otherwise $(U\varphi_i)\downarrow$ would not be in normal form. Thus, $M =_{AC} M'[(l\theta)\delta]_p$. Since $l \longrightarrow r$ is different from rule (8), we have $(l\theta)\delta = l(\theta\delta) = l\theta' \longrightarrow r\theta'$, for $\theta' = \theta\delta$, and this implies $M \longrightarrow_{dt} M'[r\theta']_p$ following a down-to-top reduction strategy since if there was a lower reduction in M it would also exists in $M\sigma$. Now, we can conclude since:

$$\begin{aligned}(M'[r\theta']_p)\sigma &=_{AC} (M'\sigma)[(r\theta')\sigma]_p \\ &=_{AC} (M'\sigma)[r(\theta'\sigma)]_p \\ &=_{AC} (M'\sigma)[r\theta]_p.\end{aligned}$$

Induction Step: We suppose that, for any term M s.t. for all term T s.t. $M \longrightarrow_{dt}^{*} T$, then $(U\varphi_i)\downarrow \notin \mathsf{St}(T)$, we have: if $M\sigma \longrightarrow_{dt}^{m} T$ in m steps involving no (8) rule, then $M \longrightarrow_{dt}^{m} T'$ with $T'\sigma =_{AC} T$. We consider now M s.t. $M\sigma \longrightarrow_{dt}^{m+1} T$ in $(m+1)$ steps with no rules (8) in the reduction path. Then $M\sigma \longrightarrow_{dt}^{m} M_1 \longrightarrow_{dt} T$. Using the induction hypothesis, we have that $M \longrightarrow_{dt}^{m} M_1'$ with $M_1'\sigma =_{AC} M_1$, thus $M_1'\sigma \longrightarrow_{dt} T$. But M' is verifying hypothesis of Claim 2, otherwise we would have a contradiction with the fact that M is satisfying them. Then, using Base Case, we have $M_1' \longrightarrow_{dt} T'$ with $T'\sigma =_{AC} T$. Thus, $M \longrightarrow_{dt}^{m} M_1' \longrightarrow_{dt} T'$ and $T'\sigma =_{AC} T$, which allows us to conclude.

□

2.5 Defining a Relation $\mathcal{R}$

In this section, we define partial evolutions of processes that will be useful to write the relation $\mathcal{R}$, we describe the relation itself and, finally, we provide a proof for Proposition 2.1. From now and for the rest of the proofs, the randomnesses r_1 and r_2 (which are the honest voters randomnesses) are replaced by t_1 and t_2 for notations issues.

2.5.1 Partial Evolutions

Let us remind that we have the following notations: $\sigma_{\mathsf{L}} = \{^{\mathsf{a}}/_{\mathsf{v}_1}, ^{\mathsf{b}}/_{\mathsf{v}_2}\}$ and $\sigma_{\mathsf{R}} = \{^{\mathsf{b}}/_{\mathsf{v}_1}, ^{\mathsf{a}}/_{\mathsf{v}_2}\}$ to summarize voting choices made by honest voters during processes. We also have $\tilde{n}_b = \mathsf{a}_1, \mathsf{a}_3, \mathsf{id}_1, \mathsf{id}_2, \mathsf{t}_1, \mathsf{t}_2, \mathsf{id}_{\mathsf{R}}, \mathsf{c}_{\mathsf{RV}_1}, \mathsf{c}_{\mathsf{RV}_2}, \mathsf{c}_{\mathsf{RA}}, \mathsf{c}_{\mathsf{DA}}, \mathsf{c}_{\mathsf{RD}}$ representing restricted names corresponding to secret keys, ids and channels. Now, let us introduce the partial evolutions of the processes involved in the overall process P_n^b.

Definition 2.3. *We decompose the process of the (honest) voter using the following notations (we remind that $V_i(\mathsf{v_i}) = V(\mathsf{c_i}, \mathsf{c}_{\mathsf{RV_i}}, \mathsf{c}_{\mathsf{out}}, \mathsf{g}_1, \mathsf{id_i}, \mathsf{idp}_{\mathsf{R}}, \mathsf{v_i})$):*

$$
\begin{array}{rcl}
V_1^i & = & \overline{\mathsf{c}_{\mathsf{out}}}\langle \mathsf{ball_i}\rangle \,.\, V_2^i \\
V_2^i & = & \overline{\mathsf{c_i}}\langle \mathsf{ball_i}\rangle \,.\, V_3^i \\
V_3^i & = & \overline{\mathsf{c}_{\mathsf{RV_i}}}\langle \mathsf{ok}\rangle \,.\, V_4^i \\
V_4^i & = & \mathsf{c}_{\mathsf{RV_i}}(x_r^i) \,.\, V_5^i \\
V_5^i & = & \mathsf{c_i}(x_b^i) \,.\, V_6^i \\
V_6^i & = & \textit{if } \phi_{\mathsf{V}}(\mathsf{id_i}, \mathsf{idp}_{\mathsf{R}}, \mathsf{v_i}, \mathsf{e_i}, \mathsf{p_i}, \mathsf{si_i}, x_b^i, x_r^i) \textit{ then } V_7^i \\
V_7^i & = & \overline{\mathsf{c}_{\mathsf{out}}}\langle \mathsf{ok}\rangle \,.\, V_8^i \\
V_8^i & = & \overline{\mathsf{c}_{\mathsf{RV_i}}}\langle \mathsf{ok}\rangle
\end{array}
\qquad \textit{with:} \quad
\begin{array}{rcl}
\mathsf{ball_i} & = & \langle \mathsf{e_i}, \mathsf{p_i}, \mathsf{si_i}\rangle \\
\mathsf{e_i} & = & \mathsf{penc}(\mathsf{v_i}, \mathsf{t_i}, \mathsf{g}_1) \\
\mathsf{p_i} & = & \mathsf{pfk}_1(\mathsf{id_i}, \mathsf{t_i}, \mathsf{v_i}, \mathsf{e_i}) \\
\mathsf{si_i} & = & \mathsf{sign}(\langle \mathsf{e_i}, \mathsf{p_i}\rangle, \mathsf{id_i})
\end{array}
$$

Definition 2.4. *$R_n(\mathsf{c}_{\mathsf{BR}}, \mathsf{c}_{\mathsf{RA}}, \mathsf{c}_{\mathsf{RD}}, \mathsf{c}_{\mathsf{out}}, \mathsf{a}_3, \mathsf{id}_{\mathsf{R}}, \mathsf{c}_{\mathsf{RV}_1}, \mathsf{idp}_1, \mathsf{p}(\mathsf{id}_1), \ldots, \mathsf{c}_{\mathsf{RV}_n}, \mathsf{idp}_n, \mathsf{p}(\mathsf{id}_n))$, the process of the Receipt generator, is decomposed using the following notations:*

$$
\begin{array}{rcl}
R_1^{i,n} & = & \mathsf{c}_{\mathsf{RV_i}}(go_r^i) \,.\, R_2^{i,n} \\
R_2^{i,n} & = & \mathsf{c}_{\mathsf{BR}}(x_i) \,.\, R_3^{i,n} \\
R_3^{i,n} & = & \textit{if } \phi_{\mathsf{R}}(\mathsf{idp_i}, x_i) \textit{ then } R_4^{i,n} \\
R_4^{i,n} & = & \overline{\mathsf{c}_{\mathsf{BR}}}\langle si_r^i\rangle \,.\, R_5^{i,n} \\
R_5^{i,n} & = & \overline{\mathsf{c}_{\mathsf{out}}}\langle r_i\rangle \,.\, R_6^{i,n} \\
R_6^{i,n} & = & \overline{\mathsf{c}_{\mathsf{RV_i}}}\langle r_i\rangle \,.\, R_7^{i,n} \\
R_7^{i,n} & = & \mathsf{c}_{\mathsf{RV_i}}(sy_i) \,.\, R_1^{i+1,n} \quad \textit{for } i < n
\end{array}
\qquad
\begin{array}{rcl}
R_7^{n,n} & = & \mathsf{c}_{\mathsf{RV_n}}(sy_n) \,.\, R_8^{1,n} \\
R_8^{i,n} & = & \overline{\mathsf{c}_{\mathsf{RA}}}\langle hbr_i\rangle \,.\, R_8^{i+1,n} \\
R_8^{n,n} & = & \overline{\mathsf{c}_{\mathsf{RA}}}\langle hbr_n\rangle \,.\, R_9 \\
R_9 & = & \overline{\mathsf{c}_{\mathsf{RD}}}\langle \mathsf{ok}\rangle
\end{array}
\qquad \textit{with:} \quad
\begin{array}{rcl}
r_i & = & \mathsf{d}(\mathsf{p}(\mathsf{id_i}), \mathsf{dec}(\Pi_2(x_i), \mathsf{a}_3)) \\
si_r^i & = & \mathsf{sign}(\mathsf{hash}(\Pi_1(x_i)), \mathsf{id}_{\mathsf{R}}) \\
hbr_i & = & \langle \mathsf{idp_i}, \mathsf{hash}(\Pi_1(x_i))\rangle
\end{array}
$$

Definition 2.5. *We decompose $D_n(\mathsf{c}_{\mathsf{BD}}, \mathsf{c}_{\mathsf{RD}}, \mathsf{c}_{\mathsf{DA}}, \mathsf{c}_{\mathsf{out}}, \mathsf{a}_1)$, the process of the decryption device using the following notations:*

$$
\begin{array}{rcl}
D_1 & = & \mathsf{c}_{\mathsf{RD}}(go_d^1) \,.\, D_2^{1,n} \\
D_2^{i,n} & = & \mathsf{c}_{\mathsf{BD}}(d_i) \,.\, D_2^{i+1,n} \\
D_2^{n,n} & = & \mathsf{c}_{\mathsf{BD}}(d_n) \,.\, D_3 \\
D_3 & = & \overline{\mathsf{c}_{\mathsf{DA}}}\langle ha_d\rangle \,.\, D_4 \\
D_4 & = & \mathsf{c}_{\mathsf{DA}}(go_d^2) \,.\, D_5^{[\![1,n]\!]}
\end{array}
\qquad
\begin{array}{rcl}
D_5^S & = & \mid_{j \in S} \overline{\mathsf{c}_{\mathsf{out}}}\langle dec_j\rangle, \quad \textit{for} \quad S \in 2^{[\![1,n]\!]} \\
D_5^{\emptyset} & = & 0
\end{array}
\qquad \textit{with:} \quad
\begin{array}{rcl}
dec_i & = & \mathsf{dec}(d_i, \mathsf{a}_1) \\
ha_d & = & \mathsf{hash}(\langle d_1, \ldots, d_n\rangle)
\end{array}
$$

Definition 2.6. *We decompose $A_n(\mathsf{c}_{\mathsf{BA}}, \mathsf{c}_{\mathsf{RA}}, \mathsf{c}_{\mathsf{DA}})$, the process of the auditor using the following notations:*

$$
\begin{array}{rcl}
A_1^{i,n} & = & \mathsf{c_{RA}}(hr_i)\ .\ A_1^{i+1,n} \\
A_1^{n,n} & = & \mathsf{c_{RA}}(hr_n)\ .\ A_2^{1,n} \\
A_2 & = & \mathsf{c_{DA}}(h_d)\ .\ A_3^{1,n} \\
A_3^{i,n} & = & \mathsf{c_{BA}}(hb_i)\ .\ A_3^{i+1,n} \\
A_3^{n,n} & = & \mathsf{c_{BA}}(hb_n)\ .\ A_4 \\
A_4 & = & \mathit{if}\ \phi_{\mathsf{A}}(h_d, hr_1, \ldots, hr_n, hb_1, \ldots, hb_n)\ \mathit{then}\ A_5 \\
A_5 & = & \overline{\mathsf{c_{DA}}}\langle \mathsf{ok} \rangle
\end{array}
$$

Now let us introduce a bunch of notations in order to make things shorter in the relation $\mathcal{R}$.

Definition 2.7. *First of all, we define* $\tilde{m} = \tilde{n}_b, go_r^1, go_r^2, x_r^1, x_r^2, sy_1, sy_2$ *and* $\tilde{h} = \tilde{m}, hr_1, \ldots, hr_n$. *We remind that* $\Gamma_b = \{^{\mathsf{pk(a_k)}}/_{\mathsf{g_k}} \mid \mathsf{k} = 1..3\} \mid \{^{\mathsf{vk(id_k)}}/_{\mathsf{idp_k}}, ^{\mathsf{s(id_k)}}/_{\mathsf{s_k}} \mid \mathsf{k} = 1, 2\} \mid \{^{\mathsf{vk(id_R)}}/_{\mathsf{idp_R}}\}$. *We consider the following set of frames:*

$$
\begin{array}{rl}
\Gamma_{i,j,k} = \Gamma_b \mid & \{^{\mathsf{ok}}/_{go_r^p} \mid \mathsf{p} = 1..i\} \mid \{^{\mathsf{ball_p}}/_{ba_p}, ^{\mathsf{ball_p}}/_{ba'_p} \mid \mathsf{p} = 1..\mathsf{min(i,2)}\} \mid \\
& \{^{r_p}/_{x_r^p} \mid \mathsf{p} = 1..j\} \mid \{^{\mathsf{ok}}/_{conf_p} \mid \mathsf{p} = 1..\mathsf{min(k,2)}\} \mid \{^{\mathsf{ok}}/_{sy_p} \mid \mathsf{p} = 1..k\} \\
\Lambda_i = \Gamma_{n,n,n} \mid & \{^{hbr_p}/_{hr_p} \mid \mathsf{p} = 1..i\} \\
\Lambda = \Lambda_n \mid & \{^{\mathsf{ok}}/_{go_d^1}, ^{\mathsf{ok}}/_{go_d^2}, ^{ha_d}/_{h_d}\} \\
\Delta_S = \Lambda \mid & \{^{dec_{\delta_S(i)}}/_{res_i} \mid \mathsf{i} = 1..\#\overline{S}\} \\
\multicolumn{2}{l}{\mathit{with}\ \delta_S : \overline{S} \mapsto [\![1, \#\overline{S}]\!]\ \mathit{a\ permutation\ and}\ \overline{S} = [\![1,n]\!] \setminus S\ \mathit{where}\ S \in 2^{[\![1,n]\!]}.} \\
\Delta_\delta = \Lambda \mid & \{^{dec_{\delta(1)}}/_{res_1}, \ldots, ^{dec_{\delta(n)}}/_{res_n}\}\ \mathit{with}\ \delta\ \mathit{a\ permutation\ of}\ [\![1,n]\!].
\end{array}
$$

We conclude with a set of contexts:

$$
\begin{array}{rl}
C_0[\_] = & \nu\tilde{n}_b.(\mathit{let}\ \mathsf{a_3} = \mathsf{a_1} + \mathsf{a_2}\ \mathit{in}\ ).[\_ \mid \Gamma_c] \\
C_1^{i,j,k}[\_] = & \nu\tilde{m}_{i,j,k}.(\mathit{let}\ \mathsf{a_3} = \mathsf{a_1} + \mathsf{a_2}\ \mathit{in}\ ).[\_ \mid \Gamma_{i,j,k}] \\
& \mathit{with}\ \tilde{m}_{i,j,k} = \tilde{n}_c, go_r^1, \ldots, go_r^i, x_r^1, \ldots, x_r^j, sy_1, \ldots, sy_k \\
C_2^{i,j,k}[\_] = & \nu\tilde{m}.(\mathit{let}\ \mathsf{a_3} = \mathsf{a_1} + \mathsf{a_2}\ \mathit{in}\ ).[\_ \mid \Gamma_{i,j,k}] \\
C_3^i[\_] = & \nu\tilde{m}, hr_1, \ldots, hr_i.(\mathit{let}\ \mathsf{a_3} = \mathsf{a_1} + \mathsf{a_2}\ \mathit{in}\ ).[\_ \mid \Lambda_i] \\
C_4[\_] = & \nu\tilde{h}, go_d^1.(\mathit{let}\ \mathsf{a_3} = \mathsf{a_1} + \mathsf{a_2}\ \mathit{in}\ ).\left[\_ \mid \Lambda_n \mid \{^{\mathsf{ok}}/_{go_d^1}\}\right] \\
C_5[\_] = & \nu\tilde{h}, go_d^1, h_d.(\mathit{let}\ \mathsf{a_3} = \mathsf{a_1} + \mathsf{a_2}\ \mathit{in}\ ).\left[\_ \mid \Lambda_n \mid \{^{\mathsf{ok}}/_{go_d^1}\} \mid \{^{ha_d}/_{h_d}\}\right] \\
C_6^S[\_] = & \nu\tilde{h}, go_d^1, go_d^2, h_d.(\mathit{let}\ \mathsf{a_3} = \mathsf{a_1} + \mathsf{a_2}\ \mathit{in}\ ).[\_ \mid \Delta_S] \\
C_7^\delta[\_] = & \nu\tilde{h}, go_d^1, go_d^2, h_d.(\mathit{let}\ \mathsf{a_3} = \mathsf{a_1} + \mathsf{a_2}\ \mathit{in}\ ).[\_ \mid \Delta_\delta]\,.
\end{array}
$$

Using these notations, we can notice that we have:

$$
\begin{array}{l}
P_n^b[V(\mathsf{c_1}, \mathsf{c_{RV_1}}, \mathsf{c_{out}}, \mathsf{g_1}, \mathsf{id_1}, \mathsf{idp_R}, \mathsf{v_1}) \mid V(\mathsf{c_2}, \mathsf{c_{RV_2}}, \mathsf{c_{out}}, \mathsf{g_1}, \mathsf{id_2}, \mathsf{idp_R}, \mathsf{v_2})] \\
\quad = C_0\left[V_1^1 \mid V_1^2 \mid R_1^{1,n} \mid D_1 \mid A_1^{1,n}\right].
\end{array}
$$

Description of the Relation

Now, we can define the relation $\mathcal{R}$.

Definition 2.8. *For* $i \in [\![1,n]\!]$, $j \in [\![1,2]\!]$, *we define* M_i, N_j, U_i *and* W_i *as terms such that:*

- $\mathsf{fv}(M_i) \subseteq \mathsf{dom}(C_1^{i,i-1,i-1})$, $\mathsf{fn}(M_i) \cap \mathsf{bn}(C_1^{i,i-1,i-1}) = \emptyset$,
- $\mathsf{fv}(N_j) \subseteq \mathsf{dom}(C_1^{j,j,j-1})$, $\mathsf{fn}(N_j) \cap \mathsf{bn}(C_1^{j,j,i-1}) = \emptyset$,
- $\mathsf{fv}(U_i) \subseteq \mathsf{dom}(C_4)$, $\mathsf{fn}(U_i) \cap \mathsf{bn}(C_4) = \emptyset$,
- $\mathsf{fv}(W_i) \subseteq \mathsf{dom}(C_5)$, $\mathsf{fn}(W_i) \cap \mathsf{bn}(C_5) = \emptyset$.

We define the following abbreviations $\widetilde{M}_k^{i,j} = \{^{M_p}/_{x_p} \mid \mathsf{p} = 1..k\} \mid \{^{si_r^p}/_{sr_p} \mid \mathsf{p} = 1..i\} \mid \{^{r_p}/_{rec_p} \mid \mathsf{p} = 1..j\}$ *and* $\widetilde{N}_k = \{^{N_i}/_{x_b^i} \mid \mathsf{i} = 1..k\}$. *Finally we have* $\widetilde{M} = \widetilde{M}_n^{n,n} \mid \widetilde{N}_2$. *We also have* $\widetilde{U}_i = \widetilde{M} \mid \{^{U_p}/_{d_p} \mid \mathsf{p} = 1..i\}$ *and* $\widetilde{W}_i = \widetilde{U}_n \mid \{^{W_p}/_{hb_p} \mid \mathsf{p} = 1..i\}$.

We consider the smallest relation $\mathcal{R}$ *which is closed under structural equivalence and includes the following pairs of extended processes.*

%*Initialization*

$$C_0\left[V_1^1 \mid V_1^2 \mid R_1^{1,n} \mid D_1 \mid A_1^{1,n}\right]\sigma_L \;\mathcal{R}\; C_0\left[V_1^1 \mid V_1^2 \mid R_1^{1,n} \mid D_1 \mid A_1^{1,n}\right]\sigma_R \tag{2.1}$$

%*Output of encrypted ballot to a public channel (Voter 1)*

$$C_1^{0,0,0}\left[V_2^1 \mid V_1^2 \mid R_1^{1,n} \mid D_1 \mid A_1^{1,n} \mid \{^{\mathsf{ball}_1}/_{ba_1}\}\right]\sigma_L$$
$$\mathcal{R}\; C_1^{0,0,0}\left[V_2^1 \mid V_1^2 \mid R_1^{1,n} \mid D_1 \mid A_1^{1,n} \mid \{^{\mathsf{ball}_1}/_{ba_1}\}\right]\sigma_R \tag{2.2}$$

%*Output of encrypted ballot to B (Voter 1)*

$$C_1^{0,0,0}\left[V_3^1 \mid V_1^2 \mid R_1^{1,n} \mid D_1 \mid A_1^{1,n} \mid \{^{\mathsf{ball}_1}/_{ba_1}, {}^{\mathsf{ball}_1}/_{ba_1'}\}\right]\sigma_L$$
$$\mathcal{R}\; C_1^{0,0,0}\left[V_3^1 \mid V_1^2 \mid R_1^{1,n} \mid D_1 \mid A_1^{1,n} \mid \{^{\mathsf{ball}_1}/_{ba_1}, {}^{\mathsf{ball}_1}/_{ba_1'}\}\right]\sigma_R \tag{2.3}$$

%*Synchronisation with the R (Voter 1)*

$$C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid R_2^{1,n} \mid D_1 \mid A_1^{1,n}\right]\sigma_L \;\mathcal{R}\; C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid R_2^{1,n} \mid D_1 \mid A_1^{1,n}\right]\sigma_R \tag{2.4}$$

%*R receives ballot coming from B (Voter 1)*

$$C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid R_3^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{0,0}\right]\sigma_L$$
$$\mathcal{R}\; C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid R_3^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{0,0}\right]\sigma_R \tag{2.5}$$

%*R's check failed (Voter 1)*

$$C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{0,0}\right]\sigma_L \;\mathcal{R}\; C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{0,0}\right]\sigma_R \tag{2.6}$$

%*R's check succeed (Voter 1)*

$$C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid R_4^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{0,0}\right]\sigma_L$$
$$\mathcal{R}\; C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid R_4^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{0,0}\right]\sigma_R \tag{2.7}$$

%*R outputs a signature for B (Voter 1)*

$$C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid R_5^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,0}\right]\sigma_L$$
$$\mathcal{R}\; C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid R_5^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,0}\right]\sigma_R \tag{2.8}$$

$$\%R \text{ outputs a signature to a public channel (Voter 1)}$$
$$C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid R_6^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1}\right]\sigma_L$$
$$\mathcal{R}\ C_1^{1,0,0}\left[V_4^1 \mid V_1^2 \mid R_6^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1}\right]\sigma_R \tag{2.9}$$
$$\%Reception \text{ of } R\text{'s receipt (Voter 1)}$$
$$C_1^{1,1,0}\left[V_5^1 \mid V_1^2 \mid R_7^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1}\right]\sigma_L$$
$$\mathcal{R}\ C_1^{1,1,0}\left[V_5^1 \mid V_1^2 \mid R_7^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1}\right]\sigma_R \tag{2.10}$$
$$\%Reception \text{ of } B\text{'s confirmation (Voter 1)}$$
$$C_1^{1,1,0}\left[V_6^1 \mid V_1^2 \mid R_7^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1\right]\sigma_L$$
$$\mathcal{R}\ C_1^{1,1,0}\left[V_6^1 \mid V_1^2 \mid R_7^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1\right]\sigma_R \tag{2.11}$$
$$\%Voter\text{'s check failed (Voter 1)}$$
$$C_1^{1,1,0}\left[V_1^2 \mid R_7^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1\right]\sigma_L$$
$$\mathcal{R}\ C_1^{1,1,0}\left[V_1^2 \mid R_7^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1\right]\sigma_R \tag{2.12}$$
$$\%Voter\text{'s check succeed (Voter 1)}$$
$$C_1^{1,1,0}\left[V_7^1 \mid V_1^2 \mid R_7^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1\right]\sigma_L$$
$$\mathcal{R}\ C_1^{1,1,0}\left[V_7^1 \mid V_1^2 \mid R_7^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1\right]\sigma_R \tag{2.13}$$
$$\%Voter \text{ outputs public confirmation (Voter 1)}$$
$$C_1^{1,1,0}\left[V_8^1 \mid V_1^2 \mid R_7^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1 \mid \{^{\mathsf{ok}}/_{conf_1}\}\right]\sigma_L$$
$$\mathcal{R}\ C_1^{1,1,0}\left[V_8^1 \mid V_1^2 \mid R_7^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1 \mid \{^{\mathsf{ok}}/_{conf_1}\}\right]\sigma_R \tag{2.14}$$
$$\%Synchronisation \text{ with } R \text{ and End Voter (Voter 1)}$$
$$C_1^{1,1,1}\left[V_1^2 \mid R_1^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1\right]\sigma_L$$
$$\mathcal{R}\ C_1^{1,1,1}\left[V_1^2 \mid R_1^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1\right]\sigma_R \tag{2.15}$$
$$\%Output \text{ of the encrypted ballot to a public channel (Voter 2)}$$
$$C_1^{1,1,1}\left[V_2^2 \mid R_1^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1 \mid \{^{\mathsf{ball}_2}/_{ba_2}\}\right]\sigma_L$$
$$\mathcal{R}\ C_1^{1,1,1}\left[V_2^2 \mid R_1^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1 \mid \{^{\mathsf{ball}_2}/_{ba_2}\}\right]\sigma_R \tag{2.16}$$
$$\%Output \text{ of the encrypted ballot to } B \text{ (Voter 2)}$$
$$C_1^{1,1,1}\left[V_3^2 \mid R_1^{1,n} \mid D_1 \mid A_1^{1,n} \mid \{^{\mathsf{ball}_2}/_{ba_2}, {}^{\mathsf{ball}_2}/_{ba_2'}\}\right]\sigma_L$$
$$\mathcal{R}\ C_1^{1,1,1}\left[V_3^2 \mid R_1^{1,n} \mid D_1 \mid A_1^{1,n} \mid \{^{\mathsf{ball}_2}/_{ba_2}, {}^{\mathsf{ball}_2}/_{ba_2'}\}\right]\sigma_R \tag{2.17}$$
$$\%Synchronisation \text{ with the } R \text{ (Voter 2)}$$
$$C_1^{2,1,1}\left[V_4^2 \mid R_2^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1\right]\sigma_L$$
$$\mathcal{R}\ C_1^{2,1,1}\left[V_4^2 \mid R_2^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1\right]\sigma_R \tag{2.18}$$

%R receives ballot coming from B (Voter 2)

$$C_1^{2,1,1}\left[V_4^2 \mid R_3^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{1,1} \mid \widetilde{N}_1\right]\sigma_L$$
$$\mathcal{R}\ C_1^{2,1,1}\left[V_4^2 \mid R_3^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{1,1} \mid \widetilde{N}_1\right]\sigma_R \tag{2.19}$$

%R's check failed (Voter 2)

$$C_1^{2,1,1}\left[V_4^2 \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{1,1} \mid \widetilde{N}_1\right]\sigma_L\ \mathcal{R}\ C_1^{2,1,1}\left[V_4^2 \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{1,1} \mid \widetilde{N}_1\right]\sigma_R \tag{2.20}$$

%R's check succeed (Voter 2)

$$C_1^{2,1,1}\left[V_4^2 \mid R_4^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{1,1} \mid \widetilde{N}_1\right]\sigma_L$$
$$\mathcal{R}\ C_1^{2,1,1}\left[V_4^2 \mid R_4^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{1,1} \mid \widetilde{N}_1\right]\sigma_R \tag{2.21}$$

%R outputs a signature to B (Voter 2)

$$C_1^{2,1,1}\left[V_4^2 \mid R_5^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,1} \mid \widetilde{N}_1\right]\sigma_L$$
$$\mathcal{R}\ C_1^{2,1,1}\left[V_4^2 \mid R_5^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,1} \mid \widetilde{N}_1\right]\sigma_R \tag{2.22}$$

%R outputs a signature to a public channel (Voter 2)

$$C_1^{2,1,1}\left[V_4^2 \mid R_6^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_1\right]\sigma_L$$
$$\mathcal{R}\ C_1^{2,1,1}\left[V_4^2 \mid R_6^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_1\right]\sigma_R \tag{2.23}$$

%Reception of R's receipt (Voter 2)

$$C_1^{2,2,1}\left[V_5^2 \mid R_7^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_1\right]\sigma_L$$
$$\mathcal{R}\ C_1^{2,2,1}\left[V_5^2 \mid R_7^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_1\right]\sigma_R \tag{2.24}$$

%Reception of B's confirmation (Voter 2)

$$C_1^{2,2,1}\left[V_6^2 \mid R_7^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_2\right]\sigma_L$$
$$\mathcal{R}\ C_1^{2,2,1}\left[V_6^2 \mid R_7^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_2\right]\sigma_R \tag{2.25}$$

%Voter's check failed (Voter 2)

$$C_1^{2,2,1}\left[R_7^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_2\right]\sigma_L\ \mathcal{R}\ C_1^{2,2,1}\left[R_7^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_2\right]\sigma_R \tag{2.26}$$

%Voter's check succeed (Voter 2)

$$C_1^{2,2,1}\left[V_7^2 \mid R_7^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_2\right]\sigma_L$$
$$\mathcal{R}\ C_1^{2,2,1}\left[V_7^2 \mid R_7^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_2\right]\sigma_R \tag{2.27}$$

%Voter outputs public confirmation (Voter 2)

$$C_1^{2,2,1}\left[V_8^2 \mid R_7^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_2 \mid \{^{\mathsf{ok}}/_{conf_2}\}\right]\sigma_L$$
$$\mathcal{R}\ C_1^{2,2,1}\left[V_8^2 \mid R_7^{2,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_2^{2,2} \mid \widetilde{N}_2 \mid \{^{\mathsf{ok}}/_{conf_2}\}\right]\sigma_R \tag{2.28}$$

The next relations describe the evolution of the protocol during the submissions of the intruder's ballots (corrupted voters).

For $i \in [\![2, n-1]\!]$*:*

$$\begin{aligned} &\%\textit{Synchronisation with } R \textit{ (Voter } i-1\textit{)} \\ &C_2^{i,i,i}\left[R_1^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_i^{i,i} \mid \widetilde{N}_2\right]\sigma_L \\ &\mathcal{R}\ C_2^{i,i,i}\left[R_1^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_i^{i,i} \mid \widetilde{N}_2\right]\sigma_R \end{aligned} \tag{2.29}$$

$$\begin{aligned} &\%\textit{Synchronisation with } R \textit{ (Voter } i\textit{)} \\ &C_2^{i+1,i,i}\left[R_2^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_i^{i,i} \mid \widetilde{N}_2\right]\sigma_L \\ &\mathcal{R}\ C_2^{i+1,i,i}\left[R_2^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_i^{i,i} \mid \widetilde{N}_2\right]\sigma_R \end{aligned} \tag{2.30}$$

$$\begin{aligned} &\%R \textit{ receives the ballot coming from the intruder (Voter } i\textit{)} \\ &C_2^{i+1,i,i}\left[R_3^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i,i} \mid \widetilde{N}_2\right]\sigma_L \\ &\mathcal{R}\ C_2^{i+1,i,i}\left[R_3^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i,i} \mid \widetilde{N}_2\right]\sigma_R \end{aligned} \tag{2.31}$$

$$\begin{aligned} &\%R\textit{'s check failed (Voter } i\textit{)} \\ &C_2^{i+1,i,i}\left[D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i,i} \mid \widetilde{N}_2\right]\sigma_L\ \mathcal{R}\ C_2^{i+1,i,i}\left[D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i,i} \mid \widetilde{N}_2\right]\sigma_R \end{aligned} \tag{2.32}$$

$$\begin{aligned} &\%R\textit{'s check succed (Voter } i\textit{)} \\ &C_2^{i+1,i,i}\left[R_4^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i,i} \mid \widetilde{N}_2\right]\sigma_L \\ &\mathcal{R}\ C_2^{i+1,i,i}\left[R_4^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i,i} \mid \widetilde{N}_2\right]\sigma_R \end{aligned} \tag{2.33}$$

$$\begin{aligned} &\%R \textit{ outputs signature to } B \textit{ (Voter } i\textit{)} \\ &C_2^{i+1,i,i}\left[R_5^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i+1,i} \mid \widetilde{N}_2\right]\sigma_L \\ &\mathcal{R}\ C_2^{i+1,i,i}\left[R_5^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i+1,i} \mid \widetilde{N}_2\right]\sigma_R \end{aligned} \tag{2.34}$$

$$\begin{aligned} &\%R \textit{ outputs receipt publicly (Voter } i\textit{)} \\ &C_2^{i+1,i,i}\left[R_6^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i+1,i+1} \mid \widetilde{N}_2\right]\sigma_L \\ &\mathcal{R}\ C_2^{i+1,i,i}\left[R_6^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i+1,i+1} \mid \widetilde{N}_2\right]\sigma_R \end{aligned} \tag{2.35}$$

$$\begin{aligned} &\%R \textit{ outputs receipt to Voter } i \\ &C_2^{i+1,i+1,i}\left[R_7^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i+1,i+1} \mid \widetilde{N}_2\right]\sigma_L \\ &\mathcal{R}\ C_2^{i+1,i,i}\left[R_7^{i+1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_{i+1}^{i+1,i+1} \mid \widetilde{N}_2\right]\sigma_R \end{aligned} \tag{2.36}$$

Next two relations concern the end of the voting phase and the beginning of the counting phase where R sends its content (hashes) to the Auditor.

For $i \in [\![0, n-1]\!]$*:*

$$C_3^i\left[R_8^{i+1,n} \mid D_1 \mid A_1^{i+1,n} \mid \widetilde{M}\right]\sigma_L\ \mathcal{R}\ C_3^i\left[R_8^{i+1,n} \mid D_1 \mid A_1^{i+1,n} \mid \widetilde{M}\right]\sigma_R \tag{2.37}$$

$$C_3^n\left[R_9 \mid D_1 \mid A_2 \mid \widetilde{M}\right]\sigma_L\ \mathcal{R}\ C_3^n\left[R_9 \mid D_1 \mid A_2 \mid \widetilde{M}\right]\sigma_R \tag{2.38}$$

The final relations show the counting phase with the different steps of the Decryption device which receives encrypted votes from (corrupted) B and waits for Auditor's approval before outputs the shuffled result.

For $i \in [\![1, n]\!]$:

$$C_4\left[D_2^{i,n} \mid A_2 \mid \widetilde{U}_{i-1}\right]\sigma_L \;\mathcal{R}\; C_4\left[D_2^{1,n} \mid A_2 \mid \widetilde{U}_{i-1}\right]\sigma_R \tag{2.39}$$

$$C_4\left[D_3 \mid A_2 \mid \widetilde{U}_n\right]\sigma_L \;\mathcal{R}\; C_4\left[D_3 \mid A_2 \mid \widetilde{U}_n\right]\sigma_R \tag{2.40}$$

$$C_5\left[D_4 \mid A_3^{i,n} \mid \widetilde{W}_{i-1}\right]\sigma_L \;\mathcal{R}\; C_5\left[D_4 \mid A_3^{i,n} \mid \widetilde{W}_{i-1}\right]\sigma_R \tag{2.41}$$

$$C_5\left[D_4 \mid A_4 \mid \widetilde{W}_n\right]\sigma_L \;\mathcal{R}\; C_5\left[D_4 \mid A_4 \mid \widetilde{W}_n\right]\sigma_R \tag{2.42}$$

$$C_5\left[D_4 \mid \widetilde{W}_n\right]\sigma_L \;\mathcal{R}\; C_5\left[D_4 \mid \widetilde{W}_n\right]\sigma_R \tag{2.43}$$

$$C_5\left[D_4 \mid A_5 \mid \widetilde{W}_n\right]\sigma_L \;\mathcal{R}\; C_5\left[D_4 \mid A_5 \mid \widetilde{W}_n\right]\sigma_R \tag{2.44}$$

For $S \in 2^{[\![1,n]\!]} \setminus \{\emptyset\}$, ${}^tS = S[1 \mapsto 2, 2 \mapsto 1]$ and $\delta_{tS} = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_S$ (with $\circ$ the classic composition of functions):

$$C_6^S\left[D_5^S \mid \widetilde{W}_n\right]\sigma_L \;\mathcal{R}\; C_6^{tS}\left[D_5^{tS} \mid \widetilde{W}_n\right]\sigma_R \tag{2.45}$$

For δ a permutation of $[\![1, n]\!]$ and ${}^t\delta$ s.t. ${}^t\delta = [1 \mapsto 2, 2 \mapsto 1] \circ \delta$:

$$C_7^\delta\left[\widetilde{W}_n\right]\sigma_L \;\mathcal{R}\; C_7^{t\delta}\left[\widetilde{W}_n\right]\sigma_R \tag{2.46}$$

2.5.2 Proof of Observational Equivalence, Assuming Static Equivalence

In this section, we show that the relation $\mathcal{R}$ described before captures all the possible evolutions of the initial processes. To prove Proposition 2.1, we first need to guarantee that the tests done by the voters and the Receipt generator pass (or fail) on both processes. This is stated by the following lemma, which assumes static equivalence of the frames. Static equivalence will be proved in the next section.

Lemma 2.9. *We consider the following notations:*

$$\begin{aligned}
\theta_0 &= \{{}^{\mathsf{vk}(\mathsf{id_k})}/_{\mathsf{idp_k}}, {}^{\mathsf{s}(\mathsf{id_k})}/_{\mathsf{s_k}} \mid \mathsf{k} = 1..n\} \mid \{{}^{\mathsf{vk}(\mathsf{id_R})}/_{\mathsf{idp_R}}\} \mid \{{}^{\mathsf{pk}(\mathsf{a_k})}/_{\mathsf{g_k}} \mid \mathsf{k} = 1..3\} \mid \\
&\quad \{{}^{\mathsf{penc}(\mathsf{v_k},\mathsf{t_k},\mathsf{g_1})}/_{\mathsf{e_k}}, {}^{\mathsf{pfk}_1(\mathsf{id_k},\mathsf{t_k},\mathsf{v_k},\mathsf{e_k})}/_{\mathsf{p_k}}, {}^{\mathsf{sign}(\langle \mathsf{e_k},\mathsf{p_k}\rangle,\mathsf{id_k})}/_{\mathsf{si_k}} \mid \mathsf{k} = 1..2\}, \\
\theta_k &= \theta_{k-1} \mid \{{}^{\mathsf{sign}(\mathsf{hash}(\Pi_1(x_k)),\mathsf{id_R})}/_{sr_k}, {}^{\mathsf{d}(\mathsf{p}(\mathsf{id_k}),\mathsf{dec}(\Pi_2(x_k),\mathsf{a_3}))}/_{rec_k}\}, \\
\sigma_i^j &= \{{}^{M_\mathsf{k}}/_{x_k} \mid \mathsf{k} = 1..i\} \mid \{{}^{N_\mathsf{k}}/_{x_b^k} \mid \mathsf{k} = 1..\mathsf{min}(\mathsf{i}, 2)\} \mid \{{}^{U_\mathsf{k}}/_{d_k}, {}^{W_\mathsf{k}}/_{hb_k} \mid \mathsf{k} = 1..j\}.
\end{aligned}$$

Let $i \in \{1, 2\}$ and let M and N be two free terms and let be $P = \mathsf{d}(\mathsf{p}(\mathsf{id_i}), \mathsf{dec}(\Pi_2(N), \mathsf{a_3}))$. We suppose that $\phi_\mathsf{R}(\mathsf{idp_i}, N)\theta_{i-1}\sigma_{i-1}^0\sigma_L = \mathsf{ok}$ and $\phi_\mathsf{V}(\mathsf{id_i}, \mathsf{idp_R}, \mathsf{v_i}, \mathsf{e_i}, \mathsf{p_i}, \mathsf{si_i}, M, P)\theta_i\sigma_i^0\sigma_L = \mathsf{ok}$ (ϕ_R and ϕ_V are defined in Section 2.2.2). We also suppose that $\theta_i\sigma_i^0\sigma_L \approx_s \theta_i\sigma_i^0\sigma_R$. Then, we have $\phi_\mathsf{V}(\mathsf{id_i}, \mathsf{idp_R}, \mathsf{v_i}, \mathsf{e_i}, \mathsf{p_i}, \mathsf{si_i}, M, P)\theta_i\sigma_i^0\sigma_R = \mathsf{ok}$.

Proof. Let $i \in \{1, 2\}$. Since $\phi_\mathsf{V}(\mathsf{id_i}, \mathsf{idp_R}, \mathsf{v_i}, \mathsf{e_i}, \mathsf{p_i}, \mathsf{si_i}, M, P)\theta_i\sigma_i^0\sigma_L = \mathsf{ok}$, we have, in particular, that:

$$(\mathsf{checksign}(\mathsf{hash}(\langle \mathsf{idp_i}, \mathsf{e_i}, \mathsf{p_i}, \mathsf{si_i}\rangle), \mathsf{idp_R}, M) =_E \mathsf{ok})\theta_i\sigma_i^0\sigma_L.$$

Since this test is completely public, and using the fact that $\theta_i\sigma_i^0\sigma_L \approx_s \theta_i\sigma_i^0\sigma_R$, we deduce that:

$$(\mathsf{checksign}(\mathsf{hash}(\langle \mathsf{idp_i}, \mathsf{e_i}, \mathsf{p_i}, \mathsf{si_i}\rangle), \mathsf{idp_R}, M) =_E \mathsf{ok})\theta_i\sigma_i^0\sigma_R.$$

We know that $(\phi_{\mathsf{R}}(\mathsf{idp_i}, N) =_E \mathsf{ok})\theta_{i-1}\sigma_{i-1}^0\sigma_L$. Thus, according to Lemma A.9, for $\sigma \in \{\sigma_L, \sigma_R\}$:

- $(\phi_{\mathsf{R}}(\mathsf{idp_i}, N) =_E \mathsf{ok})\theta_{i-1}\sigma_{i-1}^0\sigma$,
- $(\Pi_2(N) =_E \mathsf{blind}(\mathsf{renc}(\mathsf{e_i}, Q_1), Q_2))\theta_{i-1}\sigma_{i-1}^0\sigma$ for free terms Q_1, Q_2.

Thus, we have, for $\sigma \in \{\sigma_L, \sigma_R\}$:

$$P\theta_i\sigma_i^0\sigma =_E \mathsf{d}(\mathsf{p}(\mathsf{id_i}), \mathsf{dec}(\mathsf{blind}(\mathsf{renc}(\mathsf{e_i}, Q_1), Q_2), \mathsf{a_3}))\theta_i\sigma_i^0\sigma.$$

Using the fact that $\mathsf{d}(\mathsf{p}(\mathsf{id_i}), \mathsf{blind}(\mathsf{v_i}, \mathsf{s}(\mathsf{id_i})))\theta_i\sigma_i^0\sigma_L = P\theta_i\sigma_i^0\sigma_L$, we must have:

$$\mathsf{blind}(\mathsf{v_i}, \mathsf{s}(\mathsf{id_i}))\theta_i\sigma_i^0\sigma_L = \mathsf{dec}(\mathsf{blind}(\mathsf{renc}(\mathsf{e_i}, Q_1), Q_2), \mathsf{a_3})\theta_i\sigma_i^0\sigma_L,$$

which leads us to the fact that $(Q_1 =_E \mathsf{a_2})\theta_i\sigma_i^0\sigma_L$ and $(Q_2 =_E \mathsf{s_i})\theta_i\sigma_i^0\sigma_L$. Thus, since $\theta_i\sigma_i^0\sigma_L \approx_s \theta_i\sigma_i^0\sigma_R$, those equalities holds with $\theta_i\sigma_i^0\sigma_R$ and we can finally state that:

$$\mathsf{blind}(\mathsf{v_i}, \mathsf{s}(\mathsf{id_i}))\theta_i\sigma_i^0\sigma_R = \mathsf{dec}(\mathsf{blind}(\mathsf{renc}(\mathsf{e_i}, Q_1), Q_2), \mathsf{a_3})\theta_i\sigma_i^0\sigma_R,$$

and we conclude.

□

We are now ready to prove Proposition 2.1.

Proposition 2.1. *Let $\mathcal{R}$ be the relation defined in Definition 2.8 (Section 2.5). Then, $\mathcal{R}$ is verifying properties* (2) *and* (3) *of Definition 1.4.*

Proof. Let us prove that $\mathcal{R}$ verifies properties (2) and (3) of Definition 1.4.

Internal Reductions: We must show for all extended processes A and B, where $A\mathcal{R}B$, that if $A \to A'$ for some A', then $B \to^* B'$ with $A'\mathcal{R}B'$ for some B'. We observe that if $A\mathcal{R}B$ by (1), (2), (4), (6) to (8), (10), (12), (13), (15), (16), (18), (20) to (22), (24), (26), (27), (29), (30), (32) to (36), (39), (41), (43), (45) or (46), then there is no extended process A' such that $A \to A'$. We proceed by case analysis depending on which relation is satisfied:

(3) In that case, we have $A \equiv Z_0\left[V_3^1 \mid R_1^{1,n}\right]\sigma_L$ and $B \equiv Z_0\left[V_3^1 \mid R_1^{1,n}\right]\sigma_R$ with $Z_k[\_] = C_1^{k,0,0}\left[\_ \mid V_1^2 \mid D_1 \mid A_1^{1,n} \mid \{^{\mathsf{ball_1}}/_{ba_1}, {}^{\mathsf{ball_1}}/_{ba_1'}\}\right]$. If $A \to A'$, it must be the case that $A \equiv Z_0\left[\overline{\mathsf{c_{RV_1}}}\langle\mathsf{ok}\rangle \,.\, V_4^1 \mid \mathsf{c_{RV_1}}(go_r^1) \,.\, R_2^{1,n}\right]\sigma_L$ and we have $A' \equiv Z_1\left[V_4^1 \mid R_2^{1,n}\right]\sigma_L$. Now, since we have that $B \equiv Z_0\left[\overline{\mathsf{c_{RV_1}}}\langle\mathsf{ok}\rangle \,.\, V_4^1 \mid \mathsf{c_{RV_1}}(go_r^1) \,.\, R_2^{1,n}\right]\sigma_R$, it follows that $B \to B'$ where $B' = Z_1\left[V_4^1 \mid R_2^{1,n}\right]\sigma_R$ and we derive $A'\mathcal{R}B'$ by relation (4) and the closure of $\mathcal{R}$ under structural equivalence.

(5) In that case, we have $A \equiv Z\left[R_3^{1,n}\right]\sigma_L$ and $B \equiv Z\left[R_3^{1,n}\right]\sigma_R$ with the context $Z[\_] = C_1^{1,0,0}\left[\_ \mid V_4^1 \mid V_1^2 \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{0,0}\right]$ and a term M_1 such that $\mathsf{fv}(M_1) \subseteq \mathsf{dom}(C_1^{1,0,0})$ and $\mathsf{fn}(M_1) \cap \mathsf{bn}(C_1^{1,0,0}) = \emptyset$. If $A \to A'$, then $A \equiv Z\left[\text{if } \phi_{\mathsf{R}}(\mathsf{idp_1}, x_1) \text{ then } R_4^{1,n}\right]\sigma_L$

and either $A' \equiv Z\left[R_4^{1,n}\right]\sigma_L$ or $A' \equiv Z\,[0]\,\sigma_L$. In the first case (resp. the second case), $\phi_{\mathsf{R}}(\mathsf{idp_1}, M_1)\theta_0\sigma_0^0\sigma_L$, which is equivalent to $\phi_{\mathsf{R}}(\mathsf{idp_1}, x_1)\sigma_L$ in the Z context, must be equivalent (resp. not equivalent) to ok in E. Then, using Lemma A.9, we know that $\phi_{\mathsf{R}}(\mathsf{idp_1}, M_1)\theta_0\sigma_0^0\sigma_R$ (i.e. $\phi_{\mathsf{R}}(\mathsf{idp_1}, x_1)\sigma_R$ in the Z context) is also equivalent (resp. not equivalent) to ok in E. It follows from $B \equiv Z\left[\text{if } \phi_{\mathsf{R}}(\mathsf{idp_1}, x_1) \text{ then } R_4^{1,n}\right]\sigma_R$ that $B \to B'$ with $B' = Z\left[R_4^{1,n}\right]\sigma_R$ (resp. $B' = Z\,[0]\,\sigma_R$) which leads us to relation (7) (resp. (6)) and we conclude.

(11) In that case, we have $A \equiv Z\left[V_6^1\right]\sigma_L$ and $B \equiv Z\left[V_6^1\right]\sigma_R$ with the context $Z[\_] = C_1^{1,1,0}\left[\_ \mid V_1^2 \mid R_7^{1,n} \mid D_1 \mid A_1^{1,n} \mid \widetilde{M}_1^{1,1} \mid \widetilde{N}_1\right]$ with terms M_1, N_1 s.t. $\mathsf{fv}(M_1) \subseteq \mathsf{dom}(C_1^{1,0,0})$, $\mathsf{fv}(N_1) \subseteq \mathsf{dom}(C_1^{1,1,0})$, $\mathsf{fn}(M_1) \cap \mathsf{bn}(C_1^{1,0,0}) = \emptyset$ and $\mathsf{fn}(N_1) \cap \mathsf{bn}(C_1^{1,1,0}) = \emptyset$. If $A \to A'$, we have $A \equiv Z\left[\text{if } \phi_{\mathsf{V}}(\mathsf{id_1}, \mathsf{idp_R}, \mathsf{v_1}, \mathsf{e_1}, \mathsf{p_1}, \mathsf{si_1}, x_b^1, x_r^1) \text{ then } V_7^1\right]\sigma_L$ and either $A' \equiv Z\left[V_7^1\right]\sigma_L$ or $A' \equiv Z\,[0]\,\sigma_L$. In the first (resp. the second) case, $\phi_{\mathsf{V}}(\mathsf{id_1}, \mathsf{idp_R}, \mathsf{v_1}, \mathsf{e_1}, \mathsf{p_1}, \mathsf{si_1}, N_1, P)\theta_1\sigma_1^0\sigma_L$ with $P = \mathsf{d}(\mathsf{p}(\mathsf{id_1}), \mathsf{dec}(\Pi_2(M_1), \mathsf{a_3}))$, equivalent to $\phi_{\mathsf{V}}(\mathsf{id_1}, \mathsf{idp_R}, \mathsf{v_1}, \mathsf{e_1}, \mathsf{p_1}, \mathsf{si_1}, x_b^1, x_r^1)\sigma_L$ in the Z context, must be equivalent (resp. not equivalent) to ok in E. Then, using Lemma 2.9, $\phi_{\mathsf{V}}(\mathsf{id_1}, \mathsf{idp_R}, \mathsf{v_1}, \mathsf{e_1}, \mathsf{p_1}, \mathsf{si_1}, N_1, P)\theta_1\sigma_1^0\sigma_R$ (i.e. $\phi_{\mathsf{V}}(\mathsf{id_1}, \mathsf{idp_R}, \mathsf{v_1}, \mathsf{e_1}, \mathsf{p_1}, \mathsf{si_1}, x_b^1, x_r^1)\sigma_R$ in the Z context) is also equivalent (resp. not equivalent) to ok in E. It follows from the following equivalence $B \equiv Z\left[\text{if } \phi_{\mathsf{V}}(\mathsf{id_1}, \mathsf{idp_R}, \mathsf{v_1}, \mathsf{e_1}, \mathsf{p_1}, \mathsf{si_1}, x_b^1, x_r^1) \text{ then } V_7^1\right]\sigma_R$ that we have $B \to B'$ with $B' \equiv Z\left[V_7^1\right]\sigma_R$ (resp. $B' \equiv Z\,[0]\,\sigma_R$) which leads us to relation (13) (resp. (12)) and we conclude.

(42) In that case, we have $A \equiv Z\,[A_4]\,\sigma_L$ and $B \equiv Z\,[A_4]\,\sigma_R$ with the context $Z[\_] = C_5\left[\_ \mid D_4 \mid \widetilde{W_n}\right]$ and free terms $M_1, \ldots, M_n$, N_1, N_2, $U_1, \ldots, U_n$ and $W_1, \ldots, W_n$ such that $\mathsf{fv}(M_i) \subseteq \mathsf{dom}(C_1^{i+1,i,i})$, $\mathsf{fn}(M_i) \cap \mathsf{bn}(C_1^{i+1,i,i}) = \emptyset$ for $i \in [\![1, n]\!]$, $\mathsf{fv}(N_i) \subseteq \mathsf{dom}(C_1^{i,i,0})$, $\mathsf{fn}(N_i) \cap \mathsf{bn}(C_1^{i,i,0}) = \emptyset$ for $i \in \{1, 2\}$, $\mathsf{fv}(U_i) \subseteq \mathsf{dom}(C_4)$, $\mathsf{fn}(U_i) \cap \mathsf{bn}(C_4) = \emptyset$ for $i \in [\![1, n]\!]$ and $\mathsf{fv}(W_i) \subseteq \mathsf{dom}(C_5)$, $\mathsf{fn}(W_i) \cap \mathsf{bn}(C_5) = \emptyset$ for $i \in [\![1, n]\!]$. If $A \to A'$, we must have the following equivalence $A \equiv Z\,[\text{if } \phi_{\mathsf{A}}(h_d, hr_1, \ldots, hr_n, hb_1, \ldots, hb_n) \text{ then } A_5]\,\sigma_L$ and either $A' \equiv Z\,[A_5]\,\sigma_L$ or $A' \equiv Z\,[0]\,\sigma_L$. In the first case (resp. second case), we have $\phi_{\mathsf{A}}(\mathsf{hash}(\langle U_1, \ldots, U_n\rangle), hb_r^1, \ldots, hb_r^n, W_1, \ldots, W_n)\theta_n\sigma_n^n\sigma_L$ with $hb_r^i = \langle \mathsf{idp_i}, \mathsf{hash}(\Pi_1(M_i))\rangle$, which is equivalent, in the Z context, to $\phi_{\mathsf{A}}(h_d, hr_1, \ldots, hr_n, hb_1, \ldots, hb_n)\sigma_L$, must be equivalent (resp. not equivalent) to ok in E. Since this test is fully public and $\theta_n\sigma_n^n\sigma_L \approx_s \theta_n\sigma_n^n\sigma_R$, $\phi_{\mathsf{A}}(\mathsf{hash}(\langle U_1, \ldots, U_n\rangle), hb_r^1, \ldots, hb_r^n, W_1, \ldots, W_n)\theta_n\sigma_n^n\sigma_R$ (which is equivalent, in the Z context, to $\phi_{\mathsf{A}}(h_d, hr_1, \ldots, hr_n, hb_1, \ldots, hb_n)\sigma_R$) is also equivalent (resp. not equivalent) to ok in E. It follows from $B \equiv Z\,[\text{if } \phi_{\mathsf{A}}(h_d, hr_1, \ldots, hr_n, hb_1, \ldots, hb_n) \text{ then } A_5]\,\sigma_R$ that we have $B \to B'$ with $B' \equiv Z\,[A_5]\,\sigma_R$ (resp. $B' \equiv Z\,[0]\,\sigma_R$) which leads us to relation (44) (resp. (43)) and we conclude.

The cases (9), (14), (17), (23), (28), (37), (38), (40) and (44) are similar to case (3), (19) is very similar to case (5) and, finally, (25) and (31) can be treated as case (11).

Labelled Reductions: We must show for all extended processes A and B, where $A\mathcal{R}B$, that if $A \xrightarrow{\alpha} A'$ such that $\mathsf{fv}(\alpha) \subseteq \mathsf{dom}(A)$ and $\mathsf{bn}(\alpha) \cap \mathsf{fn}(B) = \emptyset$ for some A', then $B \to^* \xrightarrow{\alpha} \to^* B'$ with $A'\mathcal{R}B'$ for some B'. We observe that if $A\mathcal{R}B$ by (3), (5), (6), (9), (11), (12), (14), (17), (19), (20), (23), (25), (26), (28), (31), (32), (37), (38), (40), (42), (43), (44), or (46), then there is no extended process A' such that $A \xrightarrow{\alpha} A'$. We proceed by case analysis on the remaining cases:

(1) In that case, we have $A \equiv Z_0\left[V_1^1\right]\sigma_L$ and $B \equiv Z_0\left[V_1^1\right]\sigma_R$ with the context $Z_k[\_] =$

$C_k^{0,0,0}\left[\_ \mid V_1^2 \mid R_1^{1,n} \mid D_1 \mid A_1^{1,n}\right]$ (we consider $C_0 = C_0^{0,0,0}$). If $A \xrightarrow{\alpha} A'$ such that $\mathsf{fv}(\alpha) \subseteq \mathsf{dom}(A)$ and $\mathsf{bn}(\alpha) \cap \mathsf{fn}(B) = \emptyset$, then it must be the case that $A \equiv Z_0\left[\overline{\mathsf{c_{out}}}\langle \mathsf{ball_1}\rangle.V_2^1\right]\sigma_L$ and $A' \equiv Z_1\left[V_2^1 \mid \{^{\mathsf{ball_1}}/_{ba_1}\}\right]\sigma_L$ where $\alpha = \nu ba_1.\overline{\mathsf{c_{out}}}\langle \mathsf{ball_1}\rangle$ and $ba_1 \notin \mathsf{dom}(Z_0)$. It follows from $B \equiv Z_0\left[\overline{\mathsf{c_{out}}}\langle \mathsf{ball_1}\rangle.V_2^1\right]\sigma_R$ that $B \xrightarrow{\alpha} B'$ where $B' \equiv Z_1\left[V_2^1 \mid \{^{\mathsf{ball_1}}/_{ba_1}\}\right]\sigma_R$ and we derive $A'\mathcal{R}B'$ by relation (2) and the closure of $\mathcal{R}$ under structural equivalence.

(4) In that case, we have $A \equiv Z\left[R_2^{1,n}\right]\sigma_L$ and $B \equiv Z\left[R_2^{1,n}\right]\sigma_R$ with the context $Z[\_] = C_1^{1,0,0}\left[\_ \mid V_4^1 \mid V_1^2 \mid D_1 \mid A_1^{1,n}\right]$. If $A \xrightarrow{\alpha} A'$ such that $\mathsf{fv}(\alpha) \subseteq \mathsf{dom}(A)$ and $\mathsf{bn}(\alpha) \cap \mathsf{fn}(B) = \emptyset$, then it must be the case that $A \equiv Z\left[\mathsf{c_{BR}}(x_1).R_3^{1,n}\right]\sigma_L$ and $A' \equiv Z\left[R_3^{1,n} \mid \widetilde{M}_1^{0,0}\right]\sigma_L$ where $\alpha = \mathsf{c_{BR}}(M_1)$ and a free term M_1 such that $\mathsf{fn}(M_1) \notin \mathsf{bn}(Z)$. It follows from $B \equiv Z\left[\mathsf{c_{BR}}(x_1).R_3^{1,n}\right]\sigma_R$ that $B \xrightarrow{\alpha} B'$ where $B' \equiv Z\left[R_3^{1,n} \mid \widetilde{M}_1^{0,0}\right]\sigma_R$ which leads us to relation (5) and we conclude.

(45) In that case, we have $A \equiv Z^S\left[D_5^S\right]\sigma_L$ and $B \equiv Z^{tS}\left[D_5^{tS}\right]\sigma_R$ with $Z^S[\_] = C_6^S\left[\_ \mid \widetilde{W}_n\right]$, a set $S \in 2^{[\![1,n]\!]} \setminus \{\emptyset\}$, ${}^tS = S[1 \mapsto 2, 2 \mapsto 1]$, two permutations $\delta_S : \overline{S} \to [\![1, \#\overline{S}]\!]$ and $\delta_{tS} : \overline{{}^tS} \to [\![1, \#\overline{{}^tS}]\!]$ with $\overline{S} = [\![1,n]\!] \setminus S$ (note that $\#{}^tS = \#S$), and free terms $M_1, \ldots, M_n$, N_1, N_2, $U_1, \ldots, U_n$ and $W_1, \ldots, W_n$ such that $\mathsf{fv}(M_i) \subseteq \mathsf{dom}(C_1^{i+1,i,i})$, $\mathsf{fn}(M_i) \cap \mathsf{bn}(C_1^{i+1,i,i}) = \emptyset$ for $i \in [\![1,n]\!]$, $\mathsf{fv}(N_i) \subseteq \mathsf{dom}(C_1^{i,i,0})$, $\mathsf{fn}(N_i) \cap \mathsf{bn}(C_1^{i,i,0}) = \emptyset$ for $i \in \{1,2\}$, $\mathsf{fv}(U_i) \subseteq \mathsf{dom}(C_4)$, $\mathsf{fn}(U_i) \cap \mathsf{bn}(C_4) = \emptyset$ for $i \in [\![1,n]\!]$ and $\mathsf{fv}(W_i) \subseteq \mathsf{dom}(C_5)$, $\mathsf{fn}(W_i) \cap \mathsf{bn}(C_5) = \emptyset$ for $i \in [\![1,n]\!]$. If $A \xrightarrow{\alpha} A'$ such that $\mathsf{fv}(\alpha) \subseteq \mathsf{dom}(A)$ and $\mathsf{bn}(\alpha) \cap \mathsf{fn}(B) = \emptyset$ then it must be the case that $A \equiv Z^S\left[\overline{\mathsf{c_{out}}}\langle dec_i\rangle.D_5^{S\setminus\{i\}}\right]\sigma_L$ with $i \in S$ and $\alpha = \nu res_j.\overline{\mathsf{c_{out}}}\langle res_j\rangle$ and $res_j \notin \mathsf{dom}(Z^S)$ with $j = n + 1 - \#S$. Then, we consider two cases:

- Either $i \in [\![3,n]\!]$. Let $S' = S \setminus \{i\}$. We consider two cases:
  - If $S' \neq \emptyset$ (i.e. $j < n$), we have $\overline{S'} = \overline{S} \cup \{i\}$. Thus, $A' \equiv Z^{S'}\left[D_5^{S'}\right]\sigma_L$ with $\delta_{S'} : \overline{S'} \to [\![1, \#\overline{S'}]\!]$ and $\delta_{S'} = \delta_S \circ [j \mapsto i]$. Since $i \in [\![3,n]\!]$, we also have that ${}^tS' = {}^tS \setminus \{i\}$, thus, it follows from $B \equiv Z^{tS}\left[\overline{\mathsf{c_{out}}}\langle dec_i\rangle.D_5^{{}^tS\setminus\{i\}}\right]\sigma_R$ that $B \xrightarrow{\alpha} B'$ where $B' \equiv Z^{{}^tS'}\left[D_5^{{}^tS'}\right]\sigma_R$ with $\delta_{{}^tS'} = \delta_{{}^tS} \circ [j \mapsto i]$. Since we have $\delta_{{}^tS} = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_S$ and since $\delta_{{}^tS'}(j) = \delta_{S'}(j) = i$, we still have that $\delta_{{}^tS'} = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_{S'}$ which leads us back to relation (45) and we conclude.
  - $S' = \emptyset$ (i.e. $j = n$), we have $\overline{S'} = [\![1,n]\!]$. Thus $A' \equiv C_7^{\delta}\left[\widetilde{W}_n\right]\sigma_L$ with $\delta = \delta_S \circ [n \mapsto i]$. it follows from $B \equiv Z^{tS}\left[\overline{\mathsf{c_{out}}}\langle dec_i\rangle\right]\sigma_R$ that $B \xrightarrow{\alpha} B'$ where $B' \equiv C_7^{{}^t\delta}\left[\widetilde{W}_n\right]\sigma_R$ with ${}^t\delta = \delta_{{}^tS} \circ [n \mapsto i]$. Since we have $\delta_{{}^tS} = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_S$ and since $\delta_{{}^tS'}(n) = \delta_{S'}(n) = j$, we still have that $\delta_{{}^tS'} = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_{S'}$ which leads us to relation (46) and we conclude.
- Or $i \in \{1,2\}$. W.l.o.g., suppose that $i = 1$ and let $S' = S \setminus \{1\}$. We consider two cases:
  - If $S' \neq \emptyset$ (i.e. $j < n$), we have $\overline{S'} = \overline{S} \cup \{1\}$. Thus, $A' \equiv Z^{S'}\left[D_5^{S'}\right]\sigma_L$ with $\delta_{S'} : \overline{S'} \to [\![1, \#\overline{S'}]\!]$ and $\delta_{S'} = \delta_S \circ [j \mapsto 1]$. Since $1 \in S$, we know, by definition of ${}^tS$, that $2 \in {}^tS$. Thus, we have that ${}^tS' = {}^tS \setminus \{2\}$, thus, it follows

from $B \equiv Z^{tS}\left[\overline{\mathsf{c_{out}}}\langle dec_2\rangle.D_5^{tS\setminus\{2\}}\right]\sigma_R$ that $B \xrightarrow{\alpha} B'$ where $B' \equiv Z^{tS'}\left[D_5^{tS'}\right]\sigma_R$ with $\delta_{tS'} = \delta_{tS} \circ [j \mapsto 2]$. Since we have $\delta_{tS} = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_S$ and since $\delta_{tS'}(j) = 2 = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_{S'}(j)$, we still have that $\delta_{tS'} = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_{S'}$ which leads us back to relation (45) and we conclude.

– $S' = \emptyset$ (i.e. $j = n$), we have $\overline{S'} = [\![1, n]\!]$. Thus $A' \equiv C_7^{\delta}\left[\widetilde{W_n}\right]\sigma_L$ with $\delta = \delta_S \circ [n \mapsto 1]$. it follows from $B \equiv Z^{tS}\left[\overline{\mathsf{c_{out}}}\langle dec_2\rangle\right]\sigma_R$ that $B \xrightarrow{\alpha} B'$ where $B' \equiv C_7^{t\delta}\left[\widetilde{W_n}\right]\sigma_R$ with ${}^t\delta = \delta_{tS} \circ [n \mapsto 2]$. Since we have $\delta_{tS} = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_S$ and since $\delta_{tS'}(n) = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_{S'}(n)$, we still have that $\delta_{tS'} = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_{S'}$ which leads us to relation (46) and we conclude.

The cases (2), (7), (8), (13), (15), (16), (21), (22), (27), (33), (34) and (35) are similar to case (1). Cases (10), (18), (24), (29), (30), (36), (39) and (41) are very similar to (2).

□

2.6 Static Equivalence of the Final Frame

This section is devoted to the proof of Proposition 2.2, that states static equivalence of the final frame obtained in the relation $\mathcal{R}$. The proof of Proposition 2.2 relies on many lemmas. Some of them have been already proved in the core of the paper, some of them are given in Appendix A.

Proposition 2.2. *Let δ is a substitution of $[\![1, n]\!]$ and ${}^t\delta = \delta \circ [1 \mapsto 2, 2 \mapsto 1]$. Let θ_δ be the frame as defined above. Then we have:*

$$\nu\tilde{\omega}.\theta_\delta\sigma_{\mathsf{L}} \approx_s \nu\tilde{\omega}.\theta_{t\delta}\sigma_{\mathsf{R}}, \text{ with } \sigma_{\mathsf{L}} = \{{}^{\mathsf{a}}/_{\mathsf{v}_1}, {}^{\mathsf{b}}/_{\mathsf{v}_2}\} \text{ and } \sigma_{\mathsf{R}} = \{{}^{\mathsf{b}}/_{\mathsf{v}_1}, {}^{\mathsf{a}}/_{\mathsf{v}_2}\}.$$

Proof. We introduce the notation $\hat{\theta}_i^{\delta} = \theta_n \mid \{{}^{dec_{\delta(k)}}/_{res_k} \mid k \in [\![1, i]\!]\}$. We show by induction on i that:

$$\nu\tilde{\omega}.\hat{\theta}_i^{\delta}\hat{\sigma}_n^n\sigma_L \approx_s \nu\tilde{\omega}.\hat{\theta}_i^{t\delta}\hat{\sigma}_n^n\sigma_R.$$

Base case. The base case is ensured by Lemma A.10 which guarantees that $\nu\tilde{\omega}.\theta_n\hat{\sigma}_n^0\sigma_L \approx_s \nu\tilde{\omega}.\theta_n\hat{\sigma}_n^0\sigma_R$ thus $\nu\tilde{\omega}.\hat{\theta}_0^{\delta}\hat{\sigma}_n^n\sigma_L \approx_s \nu\tilde{\omega}.\hat{\theta}_0^{t\delta}\hat{\sigma}_n^n\sigma_R$.

Induction step. Suppose that $\nu\tilde{\omega}.\hat{\theta}_i^{\delta}\hat{\sigma}_n^n\sigma_L \approx_s \nu\tilde{\omega}.\hat{\theta}_i^{t\delta}\hat{\sigma}_n^n\sigma_R$ for some $i \geq 0$. Let us show that:

$$\nu\tilde{\omega}.\hat{\theta}_{i+1}^{\delta}\hat{\sigma}_n^n\sigma_L \approx_s \nu\tilde{\omega}.\hat{\theta}_{i+1}^{t\delta}\hat{\sigma}_n^n\sigma_R.$$

We consider $dec_{\delta(i+1)} = \mathsf{dec}(d_{\delta(i+1)}, \mathsf{a}_1) = \mathsf{dec}(U_{\delta(i+1)}, \mathsf{a}_1)$ in the frame $\hat{\theta}_{i+1}^{\delta}\hat{\sigma}_n^n$. We distinguish cases according to the value of $\delta(i+1)$:

- If $\delta(i+1) = 1$, then, using Lemma A.11 (since we are considering the decryption step of the protocol, ϕ_A must have been successful), we know that $U_k\theta_n\hat{\sigma}_n^n\sigma = \mathsf{e_k}\theta_n\hat{\sigma}_n^n\sigma$ for $k \in \{1, 2\}$ and any $\sigma \in \{\sigma_L, \sigma_R\}$. Thus, we have that $res_{i+1}\hat{\theta}_{i+1}^{\delta}\hat{\sigma}_n^n\sigma_L =_E \mathsf{dec}(U_1, \mathsf{a}_1)\hat{\theta}_{i+1}^{\delta}\hat{\sigma}_n^n\sigma_L =_E \mathsf{v}_1\sigma_L$. But, we also have the following equalities $res_{i+1}\hat{\theta}_{i+1}^{t\delta}\hat{\sigma}_n^n\sigma_R =_E \mathsf{dec}(U_2, \mathsf{a}_1)\hat{\theta}_{i+1}^{t\delta}\hat{\sigma}_n^n\sigma_R =_E \mathsf{v}_2\sigma_R$. According to the definition of σ_L and σ_R, we have $\mathsf{v}_1\sigma_L =_E \mathsf{v}_2\sigma_R =_E \mathsf{a}$. Then, using the induction hypothesis and Lemma 2.3 with the free term $U = \mathsf{a}$ and we conclude.

- If $\delta(i+1) = 2$, we adopt the same argument used in the previous case (replacing 1 by 2 and 2 by 1) and conclude using the induction hypothesis and Lemma 2.3 with the free term $U = \mathsf{b}$.

- If $\delta(i+1) = j \in [\![3, n]\!]$, then ${}^t\delta(i+1) = \delta(i+1)$ and we have $res_{i+1} = \mathsf{dec}(U_j, \mathsf{a}_1)$. According to Lemma A.11, we have that $U_j\theta_n\hat{\sigma}_n^n\sigma = \mathsf{penc}(N, P, Q)\ \theta_n\hat{\sigma}_n^n\sigma$ for free terms N, P, Q and any $\sigma \in \{\sigma_L, \sigma_R\}$. Thus, $res_{i+1}\hat{\theta}_{i+1}^{\delta}\hat{\sigma}_n^n\sigma_L =_E \mathsf{dec}(\mathsf{penc}(N, P, Q), \mathsf{a}_1)\theta_n\hat{\sigma}_n^n\sigma_L$. We consider two subcases:

  - $Q\theta_n\hat{\sigma}_n^n\sigma_L =_E \mathsf{pk}(\mathsf{a}_1)$. Then, $res_{i+1}\hat{\theta}_{i+1}^{\delta}\hat{\sigma}_n^n\sigma_L =_E N\hat{\theta}_{i+1}^{\delta}\hat{\sigma}_n^n\sigma_L$ with free N. And, in that case, we also have $Q\theta_n\hat{\sigma}_n^n\sigma_R =_E \mathsf{pk}(\mathsf{a}_1)$ (using the induction hypothesis and the fact that $\mathsf{g}_1 = \mathsf{pk}(\mathsf{a}_1)$) and $res_{i+1}\hat{\theta}_{i+1}^{t\delta}\hat{\sigma}_n^n\sigma_R =_E N\hat{\theta}_{i+1}^{t\delta}\hat{\sigma}_n^n\sigma_R$ too. Thus, we conclude using Lemma 2.3.
  - $Q\theta_n\hat{\sigma}_n^n\sigma_L \neq_E \mathsf{pk}(\mathsf{a}_1)$. Then, there is no reduction of res_{i+1} (in both frames) and we have two possibilities. If there exists $j \in [\![1, i]\!]$ such that $res_j\hat{\theta}_j^{\delta}\hat{\sigma}_n^n\sigma_L = res_{i+1}\hat{\theta}_{i+1}^{\delta}\hat{\sigma}_n^n\sigma_L$, then we conclude using Lemma 2.3. If there is not, then since a_1 is restricted and not deducible (Lemma A.2), we have that res_{i+1} is neither deducible itself, nor a subterm of $\hat{\theta}_i^{\delta}\hat{\sigma}_n^n\sigma_L$ and $\hat{\theta}_i^{t\delta}\hat{\sigma}_n^n\sigma_R$. Now since dec is a destructor in E, we use Lemma 2.7 to conclude.

Finally, we have that:

$$\nu\tilde{\omega}.\hat{\theta}_n^{\delta}\hat{\sigma}_n^n\sigma_L \approx_s \nu\tilde{\omega}.\hat{\theta}_n^{t\delta}\hat{\sigma}_n^n\sigma_R,$$

which is, by using the correct notations:

$$\nu\tilde{\omega}.\theta_{\delta}\hat{\sigma}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{t\delta}\hat{\sigma}\sigma_R.$$

□

2.7 Further Corruption Cases Using ProVerif

In order to study further corruption cases, we have used ProVerif, one of the only tools that can analyse equivalence properties in the context of security protocols. Of course, we needed to simplify the equational theory since ProVerif does not handle associative and commutative (AC) symbols and our theory requires four of them. So we have considered the theory E' defined by the equations of Figure 2.3, except equation (E-5) that represents homomorphic combination of ciphertexts and we have replaced AC symbols $+$ and $*$ by free function symbols f and g. Using this simplified theory, it is clear that we can miss some attacks, but testing corruption assumptions is still relevant even if the attacker is a bit weaker than in our first study.

As ProVerif is designed to prove equivalences between processes that differ only by terms, we need to use another tool, ProSwapper [KSR10], to model the shuffle done by the Decryption service. More precisely, we actually used their algorithm to compute directly a shuffle in our ProVerif specification. The results are displayed in Table 2.1. In these tables, ✓ indicates that ballot secrecy is satisfied and × shows that there is an attack. In particular, all the attacks described in Section 2.3.3 are displayed in the table. The ProVerif files corresponding to our experimentation can be found at [Wie]. Our case study with ProVerif indicates that ballot secrecy is still preserved even when the Receipt generator is corrupted (as well as several voters), at least in the simplified theory.

Table 2.1: Results for Ballot Secrecy

	Ballot Secrecy				
Corrupted Players \ Corrupted Voters	0	1	4	8	10
None	✓ (Theorem 2.2)				
B	✓ (Theorem 2.1)				
R	✓ (ProVerif)	✓ (ProVerif)	✓ (ProVerif)	? (>1h)	? (>1h)
A	✓ (ProVerif)	✓ (ProVerif)	✓ (ProVerif)	✓ (ProVerif)	? (>1h)
D + ⋆	× (See Section 2.3.3: Attacks)				
B + R	× (See Section 2.3.3: Attacks)				
B + A	× (See Section 2.3.3: Attacks)				
R + A	✓ (ProVerif)	✓ (ProVerif)	✓ (ProVerif)	✓ (ProVerif)	? (>1h)

2.8 Conclusion

We have proposed the first formal proof that the e-voting protocol used in Norway indeed satisfies ballot secrecy, even when all but two voters are corrupted and even when the Auditor and either the Ballot box or the Receipt generator are corrupted. As expected, ballot secrecy is no longer guaranteed if both the Ballot box and the Receipt generator are corrupted. Slightly more surprisingly, the protocol is not secure either if the Decryption service is corrupted or if the Ballot box and the Auditor are corrupted, as discussed in Section 2.3.3. It remains to study other security properties such as coercion-resistance or verifiability. Instead of doing additional (long and technical) proofs, a further step consists in developing a procedure for automatically checking for equivalences. Of course, this is a difficult problem. A first decision procedure has been proposed in [CD09] but is limited to subterm convergent theories. An implementation has recently been proposed [CCLD11] but it does not support such a complex equational theory. An alternative step would be to develop a sound procedure that over-approximate the relation, losing completeness in the spirit of ProVerif [Bla05] but tailored to privacy properties.

We would like to emphasize that the security proofs have been conducted in a symbolic thus abstract model. This provides a first level of certification, ruling out "logical" attacks. However, a full computational proof should be developed. Our symbolic proof can been seen as a first step, identifying the set of messages an attacker can observe when interacting with the protocol. There is however still a long way to go for a computational proof. In particular, it remains to identify which the security assumptions are needed.

It is also important to note that the security of the protocol strongly relies on the way initial secrets are pre-distributed. For example, three private decryption keys $\mathsf{a}_1, \mathsf{a}_2$, a_3 (such that $\mathsf{a}_1 + \mathsf{a}_2 = \mathsf{a}_3$) need to be securely distributed among (respectively) the Ballot box, the Receipt generator and the Decryption service. Also, a table (V, s_V) containing the blinding factor for each voter needs to be securely distributed to Ballot box and a table (V, d_V) containing a permutation for each voter needs to be securely distributed to the Receipt generator. Moreover, anyone with access with both the codes mailed to the voters and to the SMS emitted by the Receipt generator would immediately learn the values of all the votes. We did not find in the documentation how and by who all these secret values were distributed. It should certainly be clarified as it could be a weak point of the system.

Chapter 3

CNRS Boardroom Voting Protocol

Contents

Electronic voting has garnered a lot of attention in the past years. Most of the results in this field have been focused on two main types of settings: distant electronic voting (e.g. Helios [Adi08], Civitas [CCM08], or FOO [FOO92]) and voting machines (e.g. Diebold machines [FHF07] or Indian voting machines [WWH+10], both of them having been subject to attacks [FHF07, WWH+10]). In this chapter, we focus on a different and particular setting: boardroom meetings. Many committee meetings require their members to vote on several motions/decisions. Different techniques are typically used: show of hands which is simple and cheap, and paper ballot which grants privacy but may be tedious when there are several rounds of vote. More recently, the use of electronic devices has been developed and seems to offer both simplicity of use and privacy: committee members simply need to (privately) push a button corresponding to their choice on their own device and a central device computes and publishes the result.

However, these systems are opaque: what if someone controls the central device and therefore falsifies the result of the election? In many committees such as boarding committees or scientific

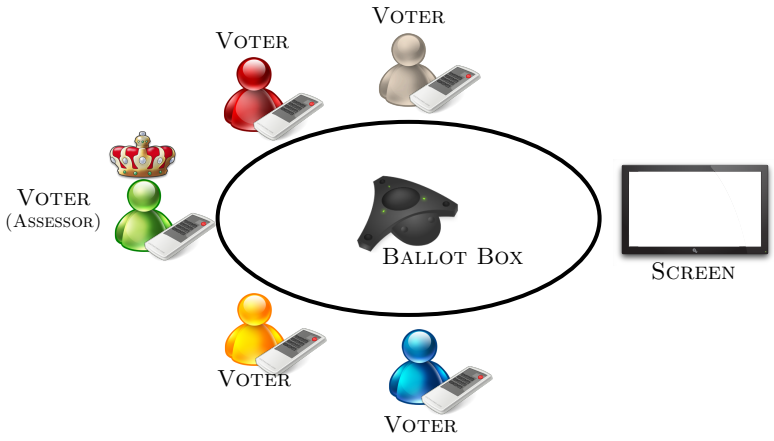


Figure 3.1: Schema of the election

councils, controlling the result of the election (e.g. choice of a new president, decision on the future of a company, *etc.*) is even more important in terms of impact than breaking privacy. Even if the system is not malicious, it can simply dysfunction with no notifications, as witnessed e.g. by the "CNRS Section 07" committee members (the scientific council in Computer Science of the CNRS, a French national research institute). In response to these dysfunctions, a subgroup of the CNRS Section 07 committee members, namely Bruno Durand, Chantal Enguehard, Marc-Olivier Killijian and Philippe Schnoebelen, with the help of Stefan Merz and Blaise Genest, have proposed a new voting system that is meant to achieve: simplicity, ballot secrecy, and full verifiability even in case of corrupted devices. A few other systems tailored to boardroom election have been proposed such as [Gro04, HRZ10]. A feature of the "CNRS Section 07" system is that it does not use cryptography, which makes the system easier to understand and trust, for non experts.

In this chapter, we provide a full review of the voting system proposed by the CNRS Section 07. After a short description of the context of boardroom voting in Section 3.1, we describe and discuss three different versions of the voting protocol towards Section 3.2. We provide a formal model of two of these versions in applied pi-calculs [AF01] in Section 3.3. Finally, we formally analyze these models regarding ballot-secrecy (Section 3.4) and vote correctness (Section 3.5) under different corruption scenarios.

3.1 Context

We consider a particular context, typically for boardroom meetings, where all voters are present in the same room and are given a dedicated voting equipment. In what follows, we assume the individual devices to be linked to a central device. The central device is responsible for collecting the ballots and publishing them. Such systems are standard in many committees (e.g. parliamentary assembly, corporate boards, *etc.*). The particularity of the voting system (and its variants) proposed by the CNRS Section 07 is that it assumes the presence of a Screen that each voter can see. The Screen ensures that all voters simultaneously see the same data and is the key element for the voting system.

Specifically, the system involves voters and their electronic voting devices, a Ballot box (the central device), and a Screen. Moreover, a voter is chosen to take on the role of an Assessor (for example the president of the committee or her secretary). This is illustrated in Figure 3.1.

Ballot box. The Ballot box is the central device that collects the ballots and tallies the votes. It communicates with the electronic devices of the voters over private individual channels. Once the voting phase is over, the Ballot box publishes the outcome of the election on the Screen.

Screen. The Screen displays the outcome of the election for validation by the voters and the Assessor . Since the voters are in the same room, they all see the same Screen.

Voter. The voter role has two phases. In the first phase, he casts his vote through her electronic device. In the second phase, he performs some consistency checks looking at the Screen and lets the Assessor know whether his checks were successful, in which case he approves the procedure.

Personal voting device. Each individual voting device has a pad or some buttons for the voter to express her choice. The device communicates the value of the vote entered by the voter directly to the Ballot box.

Assessor. The Assessor is a role that can be performed by any voter. He does not hold any secret. He is chosen before the execution of the protocol. The Assessor is responsible of some additional verifications. In particular, he checks that each voter has approved the procedure. If one voter has not, he must cancel the vote and start a new one.

3.2 Face-to-face Voting System

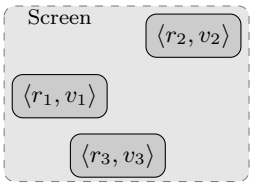
We describe in details the electronic boardroom voting system designed by the CNRS Section 07 committee. We actually present three versions of it. The three versions have in common the fact that the central device and/or the voters generate a random number that is attached to the vote. Both the vote and the random number are displayed on the Screen. This way, each voter can check that his vote (uniquely identified by its random number) is counted in the tally. We could have presented the version that offers the best security guarantees but we think the flaws in the other versions are of interest as well. The three versions differ in who generates the randomness:

- Initial version: The Ballot box generates the random identifier for each voter.
- Second version: Both the Ballot box and voters generate a random identifier.
- Third version: The voters generate their identifiers.

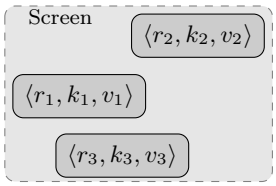
The three voting systems are summarized in Figure 3.2 and are described in details in the rest of the section. Since the votes are transmitted in clear to the central device on uniquely identified wires, ballot secrecy is clearly not guaranteed as soon as the central device is corrupted. So for ballot secrecy, we assume that the central device behaves honestly, that is, the secrecy of the ballots will be guaranteed only against external users (including the voters themselves). The major interest of the CNRS Section 07 system is that it ensures vote correctness *even if the central device is corrupted*, that is the voters do not need to trust any part of the infrastructure.

Note that in practice, the "random numbers" used in the remaining of the paper should typically be numbers of 3-4 digits, so that they are easy to copy and compare.

Initial version ($\mathsf{F2FV}^1$)
$B \to V_i : r_i$
$V_i \to B : \langle r_i, v_i \rangle$



Second version ($\mathsf{F2FV}^2$)
$B \to V_i : r_i$
$V_i \to B : \langle r_i, k_i, v_i \rangle$



Third version ($\mathsf{F2FV}^3$)
$V_i \to B : \langle k_i, v_i \rangle$

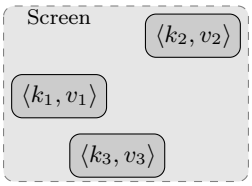


Figure 3.2: Voting processes

3.2.1 Initial System $\mathsf{F2FV}^1$

Voting Phase. The Ballot box B starts the election by generating a random number r for each voter V, and sends this random number to the voter. The voter V receives the random number r, uses it to form his ballot $\langle r, v \rangle$ where v is his vote, and sends his ballot to the Ballot box. Finally, all the ballots $\langle r, v \rangle$ are displayed on the Screen E. This marks the end of the voting process.

Validation Phase. The validation part can then begin. Each voter checks that his ballot is correctly included in the list of ballots displayed on the Screen. The Assessor waits for each voter to state that his vote appears on the Screen. He also checks that the number of ballots matches the number of voters. If all checks succeed, the Assessor approves the outcome of the election.

Possible Attacks

The key idea of this system is that each random identifier should be unique, ensuring a one-to-one correspondence between the votes that appear on the Screen and the votes cast by the voters. However, a corrupted Ballot box may still insert ballots of its choice, mounting a so-called "clash-attack" [KTV12]. The attack works as follows: the (dishonest) Ballot box guesses that two voters Alice and Bob are going to vote in the same way. (This could be a pure guess or based on statistical analysis of the previous votes.) The Ballot box then sends the *same* nonce r to Alice and Bob. Since Alice and Bob cast the same vote v, they both send back the same ballot $\langle r, v \rangle$. The Ballot box is then free to display $\langle r, v \rangle$ only once and then add any ballot of its choice. Both Alice and Bob would recognize $\langle r, v \rangle$ as their own ballot so the result would be validated.

For example, assume there are three voters A, B, and C and the Ballot box guesses that A and B vote identically. Suppose A and B cast 0 and C casts 1. The Ballot box can replace the two votes for 0 by one vote for 0 and one vote for 1, making the "1" vote win. This can be done by simply sending the same randomness r_a to both A and B.

$$
\begin{array}{lll}
B(I) \to V_A \ : \ r_a & B(I) \to V_B \ : \ r_a & B(I) \to V_C \ : \ r_c \\
V_A \to B(I) \ : \ \langle r_a, 0 \rangle & V_B \to B(I) \ : \ \langle r_a, 0 \rangle & V_C \to B(I) \ : \ \langle r_c, 1 \rangle \\
 & B(I) \to E \ : \ \langle r_a, 0 \rangle & \\
 & B(I) \to E \ : \ \langle r_b, 1 \rangle & \\
 & B(I) \to E \ : \ \langle r_c, 1 \rangle &
\end{array}
$$

3.2.2 Second System F2FV^2

The attack on the initial system F2FV^1 is due to the fact that the Ballot box may cheat when generating random unique identifiers. So a second solution has been proposed, where both the voters and the Ballot box generate a part of the random identifier.

Voting Phase. The Ballot box B starts the election by generating a random number r for each voter V, then sends this random number to the voter. The voter V receives the random number r, picks a new random number k (possibly using a pre-generated list), and uses it to form his ballot $\langle r, k, v\rangle$ where v is his vote, and then sends his ballot to the Ballot box. Finally, all the ballots $\langle r, k, v\rangle$ are displayed on the Screen E. The validation phase works like for the protocol F2FV^1.

Possible Attacks

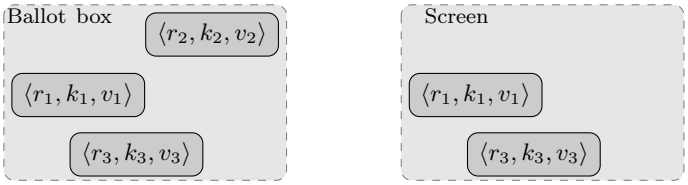
As we shall see in Section 3.5, this second version ensures vote correctness, even if the Ballot box is corrupted. As for the two other variants, ballot secrecy is not guaranteed as soon as the central device (the Ballot box) is corrupted. Indeed, the central device may leak how each voter has voted or may record it on some memory. However, such attacks against ballot secrecy assume a rather strong control of the Ballot box, where the attacker can access to the device either during or after the election. We further discuss some more subtle flaws which require a lower level of corruption. We describe two different attacks.

Encoding information in the randoms. As already mentioned, a fully corrupted Ballot box may transmit how each voter voted since it receives the votes in the clear, from uniquely identified wires. However, F2FV^2 (and F2FV^1) also suffers from offline attacks, where an attacker simply logs the election outcome. Indeed, it makes sense anyway to keep a copy of the Screen after each election. The attack works as follows. Instead of generating fully random numbers, the Ballot box could be programmed to provide a voter i (where i is the number identifying the voting device used by the voter) with a nonce r_i such that $r_i \equiv i \mod p$, where p is larger than the number of voters. In this way, an intruder could deduce from a ballot $\langle r, k, v\rangle$ the identity of the voter, simply by computing r modulo p. Of course, the identity of the voters could be encoded in the randomness in many other ways, making the detection of such an attack very unlikely. This attack simply assumes the attacker had access to the central device, at least once prior to the election (e.g. during its manufacturing). It does not require the attacker to access the Ballot box during nor after the election.

Swallowing ballots. There is a more direct (but easily detectable) way to break ballot secrecy, as sketched in Figure 3.3. Indeed, assume an attacker wants to know to whom a ballot $\langle r_2, k_2, v_2\rangle$ belongs to. In case the attacker simply controls the display of the Screen, he can send a modified set of ballots to the Screen. E.g. if he sends $\langle r_2, k_2, v_2'\rangle$ instead of $\langle r_2, k_2, v_2\rangle$), or if he simply remove this ballot, the voter who submitted the ballot $\langle r_2, k_2, v_2\rangle$ would then complain, revealing his identity.

Security Guarantees

We show in Section 3.5 that this second version ensures vote correctness, even if the Ballot box is corrupted. It also ensures ballot secrecy, assuming the Ballot box is honest.



The ballot $\langle r_2, k_2, v_2 \rangle$ is not sent on the Screen. Voter V_2 reports his ballot is missing, leaking how he voted to the attacker.

Figure 3.3: Attack against ballot secrecy.

3.2.3 Third System F2FV^3

To circumvent the ballot secrecy issue of the second system, when the Ballot box is somewhat honest (the attacker cannot access not interfere with it) but has been maliciously programmed, a third version has been proposed, where the random identifier is generated by the voter only. This version of the voting system is very similar to a protocol proposed by Bruce Schneier in [Sch95].

Voting Phase. Each voter V picks a random number k and uses it to form his ballot $\langle k, v \rangle$ where v is his vote, and then sends his ballot to the Ballot box. All the ballots $\langle k, v \rangle$ are displayed on the Screen E. The validation phase works like for systems F2FV^1 and F2FV^2.

Possible Attack

This third system is vulnerable to the same kind of attacks against vote correctness as the one described for system F2FV^1. Indeed, in case two voters pick the same random number and vote for the same candidate, for instance $(k_A, v_A) = (k_B, v_B)$, the Ballot box could remove one of these ballots and replace it by a ballot of its choice without being detected. Note that, due to the birthday theorem, it is not so unlikely that two voters use the same random number. For example, assume voters use 4 digits numbers. Then there is a probability of more than 0.2 to have a collision in a room of 67 members and more than 0.5 in a room of 118 members. In case, only 3 digits numbers are used, there is already a probability of collision of about 0.5 for only 37 members. These figures assume that the voters pick true random numbers. In case they generate numbers "manually", the entropy is usually much lower (e.g. users are sometimes reluctant to generate numbers with repeated digits). In such cases, the probability of collision increases accordingly.

Security Guarantees

We show in Section 3.5 that this third version ensures vote correctness, even if the Ballot box is corrupted (providing voters generate true randomness). It also ensures ballot secrecy, assuming the Ballot box is honest.

3.2.4 Common Weaknesses

If a voter claims that her ballot does not appear on the Screen, then the election round is canceled and everyone has to vote again. This means that a dishonest voter may choose to cancel an election (e.g. if she's not happy with the result), simply by wrongly claiming that her vote does not appear. This is mitigated by the fact that the advantage of the attack is small (the election just takes place again) and the voter could be blamed as being dishonest or inattentive if this happens too often.

3.3 Modeling the Protocols

The remaining of this chapter is devoted to the formal proof of security of ballot privacy and vote correctness for the two systems $\mathsf{F2FV}^2$ and $\mathsf{F2FV}^3$. We use the applied pi-calculus [AF01] (See Chapter 1) for the formal description of the voting systems.

3.3.1 Equational Theory

Since our voting systems do not use any cryptography, we adopt the following simple signature:

$$\Sigma_{pair} = \{\mathsf{ok}, \mathsf{fail}, \mathsf{fst}, \mathsf{snd}, \mathsf{pair}\}$$

where ok and fail are constants; fst and snd are unary functions and pair is a binary function. The term $\mathsf{pair}(m_1, m_2)$ represents the concatenation of two messages m_1 and m_2, while fst and snd represent the projections on the first and second component respectively.

The properties of the pair are modeled by the equational theory E_{pair} that states that it is possible to retrieve the two elements of a pair:

$$\mathsf{fst}(\mathsf{pair}(x, y)) = x \qquad \mathsf{snd}(\mathsf{pair}(x, y)) = y.$$

We consider equality modulo this equational theory, that is, equality of terms is the smallest equivalence relation induced by E_{pair}, closed under application of function symbols, substitution of terms for variables and bijective renaming of names. The corresponding rewriting system is convergent. (We don't have any AC property here.)

3.3.2 Modeling the Protocols in Applied Pi-calculus

We provide a formal specification of the two last variants of the CNRS voting system, in the applied pi-calculus. We do not describe the formal model of the initial voting system since it does not ensure ballot secrecy nor vote correctness.

We model the communications of the Ballot box with the voters and the Screen by secure channels (resp. $\mathsf{c_i}$ and $\mathsf{c_B}$). These channels may be controlled by the adversary when the Ballot box is corrupted. The voters and the Assessor look at the Screen. This communication cannot be altered and is modeled by an authenticated channel $\mathsf{c_{eyes}}$. The Assessor also communicates with each voter to check that the voter found his/her ballot on the Screen. This is again modeled by an authenticated channel $\mathsf{c_{A_i}}$ since we assume that voters cannot be physically impersonated. The channel connections are summarized in Figure 3.4.

Note. The applied-pi calculus provides an easy way to model both public and secure channel. Public channels are simply modeled by unrestricted names: the attacker can both read and send messages. Secure channels are modeled by restricted names: the attacker cannot read nor send any message on these channels. In contrast, an attacker may read authenticated channels but only authorized users may send messages on them. Since the applied pi-calculus does not provide us with a primitive for authenticated channels, we model authenticated channel by a secure channel, except that a copy of each emission is sent first on a public channel. In particular, we use the notation $\overline{\overline{c}}\langle M\rangle$ for $\overline{c_p}\langle M\rangle.\overline{c}\langle M\rangle$ with c_p a public channel.

Note. The role of the individual voting device is limited: it simply receives the vote from the voter and transmits it to the Ballot Box. W.l.o.g and for simplicity, we identify the voter and her individual device in the model of the voting systems.

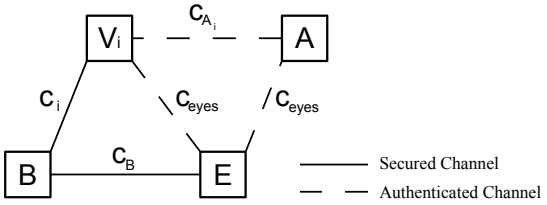


Figure 3.4: Channels between Players of the Protocol

Model of F2FV[2]

The process for the voter is parametrized by the number n of voters, its secure channel with the Ballot box c, its authenticated channel with the Screen (c_e) and the auditor (c_a), the public channel c_p and its vote v. The relation $a \in_n b$ refers to the fact that a appears in the n-tuple b.

$V_n(c, c_e, c_a, c_p, v) =$
- $\nu k \,.\, c(x)\,.$ % Creates fresh nonce and waits for input on c.
- $\overline{c}\langle\langle x, k, v\rangle\rangle\,.$ % Sends ballot on c to the Ballot box.
- $c_e(y)\,.$ % Waits for input on c_e (results on the Screen).
- if $\langle x, k, v\rangle \in_n y$ % Checks his vote.
- then $\overline{\overline{c_a}}\langle \mathsf{ok}\rangle$ else $\overline{\overline{c_a}}\langle \mathsf{fail}\rangle$ % Sends result on c_a to the Assessor.

The process for the Ballot box is parametrized by the number n of voters, the secure channels $c_v^1, \ldots, c_v^n$ with each voter and its secure channel with the Screen c_{be}.

$B_n(c_v^1, \ldots, c_v^n, c_b) =$
- $\nu r_1, \ldots, r_n\,.$ % Creates fresh randomness.
- $\overline{c_v^1}\langle r_1\rangle\,.\,\ldots\,.\,\overline{c_v^n}\langle r_n\rangle\,.$ % Sends randomness to voters.
- $c_v^1(y_1)\,.\,\ldots\,.\,c_v^n(y_n)\,.$ % Waits for inputs of ballots.
- $(\overline{c_b}\langle y_1\rangle \mid \cdots \mid \overline{c_b}\langle y_n\rangle)$ % Sends ballots in random order to E.

The Screen is modeled by a process E_n that simply broadcasts the result given by B_n. It is parametrized by the number n of voters, the authenticated channels c_e with each voter, the secure channel with the bulletin box c_b, and the public channel c_p.

$E_n(c_b, c_e, c_p) =$
- $c_b(t_1)\,.\,\ldots\,.\,c_b(t_n)\,.$ % Waits for votes from Ballot box.
- let $r = \langle t_1, \ldots, t_n\rangle$ in
- $\overline{c_p}\langle r\rangle\,.\,(!\;\overline{c_e}\langle r\rangle)$ % Displays info for all the boardroom.

The last role is the role of the Assessor. It is modeled by a process A_n that waits for the result displayed by the Screen and the confirmation of the voters. Then it verifies the outcome and validates the election if everything is correct. The process A_n is parametrized by the number n of voters, the authenticated channels $c_a^1, \ldots, c_a^n$ with each voter, the secure channel with the Screen c_e, and the public channel c_p.

$A_n(c_e, c_a^1, \ldots, c_a^n, c_p) =$
- $c_e(z')\,.$ % Waits to see result on the Screen.
- $c_a^1(z_1)\,.\,\ldots\,.\,c_a^n(z_n)\,.$ % Waits for decision of voters.
- if $\Psi_n(z', z_1, \ldots, z_n)$ % Checks if everything is fine.
- then $\overline{c_p}\langle \mathsf{ok}\rangle$ else $\overline{c_p}\langle \mathsf{fail}\rangle$ % Sends confirmation or rejection.

where $\Psi_n(p', p_1, \ldots, p_n) = (\bigwedge_{i=1}^{n} p_i = \mathsf{ok}) \wedge (p' = \langle \Pi_1^n(p'), \Pi_2^n(p'), \ldots, \Pi_n^n(p') \rangle)$. The test Ψ_n ensures that each voter approved the vote ($p_i = \mathsf{ok}$) and that the result contains as many ballots than the number of voters.

Finally the system $\mathsf{F2FV}^2$ is represented by the voter's role V_n and the voting context:

$$P_n^2[\ \_\ ] = \nu\ \widetilde{\omega}.[\ \_\ |B_n(\mathsf{c}_1, \ldots, \mathsf{c}_n, \mathsf{c}_B)|E_n(\mathsf{c}_B, \mathsf{c}_{eyes}, \mathsf{c}_{out})|A_n(\mathsf{c}_{eyes}, \mathsf{c}_{A_1}, \ldots, \mathsf{c}_{A_n}, \mathsf{c}_{out})]$$

where $\widetilde{\omega} = (\mathsf{c}_1, \ldots, \mathsf{c}_n, \mathsf{c}_{A_1}, \ldots, \mathsf{c}_{A_n}, \mathsf{c}_B, \mathsf{c}_{eyes})$ are restricted channels (c_{out} is public).

Model of the Protocol $\mathsf{F2FV}^3$

The third protocol only differs from the second one by the fact that the Ballot box does not generate any randomness. Therefore, the models of the Screen and of the Assessor are unchanged. The voter and Ballot box models are modified as follows.

$$\begin{array}{ll} V'_n(c, c_e, c_a, c_p, v) = & B'_n(c_v^1, \ldots, c_v^n, c_b) = \\ \quad \nu k\ .\ \overline{c}\langle\langle k, v\rangle\rangle\ .\ c_e(x)\ . & \quad c_v^1(y_1)\ .\ \ldots\ .\ c_v^n(y_n)\ . \\ \quad \text{if } \langle k, v\rangle \in_n x \text{ then } \overline{\overline{c_a}}\langle\mathsf{ok}\rangle \text{ else } \overline{\overline{c_a}}\langle\mathsf{fail}\rangle & \quad (\overline{c_b}\langle y_1\rangle \mid \cdots \mid \overline{c_b}\langle y_n\rangle) \end{array}$$

The system $\mathsf{F2FV}^3$ is represented by the voter's role V'_n and the voting context:

$$P_n^3[\ \_\ ] = \nu\ \widetilde{\omega}.\left[\ \_\ |B'_n(\mathsf{c}_1, \ldots, \mathsf{c}_n, \mathsf{c}_B)|E_n(\mathsf{c}_B, \mathsf{c}_{eyes}, \mathsf{c}_{out})|A_n(\mathsf{c}_{eyes}, \mathsf{c}_{A_1}, \ldots, \mathsf{c}_{A_n}, \mathsf{c}_{out})\right]$$

where $\widetilde{\omega} = (\mathsf{c}_1, \ldots, \mathsf{c}_n, \mathsf{c}_{A_1}, \ldots, \mathsf{c}_{A_n}, \mathsf{c}_B, \mathsf{c}_{eyes})$ are restricted channels.

3.4 Ballot Secrecy

For the study, we consider two cases depending on whether the Ballot box is corrupted or not. We always assume the Screen to be honest. This is however not a limitation. Indeed, requiring the Screen to be honest reflects the fact that everyone sees the same Screen, which is always the case for people in the same room.

As in Chapter 2, we use a well established definition of ballot secrecy that has been formalized in terms of equivalence by Delaune, Kremer and Ryan in [DKR09]. A protocol with voting process $V(v, id)$ and authority process A preserves *ballot secrecy* if an attacker cannot distinguish when votes are swapped, i.e. it cannot distinguish when a voter a_1 votes v_1 and a_2 votes v_2 from the case where a_1 votes v_2 and a_2 votes v_1. This is formally specified by :

$$\nu\tilde{n}.\,(A \mid V\{^{v_2}/_x, ^{a_1}/_y\} \mid V\{^{v_1}/_x, ^{a_2}/_y\}) \approx_l \nu\tilde{n}.\,(A \mid V\{^{v_1}/_x, ^{a_1}/_y\} \mid V\{^{v_2}/_x, ^{a_2}/_y\})$$

where $\tilde{n}$ represents the data (keys, nonces, channels, ...) initially shared between the authority and the voters.

3.4.1 Results

We describe here the two main results of ballot secrecy we have on $\mathsf{F2FV}^2$ and $\mathsf{F2FV}^3$.

Ballot Secrecy for Voting Protocol $\mathsf{F2FV}^2$

The voting protocol $\mathsf{F2FV}^2$ preserves ballot secrecy, even when all but two voters are dishonest, provided that the Ballot box, the Screen and the Assessor are honest. For the sake of clarity, we use the following notation for the i^{th} voter: $V^i(v) = V_n(\mathsf{c_i}, \mathsf{c_{eyes}}, \mathsf{c_{A_i}}, \mathsf{c_{out}}, v)$.

Theorem 3.1. *Let $n \in \mathbb{N}$, let (P_n^2, V_n) be the process specification for n voters of the voting protocol $\mathsf{F2FV}^2$ as defined in Section 3.3.2, and let a, b be two names. Then*

$$P_n^2\left[V^1(a) \mid V^2(b)\right] \approx_l P_n^2\left[V^1(b) \mid V^2(a)\right].$$

Sketch of Proof. The proof of Theorem 3.1 consists in two main steps as in Chapter 2. First we build a relation $\mathcal{R}$ such that

$$P_n^2\left[V^1(a) \mid V^2(b)\right] \;\mathcal{R}\; P_n^2\left[V^1(b) \mid V^2(a)\right]$$

and such that for any two processes $P\ \mathcal{R}\ Q$, any move of P can be matched by a move of Q such that the resulting processes remain in relation. This amounts to characterizing all possible successors of $P_n^2\left[V^1(a) \mid V^2(b)\right]$ and $P_n^2\left[V^1(b) \mid V^2(a)\right]$. The second step of the proof consists in showing that the sequences of messages observed by the attacker are equal (due to the shuffle performed by the Ballot box). The details of this proof can be found in Section 3.4.2. □

Ballot Secrecy for Voting Protocol $\mathsf{F2FV}^3$

Similarly, the voting protocol $\mathsf{F2FV}^3$ preserves ballot secrecy, even when all but two voters are dishonest, provided that the Ballot box, the Screen and the Assessor are honest.

Theorem 3.2. *Let $n \in \mathbb{N}$, let (P_n^3, V'_n) be the process specification for n voters of the voting protocol $\mathsf{F2FV}^3$ as defined in Section 3.3.2, and let a, b be two names. Then*

$$P_n^3\left[V'^1(a) \mid V'^2(b)\right] \approx_l P_n^3\left[V'^1(b) \mid V'^2(a)\right].$$

Proof. The proof of Theorem 3.2 is adapted from the proof of Theorem 3.1. □

3.4.2 Proving Ballot Secrecy

In this Section, we detail the different steps of the proof of Theorem 3.1. The sketch is identical to the one used to prove the ballot secrecy results for the Norwegian protocol in Chapter 2.

Partial Evolutions

First, we introduce partial evolutions of the protocol process specification.

Definition 3.1. *Notations and partial evolutions for honest voters $V(c, c_e, c_a, c_p, v)$.*

$$\begin{aligned}
V_1 &= c(x)\ .\ V_2 \\
V_2 &= \overline{c}\langle\langle x, k, v\rangle\rangle\ .\ V_3 \\
V_3 &= c_e(x')\ .\ V_4 \\
V_4 &= \mathit{if}\ \langle x, k, v\rangle \in_n x'\ \mathit{then}\ V_5\ \mathit{else}\ V_7 \\
V_5 &= \overline{c_p}\langle\mathsf{ok}\rangle\ .\ V_6 \qquad & V_7 &= \overline{c_p}\langle\mathsf{fail}\rangle\ .\ V_8 \\
V_6 &= \overline{c_a}\langle\mathsf{ok}\rangle & V_8 &= \overline{c_a}\langle\mathsf{fail}\rangle
\end{aligned}$$

Definition 3.2. *Notations and partial evolution for the Ballot box* $B_n(\mathsf{c_1},\ldots,\mathsf{c_n},\mathsf{c_B})$*, the Screen* $E_n(\mathsf{c_B},\mathsf{c_{eyes}},\mathsf{c_{out}})$ *and the Assessor* $A_n(\mathsf{c_{eyes}},\mathsf{c_{A_1}},\ldots,\mathsf{c_{A_n}},\mathsf{c_{out}})$*.*

$$\begin{aligned}
B_1^{i,n} &= \overline{\mathsf{c_i}}\langle r_i\rangle \,.\, B_1^{i+1,n} \text{ for } i<n\\
B_1^{n,n} &= \overline{\mathsf{c_n}}\langle r_n\rangle \,.\, B_2^{1,n}\\
B_2^{i,n} &= \mathsf{c_i}(y_i) \,.\, B_2^{i+1,n} \text{ for } i<n\\
B_2^{n,n} &= \mathsf{c_n}(y_n) \,.\, B_3^{[\![1,n]\!]}\\
B_3^S &= \mid_{j\in S} \overline{\mathsf{c_B}}\langle y_j\rangle \text{ with } S\in 2^{[\![1,n]\!]} \text{ and, by convention, } B_3^{\emptyset}=0.
\end{aligned}$$

Note that we have the following relation : $\forall\ j\in S, B_3^S = \overline{\mathsf{c_B}}\langle y_j\rangle \mid B_3^{S\setminus\{j\}}$.

$$\begin{aligned}
E_1^{i,n} &= \mathsf{c_B}(t_i) \,.\, E_1^{i+1,n} \text{ for } i<n\\
E_1^{n,n} &= \mathsf{c_B}(t_n) \,.\, E_2\\
E_2 &= \overline{\mathsf{c_{out}}}\langle r\rangle \,.\, E_3 \qquad \text{with } r=\langle t_1,\ldots,t_n\rangle.\\
E_3 &= \overline{\mathsf{c_{eyes}}}\langle r\rangle \,.\, E_3
\end{aligned}$$

$$\begin{aligned}
A_1 &= \mathsf{c_{eyes}}(z') \,.\, A_{2.1}^{1,n}\\
A_2^{i,n} &= \mathsf{c_{A_i}}(z_i) \,.\, A_2^{i+1,n}\\
A_2^{n,n} &= \mathsf{c_{A_n}}(z_n) \,.\, A_3\\
A_3 &= \text{if } \Psi_n(z',z_1,\ldots,z_n) \text{ then } A_4 \text{ else } A_5\\
A_4 &= \overline{\mathsf{c_{out}}}\langle\mathsf{ok}\rangle \qquad\qquad A_5 = \overline{\mathsf{c_{out}}}\langle\mathsf{fail}\rangle
\end{aligned}$$

Now we introduce partial evolutions of the global process:

Definition 3.3. *We define two main substitutions* $\sigma_L = \{^a/_{v_1}, ^b/_{v_2}\}$ *and* $\sigma_R = \{^b/_{v_1}, ^a/_{v_2}\}$ *to summarize voting choices made by honest voters during the processes. We also define* $\theta_i = \{^{\mathsf{c_i}}/_c, ^{\mathsf{c_{eyes}}}/_{c_e}, ^{\mathsf{c_{A_i}}}/_{c_a}, ^{\mathsf{c_{out}}}/_{c_p}, ^{v_i}/_v\}$ *and* $V_j^i = V_j\theta_i$*. We consider the following notations for different sets of names:*

$$\begin{aligned}
\widetilde{\omega} &= k_1,k_2,\mathsf{c_1},\ldots,\mathsf{c_n},\mathsf{c_{A_1}},\ldots,\mathsf{c_{A_n}},\mathsf{c_{eyes}},\mathsf{c_B}\\
\widetilde{r} &= \widetilde{\omega},r_1,\ldots,r_n\\
\widetilde{y} &= \widetilde{r},y_1,\ldots,y_n\\
\widetilde{t_S} &= t_1,\ldots,t_i \text{ for } i=\#\overline{S} \text{ where } \overline{S}=[\![1,n]\!]\setminus S \text{ with } S\in 2^{[\![1,n]\!]}.\\
\widetilde{t} &= \widetilde{y},t_1,\ldots,t_n\\
\widetilde{x'}_{i,j} &= \begin{cases} x'_1 & \text{if } i\geq 4, j\leq 3\\ x'_2 & \text{if } i\leq 3, j\geq 4\\ x'_1,x'_2 & \text{if } i\geq 4, j\geq 4\end{cases}
\end{aligned}$$

Let us consider the following notations for different frames:

$$\begin{aligned}
\Lambda_i &= \{^{r_j}/_{x_j} \mid j=3..i\}\\
\Gamma &= \Lambda_n \mid \{^{\langle r_1,k_1,v_1\rangle}/_{y_1}, ^{\langle r_2,k_2,v_2\rangle}/_{y_2}\}\\
\Omega_{\delta_S}^S &= \Gamma \mid \{^{y_{\delta_S(i)}}/_{t_i} \mid i=1..\#\overline{S}\} \text{ with } \delta_S:\overline{S}\mapsto[\![1,\#\overline{S}]\!].\\
\Omega_\delta &= \Gamma \mid \{^{y_{\delta(1)}}/_{t_1},\ldots,^{y_{\delta(n)}}/_{t_n}\} \text{ with } \delta \text{ a permutation of } [\![1,n]\!],\\
\Upsilon_{i,j}^\delta &= \Omega_\delta \mid \{^r/_{out_r}\} \mid \{^{\langle t_1,\ldots,t_n\rangle}/_{x'_1}, \text{if } i\geq 4\} \mid \{^{\langle t_1,\ldots,t_n\rangle}/_{x'_2}, \text{if } j\geq 4\}\\
&\quad \mid \{^{\mathsf{ok}}/_{out_1}, \text{if } i\geq 6\} \mid^{\mathsf{ok}}/_{out_2}, \text{if } j\geq 6\}
\end{aligned}$$

And, now, partial evolution of the global process:

$$
\begin{aligned}
C_0\,[\_] &= \nu\widetilde{\omega}.\,[\_]\\
C_1^{i,j}\,[\_] &= \nu(\widetilde{\omega}, r_1, \ldots, r_i, x_1, \ldots, x_j, k_1, k_2).\,[\_ \mid \Lambda_i]\\
C_2^{i}\,[\_] &= \nu(\widetilde{r}, x_1, x_2, y_1, \ldots, y_i).\,[\_ \mid \Gamma]\\
C_3^{S}\,[\_] &= \nu(\widetilde{y}, x_1, x_2, \widetilde{t_S}).\,\left[\_ \mid \Omega_{\delta_S}^{S}\right]\\
C_4^{i,j,\delta}\,[\_] &= \nu(\widetilde{t}, x_1, x_2, \widetilde{x'}_{i,j}).\,\left[\_ \mid \Upsilon_{i,j}^{\delta}\right]\\
C_5^{i,j,\delta}\,[\_] &= \nu(\widetilde{t}, x_1, x_2, \widetilde{x'}_{i,j}, z').\,\left[\_ \mid \Upsilon_{i,j}^{\delta} \mid \{^{\langle t_1,\ldots,t_n\rangle}/_{z'}\}\right]\\
C_6^{j,\delta}\,[\_] &= \nu(\widetilde{t}, x_1, x_2, \widetilde{x'}_{6,j}, z').\,\left[\_ \mid \Upsilon_{6,j}^{\delta} \mid \{^{\langle t_1,\ldots,t_n\rangle}/_{z'}, {}^{\mathsf{ok}}/_{z_1}\}\right]\\
C_7^{\delta}\,[\_] &= \nu(\widetilde{t}, x_1, x_2, x'_1, x'_2, z', z_1, z_2).\,[\_ \mid \Upsilon_{6,6}^{\delta} \mid \{^{\langle t_1,\ldots,t_n\rangle}/_{z'}, {}^{\mathsf{ok}}/_{z_1}, {}^{\mathsf{ok}}/_{z_2}\}]
\end{aligned}
$$

Using all these notations, we have that:

$$
\begin{aligned}
P_n^2\,[V_1(a) \mid V_2(b)] &= C_0\left[V_1^1 \mid V_1^2 \mid B_1^{1,n} \mid E_1 \mid A_1\right]\sigma_L,\\
P_n^2\,[V_1(b) \mid V_2(a)] &= C_0\left[V_1^1 \mid V_1^2 \mid B_1^{1,n} \mid E_1 \mid A_1\right]\sigma_R.
\end{aligned}
$$

Relation $\mathcal{R}$

Finally we can introduce $\mathcal{R}$.

Definition 3.4. *Given integer $n \geq 2$, let $M_3, \ldots, M_n$ be terms such that for every permutation δ of $[\![1, n]\!]$, $\forall\ 3 \leq j \leq n, \mathsf{fv}(M_j) \subseteq \mathsf{dom}(C_7^{\delta})$ and $\mathsf{fn}(M_j) \cap \mathsf{bn}(C_7^{\delta}) = \emptyset$. We note $\widetilde{M_i} = \{^{M_j}/_{y_j} \mid j = 3..i\}$. We consider the smallest relation $\mathcal{R}$ closed under structural equivalence and that includes the following relations between extended processes:*

$$C_0\left[V_1^1 \mid V_1^2 \mid B_1^{1,n} \mid E_1^{1,n} \mid A_1\right]\sigma_L\ \mathcal{R}\ C_0\left[V_1^1 \mid V_1^2 \mid B_1^{1,n} \mid E_1^{1,n} \mid A_1\right]\sigma_R \tag{3.1}$$

$$C_1^{1,1}\left[V_2^1 \mid V_1^2 \mid B_1^{2,n} \mid E_1^{1,n} \mid A_1\right]\sigma_L\ \mathcal{R}\ C_1^{1,1}\left[V_2^1 \mid V_1^2 \mid B_1^{2,n} \mid E_1^{1,n} \mid A_1\right]\sigma_R \tag{3.2}$$

For all $2 \leq i \leq n-1$:

$$C_1^{i,2}\left[V_2^1 \mid V_2^2 \mid B_1^{i+1,n} \mid E_1^{1,n} \mid A_1\right]\sigma_L \qquad \mathcal{R}\ C_1^{i,2}\left[V_2^1 \mid V_2^2 \mid B_1^{i+1,n} \mid E_1^{1,n} \mid A_1\right]\sigma_R \tag{3.3}$$

$$C_1^{n,2}\left[V_2^1 \mid V_2^2 \mid B_2^{1,n} \mid E_1^{1,n} \mid A_1\right]\sigma_L \qquad \mathcal{R}\ C_1^{n,2}\left[V_2^1 \mid V_2^2 \mid B_2^{1,n} \mid E_1^{1,n} \mid A_1\right]\sigma_R \tag{3.4}$$

$$C_1^{n,2}\left[V_3^1 \mid V_2^2 \mid B_2^{2,n} \mid E_1^{1,n} \mid A_1 \mid \{^{\langle r_1,k_1,v_1\rangle}/_{y_1}\}\right]\sigma_L \qquad \mathcal{R}\ C_1^{n,2}\left[V_3^1 \mid V_2^2 \mid B_2^{2,n} \mid E_1^{1,n} \mid A_1 \mid \{^{\langle r_1,k_1,v_1\rangle}/_{y_1}\}\right]\sigma_R \tag{3.5}$$

For all $2 \leq i \leq n-1$:

$$C_2^{i}\left[V_3^1 \mid V_3^2 \mid B_2^{i+1,n} \mid E_1^{1,n} \mid A_1 \mid \widetilde{M_i}\right]\sigma_L \qquad \mathcal{R}\ C_2^{i}\left[V_3^1 \mid V_3^2 \mid B_2^{i+1,n} \mid E_1^{1,n} \mid A_1 \mid \widetilde{M_i}\right]\sigma_R \tag{3.6}$$

For all $S \in 2^{[\![1,n]\!]} \setminus \{\emptyset\}$, ${}^tS = S\{1 \mapsto 2,\ 2 \mapsto 1\}$, ${}^t\overline{S} = \overline{S}\{1 \mapsto 2,\ 2 \mapsto 1\}$, $\delta_S : \overline{S} \to [\![1, \#\overline{S}]\!]$ *and* $\delta_{{}^tS} : {}^t\overline{S} \to [\![1, \#{}^t\overline{S}]\!]$ *with* $\delta_{{}^tS} = [1 \mapsto 2, 2 \mapsto 1] \circ \delta_S$*:*

$$C_3^S \left[V_3^1 \mid V_3^2 \mid B_3^S \mid E_1^{\#\overline{S}+1,n} \mid A_1 \mid \widetilde{M}_n\right] \sigma_L \quad \mathcal{R} \quad C_3^{{}^tS} \left[V_3^1 \mid V_3^2 \mid B_3^{{}^tS} \mid E_1^{\#{}^t\overline{S}+1,n} \mid A_1 \mid \widetilde{M}_n\right] \sigma_R \tag{3.7}$$

$$C_3^{\emptyset} \left[V_3^1 \mid V_3^2 \mid E_2 \mid A_1 \mid \widetilde{M}_n\right] \sigma_L \ \mathcal{R}\ C_3^{{}^t\emptyset} \left[V_3^1 \mid V_3^2 \mid E_2 \mid A_1 \mid \widetilde{M}_n\right] \sigma_R \tag{3.8}$$

Let δ *be a permutation of* $[\![1, n]\!]$ *and* ${}^t\delta$ *be such that* ${}^t\delta = \{1 \mapsto 2, 2 \mapsto 1\} \circ \delta$ *and* $i, j \in [\![3, 6]\!]$*:*

$$C_4^{i,j,\delta} \left[V_i^1 \mid V_j^2 \mid E_3 \mid A_1 \mid \widetilde{M}_n\right] \sigma_L \ \mathcal{R}\ C_4^{i,j,{}^t\delta} \left[V_i^1 \mid V_j 2 \mid E_3 \mid A_1 \mid \widetilde{M}_n\right] \sigma_R \tag{3.9}$$

$$C_5^{i,j,\delta} \left[V_i^1 \mid V_j^2 \mid E_3 \mid A_2^{1,n} \mid \widetilde{M}_n\right] \sigma_L \ \mathcal{R}\ C_5^{i,j,{}^t\delta} \left[V_i^1 \mid V_j^2 \mid E_3 \mid A_2^{1,n} \mid \widetilde{M}_n\right] \sigma_R \tag{3.10}$$

$$C_6^{j,\delta} \left[V_j^2 \mid E_3 \mid A_2^{2,n} \mid \widetilde{M}_n\right] \sigma_L \ \mathcal{R}\ C_6^{j,{}^t\delta} \left[V_j^2 \mid E_3 \mid A_2^{2,n} \mid \widetilde{M}_n\right] \sigma_R \tag{3.11}$$

$$C_7^{\delta} \left[E_3 \mid A_2^{3,n} \mid \widetilde{M}_n\right] \sigma_L \ \mathcal{R}\ C_7^{{}^t\delta} \left[E_3 \mid A_2^{3,n} \mid \widetilde{M}_n\right] \sigma_R \tag{3.12}$$

Now we must show that for all extended processes A and B where $A \ \mathcal{R}\ B$, if $A \longrightarrow A'$ (resp. $A \xrightarrow{\alpha} A'$), then $B \longrightarrow^* B'$ (resp. $B \xrightarrow{\alpha}{}^* B'$) and $A' \ \mathcal{R}\ B'$ for some B'. We proceed by case analysis. The proof is straightforward for most of the cases, except for (3.7) for which we provide a detailed proof:

Proof. We are in a case where there exists $S \in 2^{[\![1,n]\!]} \setminus \emptyset$ such that

$$\begin{cases} A \equiv C_3^S[V_3^1 \mid V_3^2 \mid B_3^S \mid E_1^{\#\overline{S}+1,n} \mid A_1 \mid \widetilde{M}_n]\sigma_L \\ B \equiv C_3^{{}^tS}[V_3^1 \mid V_3^2 \mid B_3^{{}^tS} \mid E_1^{\#{}^t\overline{S}+1,n} \mid A_1 \mid \widetilde{M}_n]\sigma_R \end{cases}$$

For the purpose of this proof, we will use the notation $E_1^{n+1,n}$ as an alias to E_2. Note that this is coherent with our relation. Consider A' such that $A \longrightarrow A'$. There exists $j \in [\![1, n]\!]$ such that $j \in S$, and

$$\begin{cases} A \equiv C_3^S \left[V_3^1 \mid V_3^2 \mid \left(\overline{\mathsf{c_B}}\langle y_j\rangle \mid B_3^{S\setminus\{j\}}\right) \mid \mathsf{c_B}(y'_{\#\overline{S}+1}).E_1^{\#\overline{S\setminus\{j\}}+1,n} \mid A_1 \mid \widetilde{M}_n\right] \sigma_L \\ A' \equiv C_3^{S\setminus\{j\}} \left[V_3^1 \mid V_3^2 \mid B_3^{S\setminus\{j\}} \mid E_1^{\#\overline{S\setminus\{j\}}+1,n} \mid A_1 \mid \widetilde{M}_n\right] \sigma_L \end{cases}$$

First we note that we have $\#{}^tS = \#S$, and thus, $\#{}^t\overline{S} = \#\overline{S}$. Moreover, if $j \notin \{1, 2\}$, then we have ${}^t(S \setminus \{j\}) = {}^tS \setminus \{j\}$.

We distinguish now two cases :

- If $j \notin \{1, 2\}$, then it follows that $j \in {}^tS$ and
  $B \equiv C_3^{{}^tS} \left[V_3^1 \mid V_3^2 \mid \left(\overline{\mathsf{c_B}}\langle y_j\rangle \mid B_3^{{}^tS\setminus\{j\}}\right) \mid \mathsf{c_B}(y'_{\#\overline{{}^tS}+1}).E_1^{\#\overline{{}^tS\setminus\{j\}}+1,n} \mid A_1 \mid \widetilde{M}_n\right] \sigma_R.$
  $B \longrightarrow B'$ where $B' = C_3^{{}^tS\setminus\{j\}} \left[V_3^1 \mid V_3^2 \mid B_3^{{}^tS\setminus\{j\}} \mid E_1^{\#\overline{{}^tS\setminus\{j\}}+1,n} \mid A_1 \mid \widetilde{M}_n\right] \sigma_R$. In that case, according to the definition of $\mathcal{R}$ we still have $A' \ \mathcal{R}\ B'$.

- If $j \in \{1,2\}$. We suppose w.l.o.g. that $j = 1$. Since ${}^tS = S\{1 \mapsto 2, 2 \mapsto 1\}$ and $j = 1 \in S$, then $2 \in {}^tS$. We can then write B as
$B \equiv C_3^{tS}\left[V_3^1 \mid V_3^2 \mid \left(\overline{\mathsf{c_B}}\langle y_2\rangle \mid B_3^{{}^tS\setminus\{2\}}\right) \mid \mathsf{c_B}(y'_{\#\overline{{}^tS}+1}).E_1^{\#\overline{{}^tS\setminus\{2\}}+1,n} \mid A_1 \mid \widetilde{M}_n\right]\sigma_R$.

  $B \longrightarrow B'$ with $B' = C_3^{{}^tS\setminus\{2\}}\left[V_3^1 \mid V_3^2 \mid B_3^{{}^tS\setminus\{2\}} \mid E_1^{\#\overline{{}^tS\setminus\{2\}}+1,n} \mid A_1 \mid \widetilde{M}_n\right]\sigma_R$

  As ${}^tS \setminus \{2\} = {}^t(S \setminus \{1\})$ and according to the definition of $\mathcal{R}$, we still have $A' \mathcal{R} B'$.

Note. It is quite easy to see that $\delta_{S\setminus\{j\}}$ can be well defined using δ_S and properties between $\delta_{S\setminus\{j\}}$ and $\delta_{{}^tS\setminus\{j\}}$ still holds in both cases (modulo the same permutation between 1 and 2 if needed).

□

To show that the relation $\mathcal{R}$ enables us to characterize all possible successors of $P_n^2\,[V_1(a) \mid V_2(b)]$ and $P_n^2\,[V_1(b) \mid V_2(a)]$, we now prove that the two processes in rule (3.12) are, in fact, the same, implying that their successors will always be in equivalence.

Proof. We consider two processes A and B such that

$$\begin{cases} A = C_7^{\delta}\left[E_3 \mid A_2^{3,n} \mid \widetilde{M}_n\right]\sigma_L \\ B = C_7^{{}^t\delta}\left[E_3 \mid A_2^{3,n} \mid \widetilde{M}_n\right]\sigma_R \end{cases}$$

Let us consider the frame $\Phi_\delta = C_7^{\delta}\,[\emptyset] = \nu\varepsilon.\phi_\delta$ considering that $\varepsilon = (\tilde{\omega}, r_1, r_2, k_1, k_2, t_1, \ldots, t_n, x_1, x_2, x_1', x_2', z', t_1, t_2)$ and $\phi_\delta = \{{}^{\mathsf{ok}}/_{z_1}, {}^{\mathsf{ok}}/_{z_2}\} \mid \{{}^{y_{\delta(1)}}/_{t_1}, \ldots, {}^{y_{\delta(n)}}/_{t_n}\} \mid \{{}^{\langle t_1,\ldots,t_n\rangle}/_{z'}, {}^{\langle t_1,\ldots,t_n\rangle}/_{x_1'}, {}^{\langle t_1,\ldots,t_n\rangle}/_{x_2'}, {}^{\mathsf{ok}}/_{out_1}, {}^{\mathsf{ok}}/_{out_2}\} \mid \theta'$ with $\theta' = \{{}^{\langle r_1,k_1,v_1\rangle}/_{y_1}, {}^{\langle r_2,k_2,v_2\rangle}/_{y_2}, {}^{M_3}/_{y_3}, \ldots, {}^{M_n}/_{y_n}\}$.

We want to show that $\Phi_\delta\theta'\sigma_L = \Phi_{{}^t\delta}\theta'\sigma_R$. One can see that is ensured if and only if

$$\overline{\Phi}_\delta\theta'\sigma_L = \overline{\Phi}_{{}^t\delta}\theta'\sigma_R \qquad (\star)$$

with $\overline{\Phi}_\delta = \nu\varepsilon.\{{}^{y_{\delta(1)}}/_{t_1}, \ldots, {}^{y_{\delta(n)}}/_{t_n}\}$.

Now, consider the renaming $\tau = \{r_1 \mapsto r_2, r_2 \mapsto r_1, k_1 \mapsto k_2, k_2 \mapsto k_1\}$ and let $\varepsilon' = \varepsilon\tau$. In that case, we have that $\varepsilon' = \varepsilon$ and we show that :

$$\nu\varepsilon.\{{}^{y_{\delta(1)}}/_{t_1}, \ldots, {}^{y_{\delta(n)}}/_{t_n}\}\theta'\sigma_L = \nu\varepsilon'.\{{}^{y_{{}^t\delta(1)}}/_{t_1}, \ldots, {}^{y_{{}^t\delta(n)}}/_{t_n}\}\theta'\tau\sigma_R. \qquad (\star\star)$$

Let $i \in [\![1,n]\!]$, we have that $t_i\theta'\sigma_L = y_{\delta(i)}\theta'\sigma_L$. Then, since δ is a permutation of $[\![1,n]\!]$, we have several cases depending on the value of $\delta(i)$:

- If $\delta(i) = 1$, $y_{\delta(i)}\theta'\sigma_L = y_1\theta'\sigma_L = \langle r_1, k_1, a\rangle$. But we have $\langle r_1, k_1, a\rangle = y_2\theta'\tau\sigma_R$ according to the renaming of nonces and, moreover, $y_2 = y_{{}^t\delta(i)}$ according to the definition of ${}^t\delta$. Then, using that $y_{{}^t\delta(i)}\theta'\tau\sigma_R = t_i\theta'\tau\sigma_R$, we can conclude that $t_i\theta'\sigma_L = t_i\theta'\tau\sigma_R$.

- If $\delta(i) = 2$, $y_{\delta(i)}\theta'\sigma_L = y_2\theta'\sigma_L = \langle r_2, k_2, b\rangle$. Using similar arguments as in the previous case (essentially switching 1 and 2), we can conclude that $t_i\theta'\sigma_L = t_i\theta'\tau\sigma_R$.

- If $\delta(i) = j$ where $j \in [\![3,n]\!]$, $y_{\delta(i)}\theta'\sigma_L = y_j\theta'\sigma_L = M_j$. According to the definition of ${}^t\delta$, we have that $\delta(i) = {}^t\delta(i)$ for $\delta(i) \in [\![3,n]\!]$. Then $y_{{}^t\delta(i)}\theta'\sigma_R = y_j\theta'\sigma_R = M_j$ and we can conclude that $t_i\theta'\sigma_L = t_i\theta'\tau\sigma_R$.

Thus, we have $(\star\star)$ from which we deduce $(\star)$ which allows us to conclude that the two frames and, therefore, A and B are equal.

□

Static Equivalence

The proof showing that the final sequences of messages observed by the attacker are equals mainly comes from the previous proof since we show that frames are the same at a certain point of the evolution of the two processes. Then, at the end of them, the two frames will be equal.

3.5 Vote Correctness

In this Section, we discuss about the property of vote correctness. After providing a definition of it, we show that $\mathsf{F2FV}^2$ and $\mathsf{F2FV}^3$ verify this property.

3.5.1 Definition

We define vote correctness as the fact that the election result should contain the votes of the honest voters. Formally, we assume that the voting protocol records the published outcome of the election t in an event $\mathsf{event}(t)$.

Definition 3.5 (Correctness)**.** *Let n be the number of registered voters, and m be the number of honest voters. Let $v_1, \ldots, v_m \in \mathcal{N}$ be the votes of the honest voters. Let $V^1, \ldots, V^m$ be the processes representing the honest voters. Each V^i is parametrized by its vote v_i. Let P_n be a context representing the voting system, besides the honest voters. We say that a voting specification $(P_n, \tilde{V})$ satisfies* vote correctness *if, for every $v_1, \ldots, v_m$, and for every execution of the protocol leading to the validation of a result t_r, i.e. of the form*

$$P_n[V^1(v_1)| \ldots |V^m(v_m)] \rightarrow^* \nu \tilde{n} \cdot (\mathsf{event}(t_r) \cdot Q \mid Q'),$$

for some names $\tilde{n}$ and processes Q, Q', then there exist votes $v_{m+1}, \ldots, v_n$ and a permutation τ of $[\![1, n]\!]$ such that $t_r = \langle v_{\tau(1)}, \ldots, v_{\tau(n)} \rangle$, that is, the outcome of the election contains all the honest votes plus some dishonest ones.

To express vote correctness in the context of the CNRS Section 07 voting system, we simply add an event that records the tally, at the end of the process specification of the Assessor (see Section 3.5.3 for the corresponding modified process A'_n). We show vote correctness for a strong corruption scenario, where even the Ballot box is corrupted. Formally, we consider the following context that represents the three voting systems, the only difference between the systems now lying in the definition of voters.

$$P_n{}'[\ \_\ ] = \nu\ \widetilde{\omega}.\ [\ \_\ \mid E_n(\mathsf{c_B}, \mathsf{c_{eyes}}, \mathsf{c_{out}}) \mid A'_n(\mathsf{c_{eyes}}, \mathsf{c_{A_1}}, \ldots, \mathsf{c_{A_n}}, \mathsf{c_{out}})]$$

where $\widetilde{\omega} = (\mathsf{c_{A_1}}, \ldots, \mathsf{c_{A_n}}, \mathsf{c_{eyes}})$, which means that the intruder has access in this scenario to channels $\mathsf{c_1}, \ldots, \mathsf{c_n}$ and $\mathsf{c_B}$ in addition to $\mathsf{c_{out}}$.

To illustrate the correctness property, let first show that $\mathsf{F2FV}^1$ does not satisfy vote correctness when the Ballot box is corrupted. First, we introduce $\hat{V}$ the process of an honest voter in the voting process $\mathsf{F2FV}^1$:

$$\hat{V}(c, c_e, c_a, c_p, v) = c(x)\ .\ \overline{c}\langle\langle x, v\rangle\rangle\ .\ c_e(y)\ .\ \text{if } \langle x, v\rangle \in_n y \text{ then } \overline{\overline{c_a}}\langle\mathsf{ok}\rangle \text{ else } \overline{\overline{c_a}}\langle\mathsf{fail}\rangle$$

Let $\hat{V}^i = \hat{V}\{^{\mathsf{c_i}}/_c, ^{\mathsf{c_{eyes}}}/_{c_e}, ^{\mathsf{c_{A_i}}}/_{c_a}, ^{\mathsf{c_{out}}}/_{c_p}\}$. It represents the i-th honest voter. Suppose now, that the first m honest voters cast the some vote: $\forall i \in [\![1, m]\!],\ v_i = v$. We show how the attack described in Section 3.2.1 is reflected. Each honest voter receives the same random number r:

$$P'_n[\hat{V}^1(v_1) \mid \cdots \mid \hat{V}^m(v_m)] \xrightarrow{\forall i \in [\![1,m]\!],\ \overline{c_i}\langle r\rangle} P'_n[\hat{V}^1_r(v_1) \mid \cdots \mid \hat{V}^m_r(v_m)]$$

where $\hat{V}_r^i(v_i) = \overline{\mathsf{c_i}}\langle\langle r, v_i\rangle\rangle \,.\, \mathsf{c_{eyes}}(y) \,.$ if $\langle r, v_i\rangle \in_n y_i$ then $\overline{\overline{\mathsf{c_{A_i}}}}\langle\mathsf{ok}\rangle$ else $\overline{\overline{\mathsf{c_{A_i}}}}\langle\mathsf{fail}\rangle$. Then, the honest voters output their vote on channels $\mathsf{c_1}, \ldots, \mathsf{c_m}$ which will always be $\langle r, v\rangle$.

$$P'_n[\hat{V}_r^1(v_1) \mid \cdots \mid \hat{V}_r^m(v_m)] \xrightarrow{\forall i \in [\![1,m]\!],\ \overline{c_i}\langle\langle r,v_i\rangle\rangle} P'_n[\hat{V}_e^1(v_1) \mid \cdots \mid \hat{V}_e^m(v_m)]$$

where $\hat{V}_e^i(v_i) = \mathsf{c_{eyes}}(y) \,.$ if $\langle r, v_i\rangle \in_n y_i$ then $\overline{\overline{\mathsf{c_{A_i}}}}\langle\mathsf{ok}\rangle$ else $\overline{\overline{\mathsf{c_{A_i}}}}\langle\mathsf{fail}\rangle$. Corrupted voters also submit their votes (which is transparent in transitions) and we move to the next phase: the corrupted Ballot box just has to output one of the honest votes to the Screen and $n-1$ other votes. Thus, the final tally t_r showed by the Screen will contain only one $\langle r, v\rangle$ but each honest voters will send ok to the Assessor since their test will succeed anyway. In that case, we would have $P'_n[\hat{V}^1(v_1)|\ldots|\hat{V}^m(v_m)] \rightarrow^* \nu\tilde{n}\cdot\mathsf{event}(t_r)$ for some $\tilde{n}$, but, clearly, t_r is not satisfying the property of the Definition 3.5 since it only contains one vote v instead of m votes v.

3.5.2 Results

In contrast, the two voting systems $\mathsf{F2FV}^2$ and $\mathsf{F2FV}^3$ satisfy vote correctness, even when the Ballot box is corrupted, assuming that the voters check that their ballots appear on the Screen.

Theorem 3.3. *The voting specifications (P'_n, V) and (P'_n, V') satisfy vote correctness.*

Sketch of Proof. The Assessor records the result of the election in an event only if $\Psi_n(p', p_1, \ldots, p_n)$ holds. This formula intuitively represents the fact that every voter has told to the Assessor that his ballot was included in the tally, and that the number of ballots in the tally matches the number of voters, i.e. n. Using this information and the fact that each honest voter has generated a random nonce uniquely identifying his ballot, we can show that the voting specifications satisfy vote correctness. □

Correctness requires that at least one person in the room checks that no one has complained and that the number of displayed ballots correspond to the number of voters. If no one performs these checks then there is no honest Assessor and correctness is no longer guaranteed.

3.5.3 Proof of Correctness

Since the proofs for the two voting specifications are very similar, we only present the proof of the correctness of $\mathsf{F2FV}^2$.

Assessor Specification for Vote Correctness

The role of the Assessor is slightly modified to enable us to check whether the behavior of the protocol was correct and if the outcome matches what can be expected from the votes of the honest voters. The modification is the following one: when all the verifications (of voters and Assessor) have been carried out, after validating the result, i.e. sending message ok on channel c_{out}, the Assessor triggers an event with a predicate $\mathsf{outcome}(t_r)$ expressing the fact that result t_r has been validated. The correctness of the protocol depends on the matching between this outcome and the votes cast by honest voters. Formally, the role of the Assessor is now defined in the following way:

$$
\begin{aligned}
&A'_n(c_e, c_a^1, \ldots, c_a^n, c_p) = \\
&\qquad c_e(z') \ . \\
&\qquad c_a^1(z_1) \ . \ \ldots \ . \ c_a^n(z_n) \ . \\
&\qquad \text{if } \Psi_n(z', z_1, \ldots, z_n) \\
&\qquad \text{then } \overline{c_p}\langle \mathsf{ok} \rangle.\mathsf{event}(\mathsf{votes}(z')) \\
&\qquad \text{else } \overline{c_p}\langle \mathsf{fail} \rangle
\end{aligned}
$$

where $\Psi_n(p', p_1, \ldots, p_n) = \left(\bigwedge_{i=1}^{n} (p_i = \mathsf{ok}) \right) \wedge (p' = \langle \Pi_1^n(p'), \ldots, \Pi_n^n(p') \rangle)$ and

$$
\mathsf{votes}(b_1, \ldots, b_n) = \begin{cases} \langle \Pi_3(b_1), \ldots, \Pi_3(b_n) \rangle \text{ for voting protocol } \mathsf{F2FV}^2, \\ \langle \Pi_2(b_1), \ldots, \Pi_2(b_n) \rangle \text{ for voting protocol } \mathsf{F2FV}^3. \end{cases}
$$

Useful Lemmas

The proof relies on two lemmas. The first one states that any (validated) outcome necessarily contains all the honest ballots.

Lemma 3.1. *Let n be the number of voters, and $m \leq n$ be the number of honest voters. Assume that $v_1, \ldots, v_m$ are votes cast by the honest voters. Consider an execution of the voting specification (P'_n, V) that ends with the validation of a result by the Assessor:*

$$
P'_n[\ V^1(v_1) | \ldots | V^m(v_m)\ ] \rightarrow^* \nu\ \widetilde{\omega}.\ \cdot\ (\mathsf{event}(t_r) |\ Q)
$$

Then there exists pairwise distinct nonces $k_1, \ldots k_m \in \widetilde{\omega}$, nonces $r'_1, \ldots, r'_m$ and M, a n-tuple such that $\langle r'_i, k_i, v_i \rangle \in_n M$ for all $1 \leq i \leq m$ and $\mathsf{votes}(M) = t_r$.

Proof of Lemma 3.1. Suppose we have such an execution, i.e.

$$
\begin{aligned}
&\nu\ \widetilde{\omega}.[\ V^1(v_1) \mid \cdots \mid V^2(v_n) \mid E_n(\mathsf{c_B}, \mathsf{c_{eyes}}, \mathsf{c_{out}}) \mid \\
&\qquad A'_n(\mathsf{c_{eyes}}, \mathsf{c_{A_1}}, \ldots, \mathsf{c_{A_n}}, \mathsf{c_{out}})] \qquad \rightarrow^* \qquad \nu\ \widetilde{\omega}.\ \cdot\ (\mathsf{event}(t_r) |\ Q).
\end{aligned}
$$

The event can only be triggered if $\Psi_n(\overline{z}\sigma, z_1\sigma, \ldots, z_n\sigma)$ is true, where:

$$
\Psi_n(\overline{p}, p_1, \ldots, p_n) = \left(\bigwedge_{i=1}^{n} (p_i = \mathsf{ok}) \right) \wedge (\overline{p} = \langle \Pi_1^n(\overline{p}), \ldots, \Pi_n^n(\overline{p}) \rangle)
$$

and $\overline{z}\sigma, z_1\sigma, \ldots, z_n\sigma$ are the values given to $\overline{z}, z_1, \ldots, z_n$ by inputs on secure channels c_{ae} and c_a^i for $i = 1..n$. Furthermore, we have that $t_r = \mathsf{votes}(\overline{z}\sigma)$. Thus, we have that:

- $\forall i \leq n,\ z_i\sigma = \mathsf{ok}$,
- $\overline{z}\sigma = \langle \Pi_1^n(\overline{z}\sigma), \ldots, \Pi_n^n(\overline{z}\sigma) \rangle$,
- $t_r = \mathsf{votes}(\overline{z}\sigma)$.

For every $1 \leq i \leq n$, we have that $z_i\sigma = \mathsf{ok}$. Each z_i is instantiated by the Assessor when he receives a message from voter V_i over (secure) channel $\mathsf{c_{A_i}}$. If voter V_i is honest, i.e. $i \leq m$, then he outputs ok on channel $\mathsf{c_{A_i}}$ only after checking that his ballot appears in the list of ballots displayed on the Screen, or more formally if $\langle r'_i, k_i, v_i \rangle \in \overline{x_i}\sigma$ where $\overline{x_i}\sigma$ is the input voter i

receives from the Screen on channel $\mathsf{c_{eyes}}$, r'_i is the random nonce it receives on channel $\mathsf{c_i}$, and k_i is a unique nonce he generates to include in his ballot (formally, this means that $k_i \in \widetilde{\omega}$).

Let $r\sigma$ be the term that the Screen outputs on channel $\mathsf{c_{eyes}}$. As channel $\mathsf{c_{eyes}}$ is secured, we have that $r\sigma = \overline{z}\sigma$ and for every $i \leq m$, $\overline{x_i}\sigma = r\sigma$. Note that $\overline{z}\sigma = \langle \Pi_1^n(\overline{z}\sigma), \ldots, \Pi_n^n(\overline{z}\sigma)\rangle = r\sigma = \overline{z}'\sigma = \overline{x_i}\sigma$, thus $r\sigma$ is an n-tuple.

To sum up, there exist pairwise distinct nonces $k_1, \ldots, k_m \in \widetilde{\omega}$, nonces $r'_1, \ldots, r'_m$ such that $\forall\ 1 \leq i \leq m$, $\langle r'_i, k_i, v_i\rangle \in_n r\sigma$, $r\sigma$ is an n-tuple and $t_r = \mathsf{votes}(r\sigma)$. □

The second lemma simply says that if a unique randomness is used in each ballot, the honest ballots are all counted.

Lemma 3.2. *Let M_n be a n-tuple such that $\langle r_i, k_i, v_i\rangle \in M_n$ for all $1 \leq i \leq m$, where the k_i are pairwise distinct. Then there exist $v_{m+1}, \ldots, v_n, \tau$ such that*

$$\mathsf{votes}(M_n) = \langle v_{\tau(1)}, \ldots, v_{\tau(n)}\rangle$$

Proof. We have that M is an n-tuple of ballots, so there exist $r'_1, \ldots, r'_n, k'_1, \ldots, k'_n, v'_1, \ldots, v'_n$ such that:

$$M = \langle \langle r'_1, k'_1, v'_1\rangle, \ldots, \langle r'_m, k'_m, v'_m\rangle\rangle.$$

As $\bigwedge_{i=1}^{m} (\langle r_i, k_i, v_i\rangle \in M)$, there exists a function $f : [\![1, m]\!] \to [\![1, n]\!]$ such that if $f(i) = j$, then $\langle r_i, k_i, v_i\rangle = \langle r'_j, k'_j, v'_j\rangle$.

Let us show that f is one-to-one. Let $i, i' \in [\![1, m]\!]$ such that $f(i) = f(i') = j$. Then, by definition of f, $\langle r_i, k_i, v_i\rangle = \langle r'_j, k'_j, v'_j\rangle = \langle r_{i'}, k_{i'}, v_{i'}\rangle$. Then in particular $k_i = k_{i'}$, and so we deduce $i = i'$ by using formula $(k_i = k_{i'}) \Rightarrow (i = i')$. Consequently, we can extend f as a bijection over $[\![1, n]\!]$, and so there exist votes $v_{m+1}, \ldots, v_n$ and a permutation τ such that:

$$\langle v_{\tau(1)}, \ldots, v_{\tau(n)}\rangle = \mathsf{votes}(M).$$

□

Proof of Theorem 3.3

We are now ready to prove Theorem 3.3. Assume that $v_1, \ldots, v_m$ are votes cast by the honest voters and consider an execution of the protocol with a validated outcome.

$$P'_n[\ V^1(v_1) | \ldots | V^m(v_m)\ ] \to^* \nu\ \widetilde{\omega}.(\mathsf{event}(t_r) |\ Q).$$

Let us show that there exist votes $v_{m+1}, \ldots, v_n$ and a permutation τ of $[\![1, n]\!]$ such that $t_r = v_{\tau(1)}, \ldots, v_{\tau(n)}$. First, applying Lemma 3.1, we have that such a sequence of transitions can only happen if there exist pairwise distinct nonces $k_1, \ldots k_m \in \widetilde{\omega}$, nonces $r'_1, \ldots, r'_m$ and M a n-tuple such that $\langle r'_i, k_i, v_i\rangle \in_n M$ for all $1 \leq i \leq m$ and $\mathsf{votes}(M) = t_r$.

Then, using Lemma 3.2, we get that there exist votes $v_{m+1}, \ldots, v_n$ and permutation τ such that $v_{\tau(1)}, \ldots, v_{\tau(n)} = t_r$. Thus the protocol ensures vote correctness.

Table 3.1: Results for the $\mathsf{F2FV}^1$,$\mathsf{F2FV}^2$, and $\mathsf{F2FV}^3$ protocols. A ✓ indicates provable security while × indicates an attack. We assume an arbitrary number of dishonest voters.

Results	ballot secrecy			Correctness		
Corr. Players / System	None	Ballot Box	Assessor	None	Ballot Box	Assessor
$\mathsf{F2FV}^1$	✓	×	✓	✓	×	×
$\mathsf{F2FV}^2$	✓	×	✓	✓	✓	×
$\mathsf{F2FV}^3$	✓	×	✓	✓	✓	×

3.6 Summary & Discussion

A summary of our findings is displayed on Table 3.1. The proofs of correctness of $\mathsf{F2FV}^2$ and $\mathsf{F2FV}^3$ in the honest case follow from the proofs in the dishonest case. Ballot secrecy is not affected by a corrupted Assessor as it actually only performs public verification. So its corruption does not provide any extra power to the attacker. Ballot secrecy and correctness for $\mathsf{F2FV}^1$ (in the honest case) follow from the proofs for $\mathsf{F2FV}^2$.

We believe that the voting system proposed by the CNRS Section 07 committee for boardroom meetings is an interesting protocol that improves over existing electronic devices. We have analyzed the security of three possible versions, discovering some interesting flaws. We think that the two last versions are adequate since they both preserve ballot secrecy and vote correctness. The choice between the two versions depends on the desired compromise between ballot secrecy and vote correctness: the second version ensures better correctness but less privacy since the randomness generated by the Ballot box may leak the identity of the voters. Conversely, the third system offers better privacy but slightly less assurance about vote correctness, in case the voters do not use proper random identifiers.

In both cases, vote correctness is guaranteed as soon as:

- Voters really use (unpredictable) random numbers. In practice, voters could print (privately and before the meeting) a list of random numbers that they would use at their will (erasing a number once used). This list of random numbers could typically be generated using a computer. Alternatively, voters may also bring dice to the meeting.

- Each voter casts a vote (possibly blank or null) and checks that his vote (and associated randomness) appears on the Screen.

Correctness does not require any trust on the devices while privacy does. This is unavoidable unless the communication between the voters and the Ballot box would be anonymized, which would require a much heavier infrastructure. Note that the system is not fair if the Ballot box is compromised since dishonest voters may then wait for honest voters to cast their votes, before making their own decision.

A weakness of the system relies in the fact that a voter may force to re-run an election by (wrongly) claiming that her vote does not appear on the Screen. As already mentioned in

Section 3.2.4, this is mitigated by the fact that the voter could then be blamed if this happens too often. This also means that an honest voter could be blamed if a dishonest Ballot Box intentionally removes her ballot at each turn. It would be interesting to devise a mechanism to mitigate this issue. Another weakness: this system does not provide coercion-resistance. Indeed, a voter may be forced, by an attacker, to use specific "random" codes, such that, when the ballots are published on the screen, the attacker (or someone inside the meeting room colluding with him) could check that the voter acts as expected.

Part II

Formal Analysis of APIs

Chapter 4

Revocation API for Symmetric Keys

Contents

Embedded systems deployed in hostile environments often employ some dedicated tamper-resistant secure hardwares to handle cryptographic operations and keep keys secure. Examples include mobile phones (which contain SIM cards), smartphones (recent models include "Secure Elements"), public transport ticketing systems (such as the Calypso system which employs smartcards and "SAM" modules [Lev10]), smart utility meters (that include a smartcard-like chip for

cryptography), on-vehicle cryptographic devices to support vehicle-to-vehicle networking [Wey11] *et cetera*. In such systems, it is often necessary to support the possibility of remotely revoking and updating the long-term keys on the device. However, while extensive research addresses the problem of establishing a new session key or determining what security properties can be guaranteed in the event of long-term key corruption, relatively little work has appeared addressing the problem of revocation and update of long-term keys.

Most existing solutions for key revocation follow one of two approaches: either key revocation actually relies on some "longer term" key that cannot be itself revoked, or key revocation is simply achieved by disabling, resetting or isolating the compromised device. For many applications, both these approaches are unsatisfactory. We propose that a key revocation API should ideally satisfy the following properties:

1. *The device should remain functional* - specifically, whenever possible, the device should return to an equivalent functional state after revocation (but with fresh keys in place).

2. *Any key should be revocable* - side-channel attacks may compromise (perhaps with significant effort) any of the keys stored on a cryptographic device, and the more sensitive a key is, the more likely an attacker is to dedicate effort to breaking it. Hence it is not prudent to decide in advance which keys may or may not be compromised.

In this chapter, we describe the design of a new API, inspired from an existing API for symmetric key management [CS09], with update and revocation functionalities that satisfy these properties. We also provide a formal security proof of this design and detail the different security properties it satisfies. The chapter is organized as follows. We describe the revocation problem and the different issues in Section 4.1. Section 4.2 details the different commands which are the core of the design of our API. Then, we provide, in Section 4.3, the formal model we used to conduct our analysis of the design. Next, we state and prove all the different security properties satisfied by the API in Section 4.4. Finally, Section 4.5 presents our implementation of our design using the Java Card language.

4.1 Revocation API Design Issues

This section describes the context of this work by introducing the different existing (but not satisfying) solutions to the problem of revocation and the main issues we try to address in the rest of this chapter.

4.1.1 Background

Most deployed key revocation mechanisms are quite simple. For example, the Trusted Platform Module [Tru11] supports a `TPM_OwnerClear` command that resets the 'ownership' status of the device and erases most keys such as the root key for the storage hierarchy. It does not support, for example, the revocation of individual keys in the hierarchy. In multicast group key management schemes, hierarchies are also standard, and here revocation of a key corresponds either to permanent removal of an entity from the group, or removal until the entity resubscribes, in which case long term secrets are assumed to be still intact. Our work will not make this assumption. One lesson from this domain is that many proposals published without proofs of security have turned out to have attacks [SB06].

Richer key revocation schemes have received some previous attention in the academic literature, e.g. in the key management schemes of wireless sensor networks. Here nodes in the

network are expected to be deployed in hostile environments where key compromise may occur. Eschenauer and Gligor propose a scheme under which every node i shares a long-term symmetric key K_{Ci} with a control server [EG02]. If a node n is compromised, the server sends a fresh signature key S_i to each node i sharing keys with n encrypted under K_{Ci}, and then signs a list of keys to be revoked and sends this list to each i. The nodes then delete the keys, isolating the compromised node. This scheme has two disadvantages for the general case: first it assumes that a central authority knows the key identifiers of all the keys in the network and between what parties they are shared, and second it has no way to recover a device which has lost its K_{Ci} to the attacker. An alternative scheme is KeyRev [YW07], which proposes the use of a secret sharing scheme to distribute session keys only to unrevoked nodes, thereby isolating compromised nodes without explicitly revoking their long term keys. Again this has the disadvantage that the loss of a long term key means the unrecoverable loss of a device: acceptable in a wireless sensor network perhaps but not in the general case. In this paper we aim to be able to securely update all keys on the device, provided not too many keys have been compromised.

The Sevecom API [Kar09] is a proposal for an on-board tamper-resistant device to handle cryptography in next-generation vehicles supporting VANETs (vehicle to vehicle ad-hoc networks). It includes two root public keys which are used to check authenticity of messages coming from a central server. The signed messages from the central server are used to trigger updates of working keys. The Sevecom API is interesting because it allows update of the long term root keys using a simple two step protocol. We will examine this example in more details in the next section.

There are other proposals for key management APIs with security proofs in the literature, but none of these address the question of revocation [CC09, CM06, FS09].

4.1.2 Revocation Issues

We set the following requirements for our design: (1) the device should return to full functionality whenever possible, and (2) all secret values should be revocable. Resources within the tamper-resistant boundary of an embedded crypto device are usually tightly restricted. We therefore design our API to use a minimum of memory (instead of storing lists of blacklisted keys, we blacklist sets of keys through their attributes) and cryptographic functionality (just symmetric encryption using an authenticated encryption scheme). We do however require that the devices contain loosely synchronized real-time clocks. This assumption is standard in some domains, e.g. on-vehicle cryptographic units [Kar09], Smart-meters, but is not suitable for every application - we will comment briefly in Section 4.6 on how to adapt our design to devices without clocks.

Our starting point is a generalization of the symmetric key management API of Cortier and Steel [CS09], which defines a simple hierarchy on keys under management whereby keys higher in the hierarchy are allowed to encrypt (wrap) keys lower down, for storage outside the device or as part of a key exchange protocol. We describe the API fully in the next section. Here we note some design constraints that are imposed on us by our key revocation requirements, since these may be of independent interest to future designers of revocation APIs.

Firstly, since we can have no single upper bound to the key hierarchy in order to satisfy requirement (2), our design must incorporate a set of keys of the same level Max which can revoke each others. The Sevecom API mentioned in Section 4.1.1 tackles this by having two root keys, where knowledge of the private halves of both is required in order to revoke and update any one of them. The update protocol is given in Figure 4.1, where the terms to the left of the semi-colon represent protocol messages, and the terms to the right are predicates on the state of the device. Initially, the device is in a "two key state", with root keys k_1 and k_2 available for use.

Then a revoke message is given for key k_1, which consists of k_1 signed under its own private half $inv(k_1)$. The device receives this and deletes k_1, moving into a "one key state" where no further revocation messages will be processed. Then an update message is given, using k_2 to sign the new root key k_3.

$$\begin{aligned}
&\mathsf{RevokeRootKey1}: \\
&\{k_1\}_{inv(k_1)}; \mathsf{keys}(k_1, k_2) \quad \rightarrow \quad ; \mathsf{keys}(_, k_2) \\
&\mathsf{UpdateRootKey1}: \\
&\{k_3\}_{inv(k_2)}; \mathsf{keys}(_, k_2) \quad \rightarrow \quad ; \mathsf{keys}(k_2, k_3)
\end{aligned}$$

Figure 4.1: Sevecom Revocation Protocol (simplified)

At first it may seem that this protocol achieves its security goal: unless the intruder manages to obtain access to both $inv(k_1)$ and $inv(k_2)$, he cannot replace a key with one of his own. However, an analysis by Mödersheim and Modesti showed that there is another attack scenario [MM11]: suppose an attacker had corrupted only $inv(k_2)$, and then waited until the server sent a revocation message for k_1. He could then inject his own root key k_i using $inv(k_2)$ to sign it. In a footnote, Mödersheim and Modesti propose that both signature keys be used to sign both messages, and the current public keys be included in the message to prevent replays of old updates. They did not verify this solution however. In our scheme we generalize the idea to a scheme where some number N_{Max} of the K level Max keys are required in order to revoke and replace one. We similarly require the revocation message to demonstrate knowledge of the current key in order to avoid replays, but we do this by encrypting under the old key rather than adding it to the plaintext, to prevent possible key cycles. Since the old key may be corrupted, it is the innermost encryption. A revocation message, e.g. "replace k_i with k_j", will therefore have the following form:

$$\{\ldots\{\{\mathsf{update}, k_j\}_{k_i}\}_{k_1}\ldots\}_{k_n}$$

In our setting, we only have symmetric key cryptography available, so unlike the public keys in Sevecom, our root keys must remain confidential. This imposes an upper limit to the security we can achieve: if at any point in the future, it is possible for the attacker to obtain all the keys $k_1, \ldots, k_n$, then he will obtain also k_j, and by repeating this for all the subsequent key updates be able to obtain N_{Max} of the current level Max keys. To prevent this, for our security proofs, we demand that "honest" updates to the level Max keys, i.e. updates generated by the server, remain unknown to the attacker. In practice this could mean that the level Max keys are only updated when the device is connected to a trusted host. If this is too cumbersome for the application then a quantitative risk assessment would have to be undertaken to set the N and K high enough to achieve the required degree of security. A full solution would require asymmetric cryptography, as we will discuss in Section 4.6. Note that we still prove that the device resist attacks against fake update messages constructed by the intruder, even when he has corrupted individual level Max keys.

For keys of lower level than Max, we assume that the attacker can see all (encrypted) update messages. We need some way to ensure freshness of messages containing keys encrypted by other keys, otherwise revocation will be ineffective. To see this, consider the following example: Assume $\{..., k_3, ...\}_{k_5}$ and $\{..., k_3, ...\}_{k_4}$ have been sent out through the network, with k_i at level $l_i < \mathsf{Max}$ in the hierarchy. Assume that at some point k_4 is lost to the attacker. Then he learns k_3 as well (by decrypting the second message). Our revocation API would allow us to remove

k_4, and we should also remove k_3 since it must be assumed to be lost as well. However, this will not suffice: the attacker could replay the key distribution message $\{..., k_3, ...\}_{k_5}$, and therefore re-inject the corrupted k_3 into the device. To avoid this one could blacklist all keys below a corrupted key, which is problematic (this blacklist will take up device memory and will have to be kept indefinitely), or revoke all the keys in the hierarchy all the way to the top (impractical), or assure some kind of freshness of key distribution messages. We choose the latter option, requiring that every key is given a validity time when it is issued. After the expiry date passes, the device will refuse to use the key. Without this property, we would need some other way to ensure freshness of all messages, such as challenge-response exchanges for every message.

Finally, note that though we design our API to use memory efficiently, we do not explicitly analyse its resistance to denial of service attacks.

4.2 Description of the API

We describe the design and commands for our key management and key revocation API.

4.2.1 Setting

We assume that keys are structured in a hierarchy, such that a key k_1 may encrypt a key k_2 only if k_1 is greater than k_2. More precisely, we assume a set of levels $\mathcal{L}$ equipped with a (partial) order $<$, a maximal element Max and minimal element 0. Management keys used for revocation or updates will be keys of maximum level Max, and knowledge of at least N_{Max} of them will be required for many operations. We also assume that a level can only be compared to finite number of levels. More precisely, we assume that for any $l \in \mathcal{L}$, the set $\{l' \mid l' < l\}$ is finite. This will ensure that we have no infinite sequence of the form $\{\dots k_1 \dots\}_{k_2}$, $\{\dots k_2 \dots\}_{k_3}$, $\dots$

We further assume that each tamper resistant device (TRD) a has:

- a clock, whose current time is given by $t_a \in \mathcal{T}$ where $\mathcal{T}$ is an infinite ordered set of time events. For example, $\mathcal{T}$ may be $\mathbb{R}^+$, the set of non negative real numbers. For simplicity, we will assume that all clocks are synchronized with a global clock, referred to as the time of the global system. Our security properties could be adapted to take account of clock drift, provided some limit on the drift is assured.

- a table Θ_a of keys. Each entry in the table is indexed by a *handle* h and the corresponding entry $\Theta_a(h)$ is (k, l, v, m) where k is the actual key stored on the TRD, l is its corresponding level, v its *validity date* of the key, and $m \in \mathcal{M}$ is a *miscellaneous field* that may describe some other attributes of the key (e.g. describing the purpose of the key).

- a blacklist $\mathfrak{B}_a$ of elements of the form (l, t) where l is a level and t is an expiration time. Intuitively, whenever (l, t) occurs in a blacklist, it means that the TRD will never accept a key of level l (or below), unless time t has now passed.

Intuitively, the design of our API will ensure that a key may only encrypt other keys lower in the hierarchy whose validity dates have not expired.

We also assume that the TRDs share a function $\delta : \mathcal{L} \to \mathcal{T}$, that associates lifetime to keys depending on their levels. We may sometimes abbreviate $\delta(l)$ by δ_l. Since we have assumed that a level can only be compared to finite number of levels, we can compute the maximum lifetime

of a chain of levels, smaller than a given level. Formally, we consider the function:

$$\Delta : l \mapsto \max_{l_1 < \cdots < l_n < l, n \in \mathbb{N}} \sum_{i=1}^{n} \delta(l_i).$$

Intuitively, $\Delta(l)$ is the time where compromising a key k of level l may compromise keys of lower levels, even if the validity time of k has expired, due to chains of encryptions (see Section 4.4).

For initialization, we assume that each device contains at least N_{Max} keys of level Max which are known to an administrator. From this, using the commands of the API, the administrator can bootstrap the system and, if necessary, update all the level Max keys.

4.2.2 Commands

We first give the standard commands which do not concern revocation or keys of level Max, but suffice for normal key management operations. They are mostly the same as in the original API [CS09], where they were shown to be sufficient to implement a number of key establishment protocols while always keeping sensitive keys in the secure memory of the TRD, never exposing them in the clear on the host machine. We have generalized the ordering on the hierarchy and introduced checking of the validity time of keys and a blacklist $\mathfrak{B}_a$ of key levels together with expiration times. We will write $l \in \mathfrak{B}_a$ if there is $(l', t) \in \mathfrak{B}_a$ with $l \leq l'$ and t an expiration time. We also assume a test $\mathsf{Distinct}(h_1, \ldots, h_n)$ which checks that the h_i are pairwise distinct.

Public generation

The generation command for public (level 0) data (e.g a fresh public nonce), which takes an optional argument $m \in \mathcal{M}$, is defined as follows. It returns both the value and the handle which points to the value stored on the device.

$\mathsf{generatePublic}(m)$
- $\mathsf{let}\ h = \mathsf{Fresh}(H_a)\ \mathsf{in}$ % Creation of a new fresh handle.
- $H_a := H_a \cup \{h\}$
- $\mathsf{let}\ n = \mathsf{Fresh}(N)\ \mathsf{in}$ % Generation of the new public value (nonce).
- $N := N \cup \{n\}$
- $\mathsf{let}\ v = t + \delta(0)\ \mathsf{in}$ % Estimation of the validity date for n.
- $\Theta_a := \Theta_a \cup \{h \to (n, 0, v, m)\}$ % Storage of the information in the device.
- $\mathsf{return}\ h, n$ % Output of the handle and the value to the user.

where $\mathsf{Fresh}(E)$ (respectively with $E \in \{H_a, N, K\}$) returns a element of the set $\mathcal{E} \setminus E$ (respectively with $\mathcal{E} \in \{\mathcal{H}_a, \mathcal{N}, \mathcal{K}\}$).

Secret generation

The generation command for the creation of a secret key, which inputs the level of security l, and an optional argument $m \in \mathcal{M}$. Since the key needs to remain hidden, the API only returns the handle pointing to the generated key.

$\mathsf{generateSecret}(l, m)$
- $\mathsf{if}\ 0 < l < \mathsf{Max}$
  - $\mathsf{let}\ h = \mathsf{Fresh}(H_a)\ \mathsf{in}$ % Creation of a new fresh handle.

$H_a := H_a \cup \{h\}$
let $k = \mathsf{Fresh}(K)$ in % Generation of the new key.
$K := K \cup \{k\}$
let $v = t_a + \delta(l)$ in % Estimation of the validity date for k.
$\Theta_a := \Theta_a \cup \{h \to (k, l, v, m)\}$ % Storage of the information in the device.
return h % Output of the handle to the user.

Encryption

The encryption command takes as input a list of data that are meant to be encrypted (which may include handles pointing to values stored on the device) and a handle for the key k that will be used for encryption. We check that k has not expired and is not below a level which has been blacklisted. For each term to be encrypted, we check that the level is lower than that of k, the expiration date is valid, and that the level is not blacklisted.

encrypt$([X_1, \ldots, X_n], h)$
let $(k, l, v, m) = \Theta_a(h)$ in % Retrieve what is stored under h.
if $(l = \mathsf{Max}) \vee (v \leq t_a) \vee (l \in \mathfrak{B}_a)$ % Checks on k.
break % If checks fail, command encrypt stops.
for $i = 1..n$ (break aborts the entire command)
% Case 1: Message
if $X_i = M_i, m_i$
$Y_i := (M_i, 0, t_a + \delta(0), m_i)$ % Message prepared for encryption.
% Case 2: Handle
if $X_i = h_i$ /* Case Handle */
let $(k_i, l_i, v_i, m_i) = \Theta_a(h_i)$ in % Retrieve what is stored under h_i.
if $(l_i < l) \wedge (v_i > t_a) \wedge (l \notin \mathfrak{B}_a)$ % Checks on k_i.
$Y_i := (k_i, l_i, v_i, m_i)$ % Content of h_i prepared for encryption.
else break
return $\{Y_1, \ldots, Y_n\}_k$ % Output of the expected encryption.

Decryption

The decryption command takes as inputs a handle for the key that will be used for decryption and a cipher-text. We also assume that decrypt throws break on failure of authentication (i.e. we assume an authenticated encryption scheme). Note that the checks on the levels etc. performed during encryption are repeated. This is important for security in the presence of corrupted keys.

decrypt(C, h)
let $(k, l, v, m) = \Theta_a(h)$ in % Retrieve what is stored under h.
if $(l = \mathsf{Max}) \vee (v \leq t_a) \vee (l \in \mathfrak{B}_a)$ % Checks on k.
break % If checks fail, decrypt stops.
let $X_1, \ldots, X_n = \mathsf{dec}(k, C)$ in % Attempt of decryption.
for $i = 1..n$ (break if fail)
let $(k_i, l_i, v_i, m_i) = X_i$ in
if $(l_i = 0) \wedge (t_a < v_i \leq t_a + \delta(l_i)) \wedge (l_i \notin \mathfrak{B}_a)$ % Checks on content of X_i.
% Case 1: Message
$Y_i := k_i, m_i$ % Will output the message directly.
elsif $(l_i < l) \wedge (t_a < v_i \leq t_a + \delta(l_i)) \wedge (l_i \notin \mathfrak{B}_a)$

```
        % Case 2: Handle
            let h_i = Fresh(H_a) in                    % Generation of a new handle.
            H_a := H_a ∪ {h_i}
            Θ_a := Θ_a ∪ {h_i → (k_i, l_i, v_i, m_i)}  % Store data under the new handle.
            Y_i := h_i                                 % Will only output the handle.
        else break
    return Y_1, ..., Y_n                               % Output of all messages and handles.
```

4.2.3 Management of Revocation Keys

We now introduce the commands for managing keys of level Max, which will only need to be used if a compromise to one of these keys is suspected. These keys, called *revocation keys*, can be used either to revoke and update lower level keys (keys with a smaller level than Max), or to revoke and update the other revocation keys. At least N_{Max} revocation keys must be given to revoke or update a revocation key.

Update Max

If we cannot trust a certain revocation key anymore, the administrator can update it using the updateMax function. It takes as inputs a cipher-text of the key to be updated and its update encrypted with n revocation keys $k_1, \ldots, k_n$ such that $n \geq N_{\mathsf{Max}}$ and takes also the corresponding handles where these keys are stored, $h_1, \ldots, h_n$. updateMax commands will be assumed to be sent on secure channels to avoid that, if an attacker breaks very old Max keys, he could immediately deduce the current active Max keys. Actually, we do not need all updateMax commands to be run under secure channels but simply that this occurs sufficiently regularly.

```
updateMax(C, h_1, ..., h_n)
    if n < N_Max
        break
    for i = 1..n
        let (k_i, Max, v_i, m_i) = Θ_a(h_i) in          % Retrieve data under h_i.
    if ∃ j ∈ [[1, n]] s.t. v_j ≤ t_a ∨ ¬Distinct(h_1, ..., h_n)   % Checks on k_1, ..., k_n.
        break
    let (updateMax, k', v'_k, m'_k) = dec(k_1, ..., dec(k_n, C)) in   % Attempt of decryption.
    if (v'_k ≤ t_a) ∨ (v'_k > t_a + δ_Max)              % Checks on validity of new key.
        break
    Θ_a(h_1) := (k', Max, v'_k, m'_k)                   % Store the new key's information.
```

Note that the updateMax command does not introduce key cycles and avoid replays.

4.2.4 Management of Working Keys

We call any key of level $l : l <$ Max a *working key*. We include separate commands for creating working keys on the device and for updating or revoking them, each of which require a N_{Max} level Max keys. These commands are in addition to the usual operational key management functions arising from the encryption and decryption commands given in section 4.2.2: they are intended to be used for bootstrapping the system or for removing and updating possibly compromised keys. In the descriptions below, $n \geq N_{\mathsf{Max}}$.

Create

The create function takes as inputs a cipher-text containing the "order" to create and the different data to create, the whole encrypted with n revocation keys $k_1, \ldots, k_n$ and takes also the corresponding handles where these keys are stored, $h_1, \ldots, h_n$.

$\mathsf{create}(C, h_1, \ldots, h_n)$
  if $n < N_{\mathsf{Max}}$
    break
  for $i = 1..n$
    let $(k_i, \mathsf{Max}, v'_i, m'_i) = \Theta_a(h_i)$ in % Retrieve data under h_i.
  if $\exists\ j \in [\![1, n]\!]$ s.t. $v'_j \leq t_a \vee \neg\mathsf{Distinct}(h_1, \ldots, h_n)$ % Checks on $k_1, \ldots, k_n$.
    break
  let $C' = \mathsf{dec}(k_1, \ldots, \mathsf{dec}(k_n, C))$ in % Attempt of decryption.
  let $(\mathsf{create}, x_1, l_1, v_1, m_1, \ldots, x_p, l_p, v_p, m_p) = C'$ in
  if $\exists\ j$ s.t. $(l_j \geq \mathsf{Max}) \vee (v_j \leq t_a) \vee (v_j > t_a + \delta(l_j)) \vee (l_j \in \mathfrak{B}_a)$
    break % Checks of new attributes.
  for $j = 1..p$
    let $h'_i = \mathsf{Fresh}(H_a)$ in % Generate a new handle h'_i.
    $H_a := H_a \cup \{h'_i\}$
    $\Theta(h'_i) := (x_i, l_i, v_i, m_i)$ % Store new information under h'_i.
  return $h'_1, \ldots, h'_p$ % Output of all handles.

Update

The update function takes as inputs a cipher-text containing the "order" to update, the different values to change and their updates, the whole encrypted with n revocation keys $k_1, \ldots, k_n$ and takes also the corresponding handles where these keys are stored, $h_1, \ldots, h_n$.

$\mathsf{update}(C, h_1, \ldots, h_n)$
  if $n < N_{\mathsf{Max}}$
    break
  for $i = 1..n$
    let $(k_i, \mathsf{Max}, v_i, m_i) = \Theta_a(h_i)$ in % Retrieve data under h_i.
  if $\exists\ j \in [\![1, n]\!]$ s.t. $v_j \leq t_a \vee \neg\mathsf{Distinct}(h_1, \ldots, h_n)$ % Checks on $k_1, \ldots, k_n$.
    break
  let $C' = \mathsf{dec}(k_1, \ldots, \mathsf{dec}(k_n, C))$ in % Attempt of decryption.
  let $(\mathsf{update}, x_1, x'_1, l'_1, v'_1, m'_1, \ldots, x_p, x'_p, \ldots, m'_p) = C'$ in
  for $j = 1..p$ % Update of handles containing x_j
    for $h \in H_a$ s.t. $\Theta(h) = (x_j, l, v, m)$ (with checks on attributes).
    if $(l < \mathsf{Max}) \wedge (l'_j = l) \wedge (t_a < v'_j \leq t_a + \delta(l_j)) \wedge (l'_j \notin \mathfrak{B}_a)$
      $\Theta(h) := (x'_j, l'_j, v'_j, m'_j)$

Delete

We define the delete command in its most general form. The administrator may wish to delete keys, for example if they are out-of-date or not useful anymore. He may wish to delete a precise set of keys but he may also wish to delete them on the basis of their attributes, e.g. validity

time. To be as flexible as possible, we consider an delete command that is parametrized by a function F such that:

$$F : \mathcal{L} \times \mathcal{T} \times \mathcal{M} \to \{\bot, \top\}$$

which defines, according to some criteria (e.g. level, validity, etc) what is going to be kept or deleted. The delete function takes as input such a function F, encrypted with n revocation keys, and the corresponding handles $h_1, \ldots, h_n$.

$\mathsf{delete}(C, h_1, \ldots, h_n)$
  if $n < N_{\mathsf{Max}}$
    break
  for $i = 1..n$
    let $(k_i, \mathsf{Max}, v_i, m_i) = \Theta_a(h_i)$ in % Retrieve data under h_i.
  if $\exists\ j \in [\![1, n]\!]$ s.t. $v_j \leq t_a \vee \neg\mathsf{Distinct}(h_1, \ldots, h_n)$ % Checks on $k_1, \ldots, k_n$.
    break
  let $(\mathsf{delete}, F) = \mathsf{dec}(k_1, \ldots, \mathsf{dec}(k_n, C))$ in % Attempt of decryption.
  for $h \in H_a$ s.t. $\Theta_a(h) = (x, i, v, m)$ % For each handle: checks if F is (resp.
    if $F(i, v, m) = \top$ is not) satisfied and delete (resp. keep)
      $\Theta_a := \Theta_a \setminus \{h \mapsto (x, i, v, m)\}$ the content of the handle.

Note that, in practice an implementation should offer some specialized delete functions for the particular application. Indeed, sending an arbitrary Boolean function as a parameter would raise both implementation and security issues. However, by defining F in this general way we ensure that our security results hold for any choice of F.

Blacklist

The Blacklist function takes as input a cipher-text containing the levels to blacklist together with expiration times, encrypted with n revocation keys $k_1, \ldots, k_n$. The effect of the command is to add the levels to the blacklist and delete all the keys of the corresponding levels, including smaller ones. Blacklisting the level instead of all the keys saves memory. It also means the administrator does not need to retain the actual values of all the working keys.

$\mathsf{blacklist}(C, h_1, \ldots, h_n)$
  if $n < N_{\mathsf{Max}}$
    break
  for $i = 1..n$
    let $(k_i, \mathsf{Max}, v_i, m_i) = \Theta_a(h_i)$ in % Retrieve data under h_i.
  if $\exists\ j \in [\![1, n]\!]$ s.t. $v_j \leq t_a \vee \neg\mathsf{Distinct}(h_1, \ldots, h_n)$ % Checks on $k_1, \ldots, k_n$.
    break
  let $(\mathsf{blacklist}, (l_1, t_1), \ldots, (l_p, t_p)) = \mathsf{dec}(k_1, \ldots, \mathsf{dec}(k_n, C))$ in % Attempt of decryption.
  for $i = 1..p$
  % First, add to Blacklist
    $\mathfrak{B}_a := \mathfrak{B}_a \cup \{(l_i, t_i)\}$
  % Then, delete affected keys
    for $h \in H_a$ s.t. $\Theta_a(h) = (x', l', v', m')$
      if $l' \leq l_i$
        $\Theta_a := \Theta_a \setminus \{h \mapsto (x', l', v', m')\}$

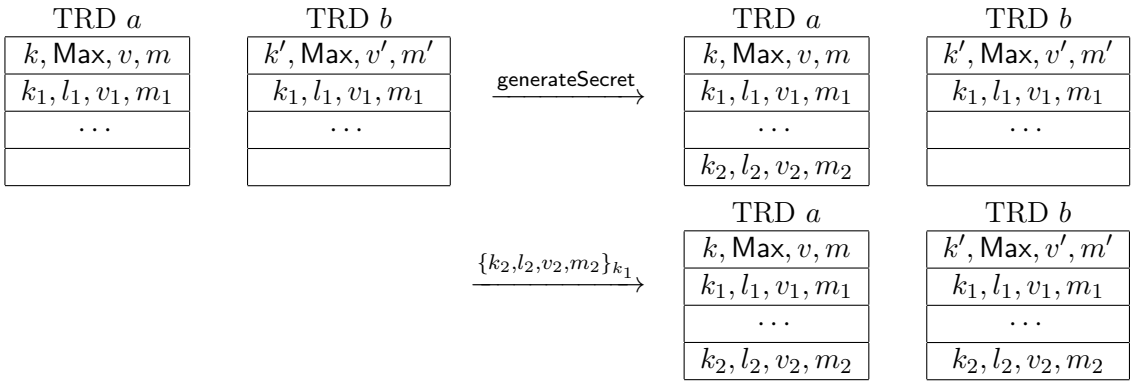


Figure 4.2: A simple execution. A key is created on device a and shared with device b.

4.2.5 Example

Let us consider two TRDs a and b (see Figure 4.2) initialized with handles containing keys of level Max k and k' and sharing key k_1 of level l_1. In a first step, the user of device a generates, using the generateSecret command, a secret key k_2 with a level l_2 (such that $l_2 < l_1$) and a validity v_2, this new secret appears in TRD a stored under a new handle. To share the key with b, a encrypts k_2 under k_1 using encrypt. When the message is received by the user of b he uses the decrypt command which will store the new key k_2 under a new handle.

4.2.6 Threat Scenario

We consider a scenario where the attacker:

- controls the network (he may read and send messages on the network),
- controls the host machines on which the TRD are connected; therefore he can execute API commands on any TRD
- may break some arbitrary keys of his choice, typically by brute-forcing some keys or employing side-channel attacks.

In the remainder of this chapter, we show that, provided the attacker does not break too many keys of level Max (an attacker should not break more than $N_{\mathsf{Max}} - 1$ keys simultaneously stored on a TRD), then our system can self-repair given time or one can explicitly fix a TRD. More precisely, we show the two following results:

- If some key of level $l \neq \mathsf{Max}$ is lost then after some time (actually, at time $v + \delta_l$ where v is the validity date of the lost key) the keys of level l are secure again.
- Moreover, if some key of level $l \neq \mathsf{Max}$ is lost and a TRD receives a command blacklisting the level l, then all the keys on the TRD are secure again.

Note that our analysis will not deal with denial of service attacks.

4.3 Formal Model

We study the security offered by our model in a symbolic model, where messages are represented by terms, following a now standard approach used for many protocols (see e.g. [ABF07, BC08, Low96]).

4.3.1 Syntax

We assume a finite set of names $\mathcal{A}$ and two infinite sets $\mathcal{N}$ and $\mathcal{K}$ respectively for nonces and keys. We also assume a finite set $\mathcal{H}$ representing handles and an infinite set $\mathcal{M}$ representing the miscellaneous fields, with $\epsilon \in \mathcal{M}$ representing the empty element. We recall that $\mathcal{L}$ denotes the set of levels. Messages are represented using a term algebra Terms defined by the following grammar:

$$T, T_1, T_2, \ldots := a \mid n \mid k \mid l \mid t \mid m \mid \{T\}_k \mid \langle T_1, T_2 \rangle$$

where $a \in \mathcal{A}$, $n \in \mathcal{N}$, $k \in \mathcal{K}$, $l \in \mathcal{L}$, $t \in \mathcal{T}$, $m \in \mathcal{M}$. The term $\{t\}_k$ represents the message t (symmetrically) encrypted by the key k while the term $\langle t_1, t_2 \rangle$ represents the concatenation (or more precisely, the pairing) of the two messages t_1 and t_2. For simplicity, we will often write $(t_1, \ldots, t_n)$ instead of $\langle t_1, \langle t_2, \ldots \langle t_{n-1}, t_n \rangle \ldots \rangle$. The notion of *subterm* is defined as usual: t' is a subterm of t if t' occurs at some position p in t, that is $t|_p = t'$. We denote by $\mathrm{St}(m)$ the set of subterms of m and by extension $\mathrm{St}(S)$ is the set of subterms of terms in the set S.

A *global state* of our system is described by a tuple $(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ where $\mathcal{P} \subseteq \mathcal{A}$ is the set of TRDs in the system, $t \in \mathcal{T}$ represents the current time, $\mathfrak{M} \subseteq \mathsf{Terms}$ represents the set of messages sent so far over the network, N and K are respectively sets of currently used nonces and keys[3] by all the APIs and $\mathcal{I}$ is a function:

$$\mathcal{I} : a \mapsto (\Theta_a, H_a, \mathfrak{B}_a, t_a, N_a, K_a)$$

that represents the current local state of the TRD $a \in \mathcal{P}$. More precisely, $H_a \subseteq \mathcal{H}$ represents the finite set of handles currently used in the API a and $\mathfrak{B}_a \subseteq \mathcal{L} \times \mathcal{T}$ represents the set of blacklisted levels for which the TRD does not accept keys anymore. $N_a \subseteq N$ and $K_a \subseteq K$ are respectively the nonces and keys that have been generated or stored on the API. Θ_a represents the key table of the TRD. Formally, it is a function of the form:

$$\Theta_a : H_a \to (\mathcal{K} \cup \mathcal{N}) \times \mathcal{L} \times \mathcal{T} \times \mathcal{M}.$$

Indeed, as seen in Section 4.2.1, each handle $h \in H_a$ points to an entry (x, l, v, m) corresponding a nonce or a key x and its attributes: the level l, its validity v, and other miscellaneous information m.

As indicated in the definition of a global state, all keys come with a level.

Definition 4.1. *Let $(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ be a global state and let k be a key in K. The level of k is defined as follows :*

$$\mathsf{Level}(k) = \{l \mid \exists\ a \in \mathcal{P}, h \in H_a \ s.t.\ \Theta_a(h) = (k, l, v, m)\}$$

It is *a priori* possible for a key k to have several levels (i.e. $\mathsf{Level}(k)$ is not a singleton) but we will show in our proofs that this never happens for uncompromised keys. We will therefore say that a key k is of level l if $l \in \mathsf{Level}(k)$.

[3] N and K are here as artifacts of the model. Since an API gets a random key (or nonce) when it creates it, there is very little chance that it generates a key (or nonce) which has already been used. To capture this, we model a global knowledge of all nonces and keys used by all APIs to be sure that freshly generated keys (or nonces) are new.

$$\frac{t \in \mathfrak{M}}{\mathfrak{M} \vdash t} \qquad \frac{\mathfrak{M} \vdash x, \mathfrak{M} \vdash y}{\mathfrak{M} \vdash \langle x, y \rangle} \qquad \frac{\mathfrak{M} \vdash \langle x, y \rangle}{\mathfrak{M} \vdash x}$$

$$\frac{\mathfrak{M} \vdash \langle x, y \rangle}{\mathfrak{M} \vdash y} \qquad \frac{\mathfrak{M} \vdash x, \mathfrak{M} \vdash y}{\mathfrak{M} \vdash \{x\}_y} \qquad \frac{\mathfrak{M} \vdash \{x\}_t, \mathfrak{M} \vdash y}{\mathfrak{M} \vdash x}$$

Figure 4.3: Deduction Rules (Intruder)

4.3.2 Semantics

The behavior of our API is modeled by the transitions of our formal system. We define four kinds of transitions: transition of time, forgery by the attacker, silent transitions, and key management commands.

Transition of time We assume that the execution of API commands is instantaneous (this could be adapted to take into account worst case execution time). The effect of the time is modelled by a separate and explicit TIM transition:

$$\text{(TIM)} \qquad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \xrightarrow{\text{(Time passes)}} (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t') \text{ with } t' > t.$$

Forgery by the attacker As is usually the case in formal models, the ability of the attacker to construct, or *deduce*, is modeled by a relation $\vdash$. We write $S \vdash m$ if m can be deduced from S. The deduction relation is formally defined in Figure 4.3 Intuitively, the attacker can deduce any term that can be obtained by pairing, encrypting, projecting, and decrypting whenever it has the encryption key.

Example 4. *Let* $S = \{\{k_1\}_{k_2}, \langle k_2, k_3 \rangle, \{k_4\}_{k_5}\}$. *Then* $S \vdash k_1$, $S \vdash k_2$, $S \vdash k_3$, *but* $S \nvdash k_4$, $S \nvdash k_5$. *We also have* $S \vdash \langle k_2, k_1 \rangle$, *but* $S \nvdash \langle k_2, k_4 \rangle$.

The attacker may send any deducible message over the network. This is reflected by the following transition:

$$\text{(DED)} \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \xrightarrow{\text{(Deduction)}} (\mathcal{P}, \mathcal{I}, \mathfrak{M} \cup \{m\}, N, K, t) \text{ provided } \mathfrak{M} \vdash m.$$

Silent transitions Each API command executed on a TRD a with input m defines a transition $\Theta_a, H_a, \mathfrak{B}_a \to \Theta'_a, H'_a, \mathfrak{B}'_a$. Moreover, the TRD outputs some message m'. Given a global state $(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ such that $m \in \mathfrak{M}$, the corresponding global transition is

$$(\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \to (\mathcal{P}, \mathcal{I}', \mathfrak{M} \cup \{m'\}, N \cup N', K \cup K', t)$$

with N' and K' being respectively the sets of nonces and keys generated by a during the execution of the command. The function $\mathcal{I}'$ is defined by $\mathcal{I}' : a \mapsto (\Theta'_a, H'_a, \mathfrak{B}'_a, t_a)$ for the corresponding API a and $\mathcal{I}'(a') = \mathcal{I}(a')$ otherwise.

Key management commands As explained in Section 4.1.2, at least some revocation commands need to be kept hidden from the attacker. Otherwise, assuming that the attacker controls and memorizes all the traffic on the network and assuming he can break keys (which corresponds

$$\text{(UPM)} \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \xrightarrow{\mathsf{updateMax}} (\mathcal{P}, \mathcal{I}', \mathfrak{M}, N', K', t)$$

$$\text{(NEW)} \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \xrightarrow{\mathsf{create}} (\mathcal{P}, \mathcal{I}', \mathfrak{M} \cup \{m\}, N', K', t)$$
$$\text{with } m = \{\mathsf{create}, N, N', x_1, l_1, v_1, m_1 \ldots, m_p\}_{k_1 \cdots k_n}$$

$$\text{(UPD)} \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \xrightarrow{\mathsf{update}} (\mathcal{P}, \mathcal{I}', \mathfrak{M} \cup \{m\}, N', K', t)$$
$$\text{with } m' = \{\mathsf{update}, x_1, x_1', l_1', v_1', m_1' \ldots, m_p'\}_{k_1 \cdots k_n}$$

$$\text{(DEL)} \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \xrightarrow{\mathsf{delete}} (\mathcal{P}, \mathcal{I}', \mathfrak{M} \cup \{m\}, N, K, t)$$
$$\text{with } m = \{\mathsf{delete}, F\}_{k_1 \cdots k_n}$$

$$\text{(BLK)} \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \xrightarrow{\mathsf{blacklist}} (\mathcal{P}, \mathcal{I}', \mathfrak{M} \cup \{m\}, N, K, t)$$
$$\text{with } m = \{\mathsf{blacklist}, (l_1, t_1), \ldots, (l_n, t_n)\}_{k_1 \cdots k_n}$$

For each transition, the function $\mathcal{I}'$ and the sets N' and K' are updated as specified by the corresponding command, as for silent transitions.

Figure 4.4: Key management command transitions

to our threat scenario), then breaking N_{Max} old keys may compromise the entire system, even if the compromised keys had not been in use for some time.

We therefore assume that the key management commands for keys of level Max are sent over a private channel. In practice, this could be achieved by several means. For example, we may assume that the key administrator has a physical access to the TRD that needs to be updated. Or we may also assume that the user would connect his/her TRD to a trusted machine, on which a secure channel (e.g. via TLS) is established with the key administrator. Note that the key management commands for keys of level Max are executed only when a key of level Max is lost (or suspected to be lost), or when keys of level Max are updated (e.g. when their validity has expired). We expect these events to occur infrequently.

Note that our assumption does not prevent an adversary from trying to run the UPM command. In particular, in case it has sufficiently many keys of level Max, it may well build a well formed Update Max command and send it to a TRD. This is reflected in the silent transitions defined above.

All the other key management commands can be sent over an insecure network. The corresponding transition system is presented in Figure 4.4. The fact that we assume the UPM command is sent over a secure channel is reflected in the fact that the set $\mathfrak{M}$ remains unchanged for this transition.

In the remainder of the chapter we assume that managers in charge of generating key management commands behave consistently. More precisely, we assume that they only use fresh keys when updating and creating keys and that they never give different attributes to the same key value.

Key compromise We model the fact that the attacker may compromise a key by adding a transition lost, which allows an attacker to obtain a key of his choice.

$$\text{(LST)} \quad (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t) \xrightarrow{\mathsf{Lost}(k)} (\mathcal{P}, \mathcal{I}, \mathfrak{M} \cup \{k\}, N, K, t)$$

where k is a key that appears on at least one TRD, that is k occurs in the image of $\mathcal{I}$.

4.3.3 Example

Let us reconsider the example of Figure 4.2. We consider E as the initial global state and we write $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$. To simplify, let us have that $\mathcal{P} = \{a, b\}$ and $K = \{k, k', k_1\}$. Both t and $\mathfrak{M}$ are arbitrary. Then $\mathcal{I} = \{a \mapsto (\Theta_a, H_a, \mathfrak{B}_a, t_a, N_a, K_a),\ b \mapsto (\Theta_b, H_b, \mathfrak{B}_b, t_b, N_b, K_b)\}$ where $t_a = t_b = t$, $H_a = \{h_1, h_2\}$, $H_b = \{h'_1, h'_2\}$, $K_a = K_b = K$. We have $\Theta_a(h_1) = (k, \mathsf{Max}, v, m)$, $\Theta_b(h'_1) = (k', \mathsf{Max}, v', m')$ and $\Theta_a(h_2) = \Theta_b(h'_2) = (k_1, l_1, v_1, m_1)$.

Using the generationSecret command with TRD a will lead to a new global state E' such that $E' = (\mathcal{P}, \mathcal{I}', \mathfrak{M}, N, K', t)$ where $\mathcal{I}' = \{a \mapsto (\Theta'_a, H'_a, \mathfrak{B}_a, t_a, N_a, K'_a), b \mapsto (\Theta_b, H_b, \mathfrak{B}_b, t_b, N_b, K_b)\}$ with $K'_a = K' = K \cup \{k_2\}$, $H'_a = H_a \cup \{h_3\}$ and $\Theta'_a = \Theta_a \cup \{h_3 \mapsto (k_2, l_2, v_2, m_2)\}$.

Then, the encrypt command leads to a new global state E'' such that $E'' = (\mathcal{P}, \mathcal{I}, \mathfrak{M}', N, K, t)$ where $\mathfrak{M}' = \mathfrak{M} \cup \{\{k_2, l_2, v_2\}_{k_1}\}$ because the message was exchanged over the network. Finally the decrypt command is used by TRD b on the message $\{k_2, l_2, v_2\}_{k_1}$. This operation leads to E''', such that $E''' = (\mathcal{P}, \mathcal{I}''', \mathfrak{M}, N, K, t)$ with $\mathcal{I}''' = \{a \mapsto (\Theta'_a, H'_a, \mathfrak{B}_a, t_a, N_a, K'_a), b \mapsto (\Theta'''_b, H'''_b, \mathfrak{B}_b, t_b,\ N_b, K'''_b)\}$ with $K'''_b = K_b \cup \{k_2\}$, $H'''_b = H_b \cup \{h'_3\}$ and $\Theta'''_b = \Theta_b \cup \{h'_3 \mapsto (k_2, l_2, v_2, m_2)\}$.

4.4 Security Properties of the Design

We are now ready to formally state the security properties discussed in Section 4.2.6. Intuitively, we would like to show that apart from explicitly lost keys, any key that has not expired is secure, that is, cannot be deduced by the attacker. However, this is not strictly the case: several commands such as encrypt or the administrator commands produce messages of the form $\{\ldots k' \ldots\}_k$ that are sent over the network and possibly stored by the attacker. It could be the case that k expires before k'. Clearly, if k becomes known to the attacker then k' will be immediately known as well, even if k has expired and k' has not.

Therefore, we distinguish three kinds of key: *valid* keys, *latent* keys, and *dead* keys. Informally, valid keys are those stored on a device whose validity time has not passed, latent keys are those which are not valid but on which security of some encryption of a valid lower level key might still depend, and dead keys are everything else.

Definition 4.2. *Let $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ be a global state and $k \in K$. Given $a \in \mathcal{P}$, we denote by Θ_a the key table stored on a and defined by $\mathcal{I}(a) = (\Theta_a, \ldots)$.*

- *k is a **valid** key if $\exists\ a \in \mathcal{P}$, $\exists\ h \in H_a$ s.t. $\Theta_a(h) = (k, l, v, m)$ with $v > t_a$ or if there exists $\{\ldots, k, l, v, m, \ldots\} \in St(\mathfrak{M})$ with $v > t$.*
  We denote by $\mathcal{V}_E$ the set of valid keys in E.
- *k is a **latent** key if $\exists\ a \in \mathcal{P}$, $\exists\ h \in H_a$ s.t. $\Theta_a(h) = (k, l, v, m)$ with $v + \Delta(l) > t$ or if there exists $\{\ldots, k, l, v, m, \ldots\} \in St(\mathfrak{M})$ with $v + \Delta(l) > t$.*

We denote by $\mathscr{U}_E$ the set of latent keys in E. (The notation $\mathscr{U}_E$ comes from the fact that these keys may also be viewed as "undead".)

- *Otherwise, k is a **dead** key.*

A key is latent if it expired but less than $\Delta(l)$ ago. The reason why latent keys still influence the security of the whole system can be illustrated with the following scenario. Let k be a key of level l and validity time v and consider $l_1 < \cdots < l_n < l$ such that $\Delta(l) = \sum_{i=1}^{n} \delta(l_i)$. Then it may be the case that k was used to encrypt a key k_n of level l_n just before its expiration time, yielding the message $\{k_n, l_n, v_n, m_n\}_k$ with $v_n = v + \delta(l_n)$. Similarly, each key k_i may have been used to encrypt a key k_{i-1} of level l_{i-1}, yielding a message $\{k_{i-1}, l_{i-1}, v_{i-1}, m_{i-1}\}_{k_i}$ with $v_{i-1} = v_i + \delta(l_{i-1})$. Then it is easy to see that $v_1 = v + \Delta(l)$ and clearly if k is lost at anytime after its expiration date v but before $v + \Delta(l)$ then the security of k_1 (valid) would be immediately compromised as well.

4.4.1 Initial State

As discussed in section 4.2.2, before being deployed, a TRD is loaded with some level Max keys so that the administrator can then remotely equip the device with an initial set of working keys.

Definition 4.3. *A global state $E_\emptyset$ is said to be* originating *if $E_\emptyset = (\mathcal{P}, \mathcal{I}_0, \emptyset, 0, \emptyset, \emptyset)$ with $\mathcal{I}_0 : a \mapsto (\Theta_a, H_a, 0, \emptyset, \emptyset)$ and for all $a \in \mathcal{P}$, we have $H_a = \{h_1^a, \ldots, h_n^a\}$ with $\Theta_a(h_i^a) = (k_i^a, \mathsf{Max}, v_i^a, m_i^a)$. The k_i^a keys are revocation keys initially placed in the memory of each API and should be all distinct: $k_i^a = k_j^a$ implies $i = j$.*

4.4.2 Keys of Level Max

The keys of level Max are crucial since they can be used to update and revoke all the keys, including keys of level Max themselves. We first show that security of these keys is preserved, assuming that not too many of the keys of level Max stored on an API are simultaneously lost.

Hypothesis 1. *Let $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ be a global state such that $E_\emptyset \to^* E$ and let $\mathscr{L}$ the set of keys that have been lost so far (i.e. the set of keys k such that the transition $\mathsf{Lost}(k)$ has been executed). We assume that $\forall a \in \mathcal{P}$, we have*

$$\#\{k \mid k \in \mathscr{U}_E \cap \mathscr{L} \cap K_a \text{ and } \mathsf{Max} \in \mathsf{Level}(k)\} \leq N_{\mathsf{Max}} - 1$$

where K_a is the set of keys associated to a by the function $\mathcal{I}$ (that is $\mathcal{I}(a) = (\ldots, K_a)$).

We assume this hypothesis to hold in all the global states we consider in the remainder of the paper. From a practical point of view, it means that the lifetime of a key of level Max should be set such that during the time when a Max key is latent, it is sufficiently unlikely that an attacker will break more than $N_{\mathsf{Max}} - 1$ such latent keys. More precisely, our assumption is slightly weaker since the attacker may break more keys provided that no more than $N_{\mathsf{Max}} - 1$ of them have occurred on the same TRD.

The design of our API ensures that no Max keys can be learned by an attacker, except for the explicitly lost ones.

Theorem 4.1. *Let E be a global state such that $E_\emptyset \longrightarrow^* E$, then:*

$$\forall k \in \mathscr{V}_E^{\mathsf{Max}} \setminus \mathscr{L},\ E \not\vdash k.$$

The proof relies on the fact that keys of level Max only appear in key position in the messages sent over the network and the fact that for any Max key, $\mathsf{Level}(k)$ is a singleton. These two properties are shown to be preserved by application of the rules and rely on Hypothesis 1 that ensures that the attacker never knows sufficiently many Max keys to forge an administrator command.

Before proving Theorem 4.1, we first prove an easy and rather straightforward lemma that will be often used in the proofs: if a key appears only in key position in a set of messages, then it is not deducible.

Lemma 4.1. *We suppose that keys are atomic. Let $\mathfrak{M}$ be a set of terms and k a key such that k only appears in key position in m, $\forall m \in \mathfrak{M}$. Then, $\forall t$ such that $\mathfrak{M} \vdash t$, k only appears in key position in t, if k appears in it. In particular, $\mathfrak{M} \nvdash k$.*

Proof. Let k be a key and the set $\mathfrak{M}$ be such as defined. Let us prove the lemma by induction on the number of steps needed to deduce t.

Base case: All terms in $\mathfrak{M}$ satisfy the property by hypothesis.

Induction Hypothesis: $\forall t$ such that $\mathfrak{M} \vdash t$ in n or less steps, then t satisfies the property.

Now suppose that $\mathfrak{M} \vdash t$ in $n+1$ steps:

- If the last step is a decomposition rule (all rules but encryption and pairing). We that t is a subterm of a term t' with $t' = \{t\}_v$ or $\langle t, v\rangle$. In the first one, $t' = \{t\}_v$, if k appears not in a key position in t, it appears not in a key position in t' too, and we have a contradiction with the induction hypothesis. The conclusion is the same in the case of the pairing. Thus t, verifies the property.

- If the last step is the encryption rule, then t is of the form $\{t'\}_y$ with $\mathfrak{M} \vdash t'$ in n steps. If $y = k$, then $\exists\, \Pi$ a proof, in n or less steps, of $\mathfrak{M} \vdash k$, which is a contradiction, since k is not in key position in the term k. Then, we must have $y \neq k$ and, in that case, the induction hypothesis gives us the fact that k only appears in key position in t' and, due to the form of t, it is clear that t also verifies this property.

- If the last step is the pairing rule, then t is of the form $\langle t', t''\rangle$ with $\mathfrak{M} \vdash t', t''$ in, at most, n steps. Using the induction hypothesis on t' and t'' it is clear that t also satisfy the property.

Moreover, suppose that $\mathfrak{M} \vdash k$. This contradicts the fact that k only appears in key position. Thus $\mathfrak{M} \nvdash k$.

□

Now let us consider the following hypothesis:

Hypothesis 2. *We suppose that the administrator does not make any mistakes when performing administration commands, especially, keys are distributed consistently, with the same attributes and the administrator should not create equalities between keys. It means that if $\exists\, m_1, m_2 \in \mathfrak{M}$ such that $m_1 = \{\mathsf{cmd}_1, \ldots, (k_1), k'_1, \mathsf{att}_1, \ldots\}_{q_1 \cdots q_n}$ and $m_2 = \{\mathsf{cmd}_2, \ldots, (k_2), k'_2, \mathsf{att}_2, \ldots\}_{q'_1 \cdots q'_n}$, where att_i are the attributes associated to k'_i in the command $\mathsf{cmd}_i \in \{\mathsf{blacklist}, \mathsf{create}, \mathsf{update}\}$ (and k_1, k_2 may not appear), then we have that $k'_1 = k'_2$ (resp. $k_1 = k_2$) implies $\mathsf{cmd}_1 = \mathsf{cmd}_2$, $\mathsf{att}_1 = \mathsf{att}_2$ and $k_1 = k_2$ (resp. $k'_1 = k'_2$).*

It also means that if there are two updateMax *commands m_1 and m_2 such that $m_1 = \{\mathsf{updateMax}, k, v, m\}_{k_1 \cdots k_n}$ and $m_2 = \{\mathsf{updateMax}, k', v', m'\}_{k'_1 \cdots k'_n}$, then if $k = k'$ (or $k_1 = k'_1$) then we have $m_1 = m_2$.*

We also assume that updateMax, update *and* create *commands are made with fresh keys. Moreover, the administrator performs operations according to the specification.*

Theorem 4.1 is proved by induction, showing that revocation keys satisfy a stronger invariant, which is reflected in the following proposition.

Proposition 4.1. *Let us consider global states E and E' such that $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ and $E' = (\mathcal{P}', \mathcal{I}', \mathfrak{M}', N', K', t')$ with $E_\emptyset \longrightarrow^* E \xrightarrow{L} E'$. Let $\mathscr{L}$ be a set of keys. Let us consider the following properties:*

(1) *$\forall a \in \mathcal{P}$, $\forall h \in H_a$ such that $\Theta_a(h) = (k_1, \mathsf{Max}, v_1, m_1)$ and $\forall b \in \mathcal{P}$, $\forall h' \in H_b$ such that $\Theta_b(h') = (k_2, \mathsf{Max}, v_2, m_2)$, then, we have:*

$$k_1 = k_2 \text{ and } k_1, k_2 \notin \mathscr{L} \Longrightarrow (v_1, m_1) = (v_2, m_2).$$

Moreover, $\forall a \in \mathcal{P}$, all revocation keys stored in a given API are different, i.e. $\forall h, h' \in H_a$ s.t. $\Theta_a(h) = (k, \mathsf{Max}, v, m)$ and $\Theta_a(h') = (k', \mathsf{Max}, v', m')$, then $k = k' \Rightarrow h = h'$.

We consider the set $\mathscr{U}_E^{\mathsf{Max}}$ of latent revocation keys of global state E.

(2) *$\forall k \in \mathscr{U}_E^{\mathsf{Max}} \setminus \mathscr{L}$, $\forall m \in \mathfrak{M}$ we have, for all occurrences of k in m is of the form $\{m'\}_k$.*

Then, if E satisfies (1) and (2) for $\mathscr{L}$, then E' also satisfies (1) and (2) for $\mathscr{L}'$ where $\mathscr{L}' = \mathscr{L} \cup \{k\}$ if $E \xrightarrow{\mathsf{Lost}(k)} E'$ or $\mathscr{L}' = \mathscr{L}$ otherwise.

Proof. To prove the proposition, we will show that properties (1) and (2) are invariant under application of transition rules:

- $E \xrightarrow{(\text{DED})} E'$. This transition does not affect handles at all, then it is obvious that (1) is invariant. We also have $\mathfrak{M}' = \mathfrak{M} \cup \{u\}$ with $\mathfrak{M} \vdash u$. Let $k \in \mathscr{U}_E^{\mathsf{Max}} \setminus \mathscr{L}$ such that $\mathsf{Level}(k) = \mathsf{Max}$, and $m \in \mathfrak{M}'$ such that $k \in \mathrm{St}(m)$. If $m \in \mathfrak{M}$, then we conclude using the induction hypothesis that E verifies (2). We now consider the case where $m = u$. Using Lemma 4.1 and induction hypothesis, we know that u is a term where k can only appear in key position and we conclude that (2) holds.

Considering that and using Lemma 4.1, we easily obtain following lemma:

Lemma 4.2. *Let E be a global state satisfying to properties (1) and (2) for $\mathscr{L}$, then:*

$$\forall k \in \mathscr{U}_E^{\mathsf{Max}} \setminus \mathscr{L},\ \mathfrak{M} \not\vdash k.$$

Now let us continue our proof of Proposition 4.1.

- $E \xrightarrow{(\text{TIM})} E'$. In this case, we have $\mathcal{I}' = \mathcal{I}$ since there is no modification of the memory of all TRD's, thus, (1) holds obviously. Moreover, as we also have $\mathfrak{M}' = \mathfrak{M}$, using the fact that there are no keys which are lost during the transition and since latent revocation keys may only become dead with time, i.e. $\mathscr{U}_{E'}^{\mathsf{Max}} \setminus \mathscr{L}' \subseteq \mathscr{U}_E^{\mathsf{Max}} \setminus \mathscr{L}$, (2) clearly holds for E'.

- $E \xrightarrow{\text{(LST)}} E'$. As in the previous case, we have $\mathcal{I}' = \mathcal{I}$ (no modification of handles on TRD's), thus, (1) holds. In addition, we have that $\mathfrak{M}' = \mathfrak{M} \cup \{k\}$, $\mathscr{L}' = \mathscr{L} \cup \{k\}$, with k the lost key, and $\mathscr{U}_{E'}^{\mathsf{Max}} = \mathscr{U}_{E}^{\mathsf{Max}}$ since $t' = t$. Let $p \in \mathscr{U}_{E'}^{\mathsf{Max}} \setminus \mathscr{L}'$ and let $m \in \mathfrak{M}'$ such that $p \in \mathrm{St}(m)$. Then, since $p \neq k$ and keys are atomic, we must have $p \notin \mathrm{St}(k)$, thus $m \in \mathfrak{M}$. Using the fact that $p \in \mathscr{U}_{E'}^{\mathsf{Max}} \setminus \mathscr{L}' \subseteq \mathscr{U}_{E}^{\mathsf{Max}} \setminus \mathscr{L}$ and using the fact that E satisfies (2), we can conclude easily that all occurrence of p in m are of the form $\{m'\}_k$ and conclude that the property (2) is invariant.

- $E \xrightarrow{\text{(UPM)}} E'$.

  (1) Let a be the TRD modified by the transition. Consider $a_1, a_2 \in \mathcal{P}$ and $h_1 \in H_{a_1}$, $h_2 \in H_{a_2}$ s.t. $h_1 \neq h_2$, $\Theta_{a_1}(h_1) = (k_1, \mathsf{Max}, v_1, m_1)$ and $\Theta_{a_2}(h_2) = (k_2, \mathsf{Max}, v_2, m_2)$ with $k_1, k_2 \notin \mathscr{L}'$ and $k_1 = k_2$. We note that $\mathscr{L}' = \mathscr{L}$. We will show that $(v_1, m_1) = (v_2, m_2)$:
  - If neither h_1 nor h_2 has been updated by the transition. Then, we conclude using the induction hypothesis.
  - The case where h_1 and h_2 where both updated by the transition is impossible according to the specification of the API.
  - Now assume that h_1 has been updated (and thus $a_1 = a$) and h_2 has not. We must have that k_1 appears in the command of the transition of the form $\{\mathsf{updateMax}, k_1, v_1, m_1\}_{q_1 \cdots q_n}$, and we have two subcases:
    * Either k_1 has never occurred in the system before, then $k_1 = k_2$ is impossible.
    * Or k_1 was already existing in the system, that is, there is a TRD c and $h \in H_c$ s.t. $\Theta_c(h) = (k_1, \mathsf{Max}, v, m)$. Using induction hypothesis over h and h_2, we have that $(v, m) = (v_2, m_2)$. Then, considering that, by case inspection, it is only possible to put a revocation key in a TRD using an $\mathsf{updateMax}$ command, thus, either k_1 was updated in c using such command or was in c since $E_\emptyset$. Since it is impossible, according to Hypothesis 1 and Lemma 4.2, for an intruder to forge an $\mathsf{updateMax}$ command, this must be an action of the administrator. Then, according to Hypothesis 2, we must have $(v_1, m_1) = (v, m)$ otherwise it implies that the administrator does not act properly. Since $(v, m) = (v_2, m_2)$, we conclude.

  Now, let us show that $\forall a \in \mathcal{P}$, $\forall h, h' \in H_a$ s.t. $\Theta_a(h) = (k, \mathsf{Max}, v, m)$ and $\Theta_a(h') = (k', \mathsf{Max}, v', m')$, then $k = k' \Rightarrow h = h'$. Let us consider $b \in \mathcal{P}$ and $h, h' \in H_b$ s.t. $\Theta_b(h) = (k, \mathsf{Max}, v, m)$ and $\Theta_b(h') = (k', \mathsf{Max}, v', m')$ with $k = k'$. If $b \neq a$, the modified TRD, then, using induction hypothesis, we have that $h = h'$. If $b = a$ with k a revocation key not added in the TRD during the transition, then, again, the induction hypothesis gives us that $h = h'$. If k is added in the TRD during the transition, let us suppose that we have $h \neq h'$. Then, we must have that $\exists\, m_1, m_2$ two $\mathsf{updateMax}$ commands coming from the administrator (we showed previously that an intruder cannot forge such commands) s.t. $m_1 = \{\mathsf{updateMax}, k, v, m\}_{q_1 \cdots q_n}$ and $m_2 = \{\mathsf{updateMax}, k, v, m\}_{q'_1 \cdots q'_n}$ with $q_1 \neq q'_1$ according to the specification since q_1 and q'_1 determine the handle where k will be stored. Such a case is in contradiction with Hypothesis 2. Thus $h' = h$. Finally, we conclude that (1) holds.

  (2) In this transition we have $\mathfrak{M}' = \mathfrak{M}$ and even if the set of latent but not lost revocation key is changed by L, there is no trace of these new keys in $\mathfrak{M}'$, thus we can conclude.

- $E \xrightarrow{\mathrm{L}} E'$ with $L \in \{(\mathrm{BLK}),(\mathrm{NEW}),(\mathrm{UPD}),(\mathrm{RVK})\}$. Since these transitions do not affect handles containing revocation keys, it is clear that (1) still holds. Concerning (2), we have $\mathfrak{M}' = \mathfrak{M} \cup \{\{m'\}_{k_1 \cdots k_n}\}$, m' may only containing keys with a level strictly lesser than Max, and $k_1, \ldots, k_n$ are revocation keys. Let $k \in \mathscr{U}_{E'}^{\mathsf{Max}} \setminus \mathscr{L}'$, i.e. $k \in \mathscr{U}_{E}^{\mathsf{Max}} \setminus \mathscr{L}$, with $\mathsf{Level}(k) = \mathsf{Max}$ and $m \in \mathfrak{M}'$ such that $k \in \mathrm{St}(m)$. If $m \in \mathfrak{M}$, we use the fact that E verifies (2) and conclude, if $m = \{m'\}_{k_1 \cdots k_n}$, then, according to the structure of m' it must be the case where $k = k_i$ for $i \in \{1, \ldots, n\}$ and it also satisfies (2).

- If $E \longrightarrow E'$. Let us consider three subcases:

  - The transition corresponds to the generationPublic, generationSecret or decrypt command. Then, handles containing revocation keys are not modified and (1) is invariant. Moreover, there is no output containing revocation keys, so (2) is also invariant.
  - The transition corresponds to the encrypt command. Such a transition does not affect handles containing revocation keys and (1) is trivially verified. Moreover, we have $\mathfrak{M}' = \mathfrak{M} \cup \{m\}$ with $m = \{U_1, \ldots, U_N\}_k$ and U_i may only contain a key of a level strictly less than Max. Moreover k can not be a revocation key according to the specification and then, (2) holds.
  - The transition corresponds to the blacklist, create, update, revoke or updateMax command. Such a transition would imply that the attacker is capable to perform an admin command, i.e. is able to forge a message with N_{Max} revocation keys for a given TRD which is in contradiction with Hypothesis 1 and Lemma 4.2.

□

We can now prove the Theorem 4.1.

Theorem 4.1. *Let E be a global state such that $E_\emptyset \longrightarrow^* E$, then:*

$$\forall k \in \mathscr{V}_{E}^{\mathsf{Max}} \setminus \mathscr{L},\ E \not\vdash k.$$

Proof. According to the Definition 4.3, we have that $E_\emptyset$ clearly satisfies properties (1) and (2) of Proposition 4.1. Using the result of that proposition, we know that E verifies these properties too. Let us consider $k \in \mathscr{V}_{E}^{\mathsf{Max}} \setminus \mathscr{L}$, then we have that $k \in \mathscr{U}_{E}^{\mathsf{Max}} \setminus \mathscr{L}$. The Lemma 4.2 allows us to conclude that $\mathfrak{M} \not\vdash k$, that is $E \not\vdash k$.

□

4.4.3 Lower Level Keys

The case of keys of lower level is more difficult than the case of Max keys. First, as soon as a key k of level l is lost, then any key of lower level $l' < l$ is compromised as well. Indeed, for any key k' of level $l' < l$ stored on the same TRD as the key k, the attacker may use the encrypt command to get k' encrypted by k and therefore deduce k'. This attack does not only compromise the keys stored on TRDs that contain the compromised key k. Indeed, assume that the key k is stored on some TRD a_1 and does not appear in some other TRD a_2. Then as soon as a_1 and a_2 share some key k'' of level $l'' > l$, the attacker may use the encrypt command on a_1 to get k encrypted by k''. Sending this encryption to a_2 and executing the decrypt command on a_2, the attacker registers k on a_2 and can now gain any key of level lower than l.

Therefore, as soon as a key of level l is lost, any key of level $l' < l$ should be considered as lost too. We introduce the notion of keys *compatible* with a set of levels, which are latent keys that have an uncompromised level.

Definition 4.4. *Let $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ be a global state, $\mathscr{L}$ a set of keys (typically lost keys), and $k \in K$. Let $\mathtt{L_v}$ be a set of levels (that typically corresponds to the levels of keys lost so far). Then k is* compatible with $\mathtt{L_v}$ *if $k \in \mathscr{U}_E \setminus \mathscr{L}$, $\mathsf{Max} \notin \mathsf{Level}(k)$, and there exists $l \in \mathsf{Level}(k)$ such that $l \not< \mathtt{L_v}$ (that is $l \not< l'$ for any $l' \in \mathtt{L_v}$). We note $\mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$ the set of keys compatible with $\mathtt{L_v}$ in the global state E. We may write $\mathcal{G}^{\mathtt{L_v}}_{E}$ instead of $\mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$ when $\mathscr{L}$ is clear from the context.*

The first property of our API is that keys of levels not marked as lost are uncompromised. We also show that for such keys, the level and all other attributes are unique (i.e. consistent across all TRDs). We actually need to enforce a stronger property, called *robustness*.

Definition 4.5. *A global state $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ is* robust up to *a set of levels $\mathtt{L_v}$ and a set $\mathscr{L}$ of (lost) keys if the following properties are satisfied:*

(1) *We consider that (l, v, m) are* attributes *of k if any of these three cases holds:*

   (a) $\exists\ a \in \mathcal{P}$, $h \in H_a$ *s.t.* $\Theta_a(h) = (k, l, v, m)$,

   (b) $\exists\ q \in K$, $m' \in \mathfrak{M}$ *s.t.* $\mathsf{Max} \in \mathsf{Level}(q)$ *and* $\{\mathsf{cmd}, \dots, k, l, v, m, \dots\}_{q_1 \cdots q_j q} \in St(m')$ *with* $\mathsf{cmd} \neq \mathsf{blacklist}$ *and* $\mathsf{Max} \in \mathsf{Level}(q_i)$,

   (c) $\exists\ q \in K$, $m' \in \mathfrak{M}$ *s.t.* $\{\dots, k, l, v, m, \dots\}_q \in St(m')$.

   If (l_1, v_1, m_1) and (l_2, v_2, m_2) are attributes of a key $k \in \mathcal{G}^{\mathtt{L_v}}_E$, then $(l_1, v_1, m_1) = (l_2, v_2, m_2)$.

(2) $\forall k \in \mathcal{G}^{\mathtt{L_v}}_E$, $\forall m \in \mathfrak{M}$ *s.t.* $k \in St(m)$, *then any occurrences of k is of one of these three forms:*

   (i) $\{m'\}_k$,

   (ii) $\{\dots, k, \dots\}_{k'}$ *with* $k' \in K$, $v_k \leq v_{k'} + \delta_k$ *and* $\mathsf{Level}(k) < \mathsf{Level}(k') \neq \mathsf{Max}$,

   (iii) $\{\dots, k, \dots\}_{q_1 \cdots q_n k'}$ *with* $q_1, \dots, q_n$, $k' \in K$ *such that, for* $i \in [1, n]$, $\mathsf{Max} \in \mathsf{Level}(q_i)$, $\mathsf{Max} \in \mathsf{Level}(k')$, $k' \notin \mathscr{L}$ *and* $v_k \leq v_{k'} + \delta_k$,

(3) $\mathtt{L_v} \geq_s \{l \mid \exists k \in \mathscr{L} \cap \mathscr{U}_E \text{ s.t. } l \in \mathsf{Level}(k)\})$ *where $S_1 \geq_s S_2$ if for all $l_2 \in S_2$, there exists $l_1 \in S_1$ such that $l_1 \geq l_2$.*

Robustness is meant to precisely control how compatible keys occur in a global state E. It ensures two main properties. First, compatible keys have a unique attribute in the system, that is, levels and validity times of compatible keys are consistently propagated through the system. Moreover, robustness ensures that compatible keys occur only in key position, except if they are encrypted by a key of greater level or if they occur in an administrator command (thus encrypted by a key of level Max). The third item is a technical condition that ensures that levels corresponding to lost (and not yet dead) keys should always be considered as compromised.

Whenever a state is robust, all its compatible keys remain secret from the attacker.

Proposition 4.2. *Let $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ be a global state, $\mathscr{L}$ a set of keys (typically lost keys), and $k \in K$. Let $\mathtt{L_v}$ be a set of levels. Assume that E is robust up to $\mathtt{L_v}$ and $\mathscr{L}$ then:*

$$\forall k \in \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})} \quad \mathfrak{M} \not\vdash k.$$

We show that good keys are never deducible by an attacker. More precisely, we show that the property of robustness is preserved under deduction.

Lemma 4.3. *The properties defined in Definition 4.5 are invariant under deduction.*

Proof. Let $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ be a global state robust up to $\mathtt{L_v}$ and $\mathscr{L}$. We consider E' another global state such that $E \xrightarrow{\text{(DED)}} E'$ and we prove that E' verifies properties (1), (2) and (3). To do so, we show that if E verifies (1), (2) and (3) for $\mathtt{L_v}$ and $\mathscr{L}$, then $E_p = (\mathcal{P}, \mathcal{I}, \mathfrak{M}_p, N, K, t)$ where $\mathfrak{M}_p = \mathfrak{M} \cup \{p\}$ with $\mathfrak{M} \vdash p$ in one step also verifies (1), (2) and (3) for $\mathtt{L_v}$ and $\mathscr{L}$. We can then conclude by induction.

(1) Let (l, v, m) be attributes of k in E_p and let us show that they must be attributes of k in E already.

- If (l, v, m) are attributes of k by case (a) in E_p, then, since the transition does not affect handles, we must have that (l, v, m) were attributes of k by case (a) in E too.
- If (l, v, m) are attributes of k by case (b) in E_p, we know that $\exists\ q \in K$, $m \in \mathfrak{M}_p$ s.t. $\mathsf{Max} \in \mathsf{Level}(q)$ and $\{\mathsf{cmd}, \dots, k, l_2, v_2, m_2, \dots\}_{q_1 \cdots q_j q} \in \mathrm{St}(m)$ with $\mathsf{Max} \in \mathsf{Level}(q_i)$. We consider two subcases:
  - If $m \in \mathfrak{M}$, then (l, v, m) were already attributes of k in E.
  - If $m = p$. Then we have two sub-subcases:
    - The deduction rule is a decomposition rule or the rule $\dfrac{m \in \mathfrak{M}}{\mathfrak{M} \vdash m}$. Then, we have that $m \in \mathrm{St}(\mathfrak{M})$ and we can conclude as in the previous case.
    - The deduction rule is a composition rule. If the rule $\dfrac{x \in \mathfrak{M}\ y \in \mathfrak{M}}{\mathfrak{M} \vdash \langle x, y \rangle}$ is applied, then $\{\mathsf{cmd}, \dots, k, l_2, v_2, m_2, \dots\}_{q_1 \cdots q_j q}$ must be subterm of either x or y and we can conclude. If the rule is $\dfrac{x \in \mathfrak{M}\ y \in \mathfrak{M}}{\mathfrak{M} \vdash \{x\}_y}$, then we have two possibilities:
      - $\{\mathsf{cmd}, \dots, k, l, v, m, \dots\}_{q_1 \cdots q_j q}$ is subterm of $x \in \mathfrak{M}$ and we conclude.
      - $m = \{\mathsf{cmd}, \dots, k, l, v, m, \dots\}_{q_1 \cdots q_j q}$ and then we have $y = q$ and $x = \{\mathsf{cmd}, \dots, k, l, v, m, \dots\}_{q_1 \cdots q_j}$. In that case, we must have $j \geq 1$ since $j = 0$ would lead to a contradiction in E. Indeed, we would have $k \in \mathcal{G}^{\mathtt{L_v}}_{(E_p, \mathscr{L})}$, i.e. $k \in \mathcal{G}^{\mathtt{L_v}}_{(E, \mathscr{L})}$ s.t. $\exists\ m \in \mathfrak{M}$ with an occurrence of k not satisfying (2) in E. Thus, $j \geq 1$. Then, using induction hypothesis over x, we know that (l, v, m) were already attributes of k in E.
- If (l, v, m) are attributes of k by case (c) in E_p, we know that $\exists\ q \in K$, $m \in \mathfrak{M}_p$ s.t. $\{\dots, k, l, v, m, \dots\}_q \in \mathrm{St}(m)$. We have two subcases:
  - If $m \in \mathfrak{M}$, then (l, v, m) were already attributes of k in E.
  - If $m = p$. Then we have two sub-subcases:
    - The deduction rule is a decomposition rule or the rule $\dfrac{m \in \mathfrak{M}}{\mathfrak{M} \vdash m}$. Then, we have that $m \in \mathrm{St}(\mathfrak{M})$ and we can conclude as in the previous case.
    - The deduction rule is a composition rule. If the rule $\dfrac{x \in \mathfrak{M}\ y \in \mathfrak{M}}{\mathfrak{M} \vdash \langle x, y \rangle}$ is applied, then $\{\dots, k,\ l_2, v_2, m_2, \dots\}_q$ must be subterm of either x or y and we can conclude. If the rule is $\dfrac{x \in \mathfrak{M}\ y \in \mathfrak{M}}{\mathfrak{M} \vdash \{x\}_y}$, then we have two possibilities:
      - $\{\dots, k, l, v, m, \dots\}_{q_1 \cdots q_j q}$ is subterm of $x \in \mathfrak{M}$ and we conclude.

⋆ $m = \{\dots, k, l, v, m, \dots\}_q$ and we have $y = q$ and $x = \{\dots, k, l, v, m, \dots\}$. But, since $x \in \mathfrak{M}$ and $k \in \mathcal{G}_E^{\mathtt{L_v}}$, this is a contradiction since x does not satisfy (2).

(2) Let $k \in \mathcal{G}^{\mathtt{L_v}}_{(E_p,\mathscr{L})}$, $m \in \mathfrak{M}_p$ s.t. $k \in \mathrm{St}(m)$. If $m \in \mathfrak{M}$, we the use induction hypothesis and we can conclude easily. Now we consider $m = p$. We consider different cases according to the applied deduction rule:

- If the rule is $\dfrac{}{\mathfrak{M} \vdash m}$, then the result is obvious.
- If the rule is $\dfrac{m_1 \in \mathfrak{M}\ m_2 \in \mathfrak{M}}{\mathfrak{M} \vdash \langle m_1, m_2 \rangle}$, then since k occurs in $m = \langle m_1, m_2 \rangle$, k occurs in m_1 or/and m_2. Since $m_1, m_2 \in \mathfrak{M}$, using the induction hypothesis, we know that m_1 and m_2 are verifying (2), thus, it is clear that m also verifies (2).
- If the rule is $\dfrac{\langle m_1, m_2 \rangle \in \mathfrak{M}}{\mathfrak{M} \vdash m_i}$, then since m is a subterm of a term already verifying (2) using the induction hypothesis, we obviously have that m verifies (2).
- If the rule is $\dfrac{x \in \mathfrak{M}\ y \in \mathfrak{M}}{\mathfrak{M} \vdash \{x\}_y}$, then as k occurs in $m = \{x\}_y$, k may occur in x or/and y. If $y = k$ (k occurs in y), we have that $k \in \mathfrak{M}$, which contradicts (2). Then k can only occurs in x and since $x \in \mathfrak{M}$, using the induction hypothesis, we have that x, and by extension, m satisfies (2).
- If the rule is $\dfrac{\{x\}_y \in \mathfrak{M}\ y \in \mathfrak{M}}{\mathfrak{M} \vdash x}$, then k occurring in $m = x$ implies that k occurs in $\{x\}_y$. Since $\{x\}_y \in \mathfrak{M}$, using the induction hypothesis, we know that it verifies (2). Moreover, since $y \in \mathfrak{M}$ we have that $y = k$ leads to a contradiction with (2) in E. We consider three subcases:
  - In the first subcase, k appears in key position in $\{x\}_y$ and as we have just seen it, it can not be the case $y = k$, then $\exists\, m'$ s.t. $\{m'\}_k$ occurs in x. In that case, this occurrence of k in x still satisfies (2.i).
  - In the second subcase, we have $\{\dots, k, \dots\}_{k'} \in \mathrm{St}(\{x\}_y)$ with $k' \in K$ s.t. $v_k \leq v_{k'} + \delta(l)$ with $l = \mathsf{Level}(k) < \mathsf{Level}(k')$ and $\mathsf{Max} \notin \mathsf{Level}(k')$ (k appears in $\{x\}_y$ according to (2.ii)). If $\{\dots, k, \dots\}_{k'} \in \mathrm{St}(x)$ we can conclude easily, the most complicated case is when $\{\dots, k, \dots\}_{k'} = \{x\}_y$. In that case, since $v_k \leq v_{k'} + \delta(l)$ and $k \in \mathscr{U}_{E_p}$, we have that $k' \in \mathscr{U}_{E_p}$, indeed, if we consider $l' = \mathsf{max}(\mathsf{Level}(k'))$, we have:

$$\begin{aligned} v_{k'} + \Delta(l') &= v_{k'} + \delta(l) - \delta(l) + \Delta(l'), \\ &\geq v_k + \Delta(l') - \delta(l), \\ &\geq v_k + \Delta(l), \text{ since } l < l' \\ &> t. \end{aligned}$$

    Now, with $k \in \mathcal{G}^{\mathtt{L_v}}_{(E_p,\mathscr{L})}$ and that $k' \in \mathscr{U}_{E_p}$, we have that $k' \notin \mathscr{L}$ otherwise, using property (3), there would be some level l' in $\mathtt{L_v}$ corresponding to k' and

since $\mathsf{Level}(k) < \mathsf{Level}(k')$, k would not be in $\mathcal{G}^{\mathtt{L_v}}_{(E_p,\mathscr{L})}$. Using this fact and that $\mathsf{Level}(k) < \mathsf{Level}(k')$ we also have that $\nexists\, i,j$ s.t. $l_i < l_j$ with $l_i \in \mathsf{Level}(k')$ and $l_j \in \mathtt{L_v}$. Thus, we have that $k' \in \mathcal{G}^{\mathtt{L_v}}_{(E_p,\mathscr{L})}$, especially $k' \in \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$ (the transition does not affect time or the set of lost keys). Then, $y = k'$ would implies that $k' \in \mathfrak{M}$ which is a contradiction with (2).

– In the third subcase, we have $\{\ldots, k, \ldots\}_{q_1 \cdots q_n k'} \in \mathrm{St}(\{x\}_y)$ with $q_1, \ldots, q_n$, $k' \in K$ s.t., for $i \in [1,n]$, $\mathsf{Max} \in \mathsf{Level}(q_i)$, $k' \notin \mathscr{L}$, $\mathsf{Max} \in \mathsf{Level}(k')$ and $v_k \leq v_{k'} + \delta(l)$ with $l = \mathsf{Level}(k)$ (The occurrence of k is of the form (2.iii)). As always if the whole occurrence is included in x, the conclusion is easy. We consider the worst case where there is the equality. Using the argument above on validities, we know that $k' \in \mathscr{U}_{E_p}$, i.e. in $\mathscr{U}_E$, and since it is not lost and that $\mathsf{Max} \in \mathsf{Level}(k')$, we have that $k' \in \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$. Then, using Lemma 4.2, we know that $E \nvdash k'$ and then that $y = k'$ leads to a contradiction.

(3) Since $\mathscr{L}' = \mathscr{L}$ and $\mathscr{U}_E = \mathscr{U}_{E'}$, we have that $\{l \mid \exists k \in \mathscr{L} \cap \mathscr{U}_E \text{ s.t. } l \in \mathsf{Level}(k)\} = \{l \mid \exists k \in \mathscr{L}' \cap \mathscr{U}_{E'} \text{ s.t. } l \in \mathsf{Level}(k)\}$ and (3) holds obviously with the same $\mathtt{L_v}$.

□

We can now easily prove Proposition 4.2.

Proposition 4.2. *Let $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ be a global state, $\mathscr{L}$ a set of keys (typically lost keys), and $k \in K$. Let $\mathtt{L_v}$ be a set of levels. Assume that E is robust up to $\mathtt{L_v}$ and $\mathscr{L}$ then:*

$$\forall k \in \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})} \quad \mathfrak{M} \nvdash k.$$

Proof. Let us consider that $\exists\, k \in \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$ such that $E \vdash k$, i.e. $\mathfrak{M} \vdash k$. According to this, $\exists\, E'$ a global state such that $E \xrightarrow{\text{(DED)}} E'$ where $\mathfrak{M}' = \mathfrak{M} \cup \{k\}$. Since E is robust up to $\mathtt{L_v}$ and $\mathscr{L}$, using Lemma 4.3, we know that E' is robust up to $\mathtt{L_v}$ and $\mathscr{L}$. But $k \in \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$ implies $k \in \mathcal{G}^{\mathtt{L_v}}_{(E',\mathscr{L})}$ and since $k \in \mathfrak{M}'$, then E' cannot satisfies (2). Contradiction.

□

4.4.4 Main Results

We describe here three different properties that provide security to the design:

- A global state is always robust up to the levels of lost keys (Theorem 4.2). This means that losing a key does not compromise any other key apart from those at lower levels than the lost key, i.e. those which are trivially compromised.
- Once a lost key is dead, the corresponding level is protected again (Theorem 4.3), meaning that even when a key is lost and no special action is taken, the system repairs itself after a certain time.
- When a TRD receives a blacklist command, the corresponding levels are immediately protected again on this TRD (Theorem 4.4).

Theorem 4.2. *Let E be a global state s.t. $E_\emptyset \longrightarrow^* E$ and let $\mathscr{L}$ be the set of keys lost along the execution $E_\emptyset \longrightarrow^* E$. Then E is robust up to $\mathtt{L_v} = \{l \mid \exists\ \mathsf{Lost}(k) \in \mathscr{L}$ and $\mathsf{Level}(k) = l\}$ and $\mathscr{L}$.*

Theorem 4.2 is the key theorem of this work and its proof consists in proving that robustness is invariant under the rules of our API. Theorem 4.1 ensures that it is impossible for an attacker to break more than $N_{\mathsf{Max}} - 1$ keys of level Max. This, in turn, ensures that all accepted key management commands come from the administrator. Then it remains to show that the robustness property is invariant under application of the other rules of the API. For the deduction rule, the first step is to show that compatible (latent) keys are all non-deducible. For decryption, the robustness property in the global state E ensures that encrypted messages have the right form. Therefore when decrypting a message, the newly stored keys also satisfy the invariant. In case the attacker forges a message, the checks on the levels and the validity times ensure that the newly stored keys are of an incompatible level and therefore their security does not need to be assessed. The arguments for encryption and the other commands are along the same lines.

The proof of this theorem by induction. The initial case is obvious and the general case is proved by the following proposition.

Proposition 4.3. *Let us consider global states E and E' such that $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ and $E' = (\mathcal{P}', \mathcal{I}', \mathfrak{M}', N', K', t')$ with $E_\emptyset \longrightarrow^* E \xrightarrow{L} E'$. We suppose that E is robust up to $\mathtt{L_v} = \{l \mid \exists\ \mathsf{Lost}(k) \in \mathscr{L}$ and $\mathsf{Level}(k) = l\}$ and $\mathscr{L}$. If $L \neq \mathsf{Lost}(k)$, then E' is robust up to $\mathtt{L_v}$ and $\mathscr{L}$, otherwise E' is robust up to $\mathtt{L'_v} = \mathtt{L_v} \cup \mathsf{Level}(k)$.*

Proof. We will show that (1), (2) and (3) are invariant under application of transition rules:

- If $E \xrightarrow{\text{(DED)}} E'$. This step is already proven in Lemma 4.3.
- If $E \xrightarrow{\text{(TIM)}} E'$. During this transition, only some latent keys may become dead since $t_{E'} > t_E$. In that case, we have $\mathcal{G}^{\mathtt{L_v}}_{(E',\mathscr{L})} \subseteq \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$, moreover, by inspection, we have that existential quantifications over keys of properties (1) and (2) are stable by time passing. (Time does not turn non lost keys into lost keys.) Thus, we conclude using the induction hypothesis on E. Finally, since we may only have $\{l \mid \exists k \in \mathscr{L} \cap \mathscr{U}_E$ s.t. $l \in \mathsf{Level}(k)\} \supseteq \{l \mid \exists k \in \mathscr{L} \cap \mathscr{U}_{E'}$ s.t. $l \in \mathsf{Level}(k)\}$, (3) holds obviously.
- If $E \xrightarrow{\text{(LST)}} E'$. Then, we consider $\mathtt{L'_v} = \mathtt{L_v} \cup \mathsf{Level}(k)$ and $\mathscr{L}' = \mathscr{L} \cup \{k\}$ with k the key lost during the transition. Considering this, we have that $\mathcal{G}^{\mathtt{L'_v}}_{(E',\mathscr{L}')} \subseteq \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$ and, since, by inspection, existential quantifications over keys in property (1) are stable under the lost transition, using the induction hypothesis on E, (1) holds easily.

  For property (2), There are two issues. The first one concerns the case where $k_1 \in \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$ s.t. we have $\{\ldots, k_1, \ldots\}_k \in \mathrm{St}(\mathfrak{M})$ with $k \in K$ and $\mathsf{Level}(k_1) = l_1 < \mathsf{Level}(k)$, $\mathsf{Max} \notin \mathsf{Level}(k)$ and $v_{k_1} \leq v_k + \delta(l_1)$. But since we lost k during the transition, then k_1 is no more in $\mathcal{G}^{\mathtt{L'_v}}_{(E',\mathscr{L}')}$ and then there is nothing to verify. The second problem appears when we have $k_1 \in \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$ s.t. $\{\ldots, k_1, \ldots\}_{q_1 \cdots q_n k} \in \mathrm{St}(\mathfrak{M})$ with $q_1, \ldots, q_n, k \in K$ s.t., for $i \in [1, n]$, $\mathsf{Max} \in \mathsf{Level}(q_i)$, $k \notin \mathscr{L}$, $\mathsf{Max} \in \mathsf{Level}(k')$ and $v_{k_1} \leq v_k + \delta_{l_1}$ with $l_1 = \mathsf{Level}(k_1)$. Even if we lose k during the transition, we may have that k_1 remains in $\mathcal{G}^{\mathtt{L'_v}}_{(E',\mathscr{L}')}$. Let us suppose

that, then, using the last inequality, we have:

$$\begin{aligned} v_{k_1} + \Delta(l_1) &\leq v_k + \delta(l_1) + \Delta(l_1), \\ &\leq v_k + \Delta(\mathsf{Max}), \text{ since } l_1 < \mathsf{Max} \in \mathsf{Level}(k'). \end{aligned}$$

This shows that $k \in \mathscr{U}_{E'}$ (i.e. in $\mathscr{U}_E$) since, if it is not, then $v_k + \Delta(\mathsf{Max}) \leq t$ which implies $v_{k_1} + \Delta(\mathsf{Max}) \leq t$ and, especially, that k_1 is dead implying a contradiction with the fact that $k_1 \in \mathcal{G}^{\mathrm{L_v}}_{(E',\mathscr{L}')}$. Moreover, since $k \notin \mathscr{L}$ in E, we know that k was in $\mathcal{G}^{\mathrm{L_v}}_{(E,\mathscr{L})}$ and, using Lemma 4.2, we have that $E \not\vdash k$. Let $m \in \mathfrak{M}$ containing the occurrence of k_1 we focus on. Then, we have two possibilities: m is an admin message or m was deduced by the intruder. In the second case, we have that $\exists\, m' \in \mathfrak{M}$ an admin message such that $\{m', t_1, \ldots, t_n\} \vdash m$ with $t_1, \ldots, t_n \in \mathfrak{M}$ and $\{\ldots, k_1, \ldots\}_{q_1 \cdots q_n k} \in \mathrm{St}(m')$. Otherwise, it would implies that the intruder forged m containing an undeducible revocation key without any admin message, which is impossible. Then, in both cases, we have that $\exists\, m \in \mathfrak{M}$ an admin message s.t. $\{\ldots, k_1, \ldots\}_{q_1 \cdots q_n k} \in \mathrm{St}(m)$. We distinguish three subcases:

- If we have $n > N_{\mathsf{Max}} - 1$, we have a term with more than N_{Max} revocation keys subterm of a admin message which is in contradiction with Hypothesis 2 since the specification requires N_{Max} revocation keys, no more.

- If we have $n = N_{\mathsf{Max}} - 1$, then $m = \{\ldots, k, \ldots\}_{q_1 \cdots q_n k}$. Suppose that $\forall i \in [1, n]$, $q_i \in \mathscr{L}$. Then, if no keys are dead, we have N_{Max} latent revocation keys lost, which is a contradiction to Hypothesis 1, since all these revocation keys belong to one TRD. Then, one key is dead. If k is dead, we already saw that k_1 is dead too and we have a contradiction. If $\exists\, j \in [1, n]$ such that q_j is dead, we have:

  $$\begin{aligned} v_{k_1} + \Delta(l_1) &= v_{k_1} - v_{q_j} + v_{q_j} + \Delta(l_1), \\ &\leq \delta(l_1) + v_{q_j} + \Delta(l_1), \\ &\leq v_{q_j} + \Delta(\mathsf{Max}), \\ &\leq t. \end{aligned}$$

  Indeed, we have $v_{k_1} - v_{q_j} \leq \delta(l_1)$ since at the time of emission of the message, time t_e, we know that $v_{k_1} \leq t_e + \delta(l_1)$ and that $t_e < v_{q_j}$ since q_j was necessarily valid. Thus, in that case, we also have a contradiction on k_1, then, we must have that $\exists\, i \in [1, n]$ such that $q_i \notin \mathscr{L}$. Moreover, we have that $v_{k_1} \leq v_{q_j} + \delta(l_1)$ using the two previous inequalities. Then, we show that the occurrence of k_1 still verifying (2.iii) after the loss of k.

- If $n < N_{\mathsf{Max}} - 1$. We have $m = \{\ldots, k, \ldots\}_{q_1 \cdots q_n k' q'_1 \cdots q'_p}$ with $n + p = N_{\mathsf{Max}} - 1$ and $\forall i \in [1, p]$, $\mathsf{Level}(q'_i) = \mathsf{Max}$ and $E \vdash q'_i$ for $i \in [1, p]$. ($\{\ldots, k, \ldots\}_{q_1 \cdots q_n k'}$ is deduced from m, then it implies that the keys 'protecting' this term were deducible otherwise there will be a contradiction.) This implies, for each $i \in [1, p]$ that either $q'_i \in \mathscr{L}$ or q'_i is dead. If q'_i is dead, we prove, using the argument with the inequalities above, that k_1 is dead, which is a contradiction with the fact that $k_1 \in \mathcal{G}^{\mathrm{L}'_{\mathrm{v}}}_{(E',\mathscr{L}')}$, thus, we have that $\forall i \in [1, p]$, $q'_i \in \mathscr{L}$. Now, we can apply the previous case ($n = N_{\mathsf{Max}} - 1$) to m in order to prove that $\exists\, i \in [1, n]$ s.t. $q_i \notin \mathscr{L}$ and $v_{k_1} \leq v_{q_i} + \delta(l_1)$. Thus, we show that the occurrence of k_1 still verifying (2.iii).

For property (3), $\{l \mid \exists k \in \mathscr{L}' \cap \mathscr{U}_{E'} \text{ s.t. } l \in \mathsf{Level}(k)\} = \{l \mid \exists k \in \mathscr{L} \cap \mathscr{U}_E \text{ s.t. } l \in \mathsf{Level}(k)\} \cup \mathsf{Level}(k)$ and since $\mathtt{L}'_\mathtt{v} = \mathtt{L}_\mathtt{v} \cup \mathsf{Level}(k)$ and $\mathtt{L}_\mathtt{v} \geq_s \{l \mid \exists k \in \mathscr{L} \cap \mathscr{U}_E \text{ s.t. } l \in \mathsf{Level}(k)\}$, (3) holds.

- If $E \xrightarrow{\text{(UPM)}} E'$. This transition only affects handles containing revocation keys, implying that we have $\mathcal{G}^{\mathtt{L}_\mathtt{v}}_{(E',\mathscr{L}')} = \mathcal{G}^{\mathtt{L}_\mathtt{v}}_{(E,\mathscr{L})}$ and, according to the semantic, we also have $\mathfrak{M}' = \mathfrak{M}$. Moreover, by inspection and using hypothesis 2 on admin commands, we have that existential quantifications over keys of properties (1) and (2) are stable by these transitions. Then, it is clear, that, using the induction hypothesis on E, we have that (1) and (2) holds for E'. Finally, since $\mathscr{L} = \mathscr{L}'$, (3) holds obviously.

- If $E \xrightarrow{\text{(RVK)}} E'$. In this case, we remove a certain number of handles (keys) from a TRD and we have that $\mathfrak{M}' = \mathfrak{M} \cup \{m\}$ where m does not contain key values. Then, using the induction hypothesis, we have that (1) and (2) holds. Finally, since $\mathscr{L} = \mathscr{L}'$, (3) holds obviously.

- If $E \xrightarrow{\text{(BLK)}} E'$. In this case, we remove a certain number of handles (keys) from a TRD and we have that $\mathfrak{M}' = \mathfrak{M} \cup \{m\}$ where m contains no key values, i.e. $\forall k \in \mathcal{G}^{\mathtt{L}_\mathtt{v}}_{(E',\mathscr{L}')}$ s.t. $k \notin \mathrm{St}(m)$. Considering this, and the induction hypothesis, it is obvious that (1) and (2) holds for E'. Finally, since $\mathscr{L} = \mathscr{L}'$, (3) holds obviously.

- If $E \xrightarrow{\mathrm{L}} E'$ with $L \in \{\text{(NEW),(UPD)}\}$. In this case, we have $\mathcal{G}^{\mathtt{L}_\mathtt{v}}_{(E',\mathscr{L}')} \supseteq \mathcal{G}^{\mathtt{L}_\mathtt{v}}_{E}$ and $\mathfrak{M}' = \mathfrak{M} \cup \{m\}$ where $m = \{m'\}_{q_1 \cdots q_r}$ with $q_1, \ldots, q_r$ valid revocation keys for the concerned TRD, and then, using Hypothesis 1, $\exists\, j \in [1, r]$ s.t. $q_r \notin \mathscr{L}'$.

  (1) Let $k \in \mathcal{G}^{\mathtt{L}_\mathtt{v}}_{(E',\mathscr{L}')}$. If k appears in m with no attributes (in the case where k will be updated by the command), then, using the induction hypothesis is enough to conclude. If k appears with attributes (then it is the new key created or updated by the command), then let us consider two cases:

  – Either k has never occurred before in $\mathfrak{M}$ (in the commands that have been previously executed), then (1) holds obviously.

  – Or k has occurred before in $\mathfrak{M}$. Then, during the evolution $E_\emptyset \longrightarrow^* E$, we consider the increasing sequence of history $(\mathfrak{M}_i)_i$ and consider the first one such that k does not appear in $\mathfrak{M}_{i-1}$ and k appears in $\mathfrak{M}_i$. To set a fixed lower level key in a TRD, there are three ways: a create, update or decrypt command. Since the two first commands require N_{Max} valid revocation keys, if k appears with such a command, this one must have been done by the administrator according to Hypothesis 1 and Lemma 4.2. The use of a decrypt command implies that the command comes from the intruder (an administrator will not use this way to put k in a TRD) but this would be in contradiction with the fact that $k \in \mathcal{G}^{\mathtt{L}_\mathtt{v}}_{(E',\mathscr{L}')}$ since k would be lost from the start. Thus k was set up by an admin command m', and, using the Hypothesis 2, we know that the attributes associated to k in m are the same to those which are associated to k in m'. Now, using the induction hypothesis in E, we know that all previous attributes associated to k are the same as those in m', which are the same as those in the new handle and m. Thus, (1) holds.

(2) Let $k \in \mathcal{G}^{\mathsf{Lv}}_{(E',\mathscr{L}')}$ s.t. $\exists\, m' \in \mathfrak{M}'$ s.t. $k \in \mathrm{St}(m')$. We have the same discussion as is the previous transition: there is at least one revocation key q which is not lost in m and we must have $v_k \leq v_q + \delta(l)$ where $l = \mathsf{Level}(k)$. Then, (2) holds.

(3) Finally, since $\mathscr{L} = \mathscr{L}'$, (3) holds obviously.

- If $E \longrightarrow E'$. Let us consider four subcases:
  - The transition corresponds to the generatePublic or generateSecret command. For the first one, only information of level 0 is created and then properties hold obviously. The generateSecret command creates fresh keys, thus (1) holds and since $\mathfrak{M}' = \mathfrak{M}$ we have also that (2) holds. Finally, since $\mathscr{L} = \mathscr{L}'$, (3) holds obviously.
  - The transition corresponds to the blacklist, create, update, revoke or updateMax command. This implies that the intruder may forge an admin message, and, therefore, can know N_{Max} different and valid revocation keys for the API, which is not possible according to Hypothesis 1 and the Theorem 4.1. Finally, since $\mathscr{L} = \mathscr{L}'$, (3) holds obviously.
  - The transition corresponds to the encrypt command. For property (1), the command only uses handles, or information of level 0, already existing, then, using induction hypothesis on E, we can conclude that the property holds. For property (2), we have that $\mathfrak{M}' = \mathfrak{M} \cup \{\{U_1, \ldots, U_n\}_k\}$ with U_i a level 0 message or a (k_i, l_i, v_i, m_i) set with $1 \leq l_i < \mathsf{Max}$. Suppose that $\exists\, i \in [1, n]$ s.t. $k_i \in \mathcal{G}^{\mathsf{Lv}}_{(E',\mathscr{L}')}$, then, according to the specification, we have that $\mathsf{Level}(k_i) < \mathsf{Level}(k)$, $\mathsf{Max} \notin \mathsf{Level}(k)$ and $v_k > t$, $v_{k_i} > t$. Thus, we must have $v_{k_i} \leq v_k + \delta(l_i)$ otherwise, we have $v_{k_i} > v_k + \delta(l_i) > t + \delta(l_i)$ which is impossible. Indeed, let us consider the first apparition, as in a previous case, of k_i in the system. We have that k_i appeared from a create, update, decrypt or generateSecret command. But, whatever the applied command, the TRD does a test on validity (or sets it itself) such that $v_{k_i} \leq t_c + \delta(l_i)$ where t_c is the time of the state where k_i appears for the first time. Since $t + \delta(l_i) > t_c + \delta(l_i)$, having $v_{k_i} > v_k + \delta(l_i)$ leads to a contradiction. Then (2) holds. Finally, since we still have $\mathscr{L} = \mathscr{L}'$, (3) holds obviously.
  - The transition corresponds to the decrypt command. We have that $\mathfrak{M}'$ does not contain information of a level greater or equal to 1. Moreover, we add a key in the API concerned by the decrypt command and $\mathcal{G}^{\mathsf{Lv}}_{(E',\mathscr{L}')}$ may be larger than $\mathcal{G}^{\mathsf{Lv}}_{(E,\mathscr{L})}$. Let us consider $k, q \in K'$ s.t. $(\mathsf{decrypt}, \{\ldots, k, l, v, m, \ldots\}_q, h_q)$, where h_q is the handle pointing to q in the TRD, is the message implying the considered decrypt command. Suppose that $k \in \mathcal{G}^{\mathsf{Lv}}_{(E',\mathscr{L}')} \setminus \mathcal{G}^{\mathsf{Lv}}_{(E,\mathscr{L})}$. Then, the message is coming from the intruder, otherwise, k should already appear in a handle of another TRD and we should have $k \in \mathcal{G}^{\mathsf{Lv}}_{(E,\mathscr{L})}$. Since the message comes from the intruder, it implies that $E \vdash q$. Using Proposition 4.2, we must have that $q \notin \mathcal{G}^{\mathsf{Lv}}_{(E,\mathscr{L})}$. But we know that the message was accepted by the TRD, then, q must be valid (and then latent), then, if $q \notin \mathcal{G}^{\mathsf{Lv}}_{(E,\mathscr{L})}$, we must have that $q \in \mathscr{L}$ or $\exists\, q' \in K \cap \mathscr{L}$ and $\mathsf{Level}(q) \leq \mathsf{Level}(q')$. Adding the fact that we must have $\mathsf{Level}(k) < \mathsf{Level}(q)$, we must have $k \notin \mathcal{G}^{\mathsf{Lv}}_{(E',\mathscr{L}')}$ which is a

contradiction. Thus, k was already in $\mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$ and $\mathcal{G}^{\mathtt{L_v}}_{(E',\mathscr{L}')} = \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$ which, using the induction hypothesis, implies that E' verifies (2).

For property (1), we consider two cases:

* If k, the key used to decrypt, is in $\mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$. Then, we know, using Proposition 4.2, that $E \nvdash k$. Thus, the message was already existing in $\mathfrak{M}$ and verifies property (1) by induction hypothesis.

* If k, the key used to decrypt, is not in $\mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$. Then, it must be the case that $k \in \mathscr{L}$ or k is under a lost key (and may be considered as lost.) since it must be valid (then, a fortiori, latent). Since all keys q in the encrypted message must verify $\mathsf{Level}(q) < \mathsf{Level}(k)$, then, $q \notin \mathcal{G}^{\mathtt{L_v}}_{(E,\mathscr{L})}$, i.e. $q \notin \mathcal{G}^{\mathtt{L_v}}_{(E',\mathscr{L}')}$ and then does not interfere with property (1).

Finally, since $\mathscr{L} = \mathscr{L}'$, (3) holds obviously.

□

Theorem 4.3. *Let E and E' be two global states such that $E_\emptyset \longrightarrow^* E \longrightarrow^* E'$. Assume that E is robust up to $\mathtt{L_v}$ and $\mathscr{L}$ and let $l \in \mathtt{L_v}$ be a compromised level. Let $\mathscr{L}'$ be the set of keys lost along the execution $E \longrightarrow^* E'$. We further assume that there has been no 'lost' event for keys of level greater than l, that is, there is no $k \in \mathscr{L}'$ such that $\mathsf{Level}(k) \geq l$. Let $v = \mathsf{sup}(\{v_k \mid k \in \mathscr{L} \text{ and } \mathsf{Level}(k) = l\})$, where v_k is the validity time associated to k (Theorem 4.2 ensures v_k is unique). Then:*

$$t' \geq v + \Delta(l) \Longrightarrow E'' \text{ is robust up to } \mathtt{L}''_{\mathtt{v}} \text{ for } \mathscr{L}'',$$

where t' is the global time of E' and E'' is such that $E' \xrightarrow{\mathsf{Lost}(A)} E''$ with $A = \{k \mid \mathsf{Level}(k) = l' \text{ with } l' < l\}$ and $\mathtt{L}''_{\mathtt{v}} = (\mathtt{L_v} \cup \bigcup_{k \in \mathscr{L}''} \mathsf{Level}(k)) \setminus \{l\}$ where $\mathscr{L}'' = \mathscr{L}' \cup A$.

Intuitively, Theorem 4.3 ensures that whenever a key k has been lost then once k is not latent any more (*i.e.* once k is dead), the system is now robust up to $\mathtt{L_v} \setminus \{l\}$ plus the levels $\{l' \mid l' < l\}$ which can still be compromised and the newly lost keys $\mathscr{L}'$, that is, the keys that might have been lost during the execution between E and E'. Theorem 4.3 is actually a consequence of Theorem 4.2, which can be seen by carefully analysing the robustness property and noticing that once the lost key k is dead, the set of keys compatible with $\mathtt{L}''_{\mathtt{v}}$ and $\mathtt{L_v}$ coincide (if the keys of lower levels are explicitly lost).

Proof. We know, using Theorem 4.2, that E' is robust up to $\mathtt{L}'_{\mathtt{v}} = \mathtt{L_v} \cup \bigcup_{k \in \mathscr{L}'} \mathsf{Level}(k)$ and $\mathscr{L}'$. Using Theorem 4.2 again, we have that E'' is robust up to $L = \mathtt{L}'_{\mathtt{v}} \cup \{l' \mid l' < l\} = \mathtt{L_v} \cup \bigcup_{k \in \mathscr{L}''} \mathsf{Level}(k)$ and $\mathscr{L}'' = \mathscr{L}' \cup A$.

Now, if we consider $\mathtt{L}''_{\mathtt{v}} = (\mathtt{L_v} \cup \bigcup_{k \in \mathscr{L}''} \mathsf{Level}(k)) \setminus \{l\} = L \setminus \{l\}$, we have that $\mathcal{G}^{L}_{(E'',\mathscr{L}'')} =$

$\mathcal{G}^{\mathtt{L}''_{\mathtt{v}}}_{(E'',\mathscr{L}'')}$. Indeed, we have:

$$\begin{aligned}\mathcal{G}^{L}_{(E'',\mathscr{L}'')} &= \{k \mid \mathsf{Max} \notin \mathsf{Level}(k), k \in \mathscr{U}_{E''} \setminus \mathscr{L}'',\ \exists l' \in \mathsf{Level}(k) \text{ s.t. } l' \not< L\} \\ &= \{k \mid \mathsf{Max} \notin \mathsf{Level}(k), k \in \mathscr{U}_{E''} \setminus \mathscr{L}'',\ \exists l' \in \mathsf{Level}(k) \text{ s.t. } l' \not< \mathtt{L}''_{\mathtt{v}}\} \\ &\setminus \{k \mid \mathsf{Max} \notin \mathsf{Level}(k), k \in \mathscr{U}_{E''} \setminus \mathscr{L}'',\ \exists l' \in \mathsf{Level}(k) \text{ s.t. } l' < l\} \\ &= \mathcal{G}^{\mathtt{L}''_{\mathtt{v}}}_{(E'',\mathscr{L}'')} \setminus G,\end{aligned}$$

where $G = \{k \mid \mathsf{Max} \notin \mathsf{Level}(k), k \in \mathscr{U}_{E''} \setminus \mathscr{L}'', \exists l' \in \mathsf{Level}(k) \text{ s.t. } l' < l\}$.

But, in E'', $\forall k$ s.t. $l' \in \mathsf{Level}(k)$ with $l' < l$, then $k \in \mathscr{L}''$. Then, $G = \emptyset$. Thus, we have that E'' verifies (1) and (2) for $\mathtt{L}''_{\mathtt{v}}$. Moreover, we know that $\forall k \in \mathscr{L}''$ s.t. $\mathsf{Level}(k) = l$, then $k \notin \mathscr{U}_{E''}$, according to this, we have $\mathtt{L}''_{\mathtt{v}} \geq_s \{l_i \mid k_i \in \mathscr{L}'' \cap \mathscr{U}_{E''}$ where $l_i = \mathsf{Level}(k_i)\}$) using the fact above and the construction of $\mathtt{L}''_{\mathtt{v}}$. Thus E'' also ensures (3) for $\mathtt{L}''_{\mathtt{v}}$ and E'' is robust up to $\mathtt{L}''_{\mathtt{v}}$ for $\mathscr{L}''$. □

We now state the security gained with the blacklist command. This command only fixes the system locally. We therefore need to define robustness locally to a TRD.

Definition 4.6. *Let E be a global state, $\mathscr{L}$ be a set of lost keys, and $\mathtt{L}_{\mathtt{v}}$ be a set of levels. Let $a \in \mathcal{P}$ be a TRD. Then a is* locally robust up to $\mathtt{L}_{\mathtt{v}}$ *and $\mathscr{L}$ in E if $E_a = (\{a\}, \mathcal{I}_a, \mathfrak{M}, N, K, t)$ is robust up to $\mathtt{L}_{\mathtt{v}}$ where $\mathcal{I}_a$ is the restriction of $\mathcal{I}$ on $\{a\}$ and where the conditions of robustness are considered for keys that are stored on a (that is, appear in the image of Θ_a).*

Proposition 4.4 is the analog of Proposition 4.2 for locally robustness and states that compatible keys on locally robust TRDs are secure.

Proposition 4.4. *Let $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ be a global state, $\mathscr{L}$ a set of keys (typically lost keys), and $k \in K$. Let $\mathtt{L}_{\mathtt{v}}$ be a set of levels. Assume that a is locally robust up to $\mathtt{L}_{\mathtt{v}}$ and $\mathscr{L}$ in E, then:*

$$\forall k \in \mathcal{G}^{\mathtt{L}_{\mathtt{v}}}_{(E,\mathscr{L})} \cap \{k' \mid \exists h \text{ s.t. } \Theta_a(h) = (k', l, v, m)\} \quad \mathfrak{M} \not\vdash k.$$

Finally, we can state our final theorem that ensures that blacklisting a level protects the corresponding keys on a TRD.

Theorem 4.4. *Let $E = (\mathcal{P}, \mathcal{I}, \mathfrak{M}, N, K, t)$ a be a global state s.t. $E_\emptyset \longrightarrow^* E$. Let $\mathscr{L}$ be a set of (lost) keys. Assume that $a \in \mathcal{P}$ is locally robust up to $\mathtt{L}_{\mathtt{v}}$ and $\mathscr{L}$ in E. Assume that $E \xrightarrow{L} \longrightarrow^* E'$ with L a blacklist command for TRD a, level l, and validity time t_h (with t_h greater or equal to $v_h = \mathsf{sup}(\{v_k \mid k \in \mathscr{L}$ and $l \in \mathsf{Level}(k)\})$. Let $\mathscr{L}'$ be the set of keys lost along the execution $E \xrightarrow{L} \longrightarrow^* E'$. We further assume that there has been no lost event for keys of level greater than l, that is, there is no $k \in \mathscr{L}'$ such that $\mathsf{Level}(k) \geq l$.*

Then a is locally robust up to $\mathtt{L}''_{\mathtt{v}} = (\mathtt{L}_{\mathtt{v}} \cup \bigcup_{k \in \mathscr{L}''} \mathsf{Level}(k)) \setminus \{l\}$ in E'' where E'' is such that $E' \xrightarrow{\mathsf{Lost}(A)} E''$ with $A = \{k \mid \mathsf{Level}(k) = l'$ with $l' < l\}$ and $\mathscr{L}'' = \mathscr{L}' \cup A$.

Similarly to Theorem 4.3, Theorem 4.4 ensures that once a level is blacklisted, the corresponding level is secure again. More precisely, it states that the system is then robust up to $\mathtt{L}_{\mathtt{v}} \setminus \{l\}$ plus the levels $\{l' \mid l' < l\}$ which can still be compromised and the newly lost keys $\mathscr{L}'$, that is, the keys that might have been lost during the execution between E and E'. The proof follows easily from Theorem 4.3.

1.	a	$\rightarrow$	KDC	:	$request$, KDC
2.	KDC	$\rightarrow$	a	:	$\{tgt\}_{K_{TGS}}, \{K_{a,TGS}\}_{K_a}$
3.	a	$\rightarrow$	TGS	:	a,b, $\{tgt\}_{K_{TGS}}, \{Auth1\}_{K_{a,TGS}}$
4.	TGS	$\rightarrow$	a	:	$\{K_{a,b}\}_{K_b}, \{K_{a,b}\}_{K_{a,TGS}}$
5.	a	$\rightarrow$	b	:	$b, \{K_{a,b}\}_{K_b}, \{Auth2\}_{K_{a,b}}$

Figure 4.5: A Kerberos-like protocol

Proof. When the blacklist command is performed, we know, according to the specification, that a is robust up to $(\mathtt{L_v} \cup \bigcup_{k\in\mathscr{L}} \mathsf{Level}(k)) \setminus \{l' \mid l' \leq l\}$ and $\mathscr{L}$ (in the state E_L such that $E \xrightarrow{L} E_L \longrightarrow^* E'$). Then, if t', the current time of global state E' is smaller than t_h, then, since the TRD a is blocked under level l, it is clear that it is robust up to $(\mathtt{L_v} \cup \bigcup_{k\in\mathscr{L}'} \mathsf{Level}(k)) \setminus \{l' \mid l' \leq l\}$ and $\mathscr{L}'$ in E', and, robust up to $(\mathtt{L_v} \cup \bigcup_{k\in\mathscr{L}''} \mathsf{Level}(k)) \setminus \{l\}$ and $\mathscr{L}''$ in E''. (In fact it is even robust to a greater set of levels before the end of the effect of the blacklist.)

If $t' > t_h$, then, according to the hypothesis on t_h, we can use the Theorem 4.3 to prove that E'' itself is robust up to $\mathtt{L}''_{\mathtt{v}} = (\mathtt{L_v} \cup \bigcup_{k\in\mathscr{L}''} \mathsf{Level}(k)) \setminus \{l\}$ and $\mathscr{L}''$, implying immediately that a is robust up to $\mathtt{L}''_{\mathtt{v}}$ and $\mathscr{L}''$ in E''.

□

4.4.5 Application

Suppose we are using our API to implement a protocol similar to Kerberos [NYHR05] for granting access to resources (see Figure 4.5). We consider key distribution centre (KDC), a ticket granting server (TGS) and principals Alice (a) and Bob (b).

We order the levels using an inclusion ordering, where K_a has level (a), $K_{a,TGS}$ has level (a, TGS, KDC) and $K_{a,b}$ has level (a, b, TGS, KDC) (i.e. the levels correspond to the parties assumed to have access to the key). Other terms are considered public (level 0). For two levels l, l', $l > l' \Leftrightarrow l \subset l'$. The protocol is now implementable using the commands of the API.

In a deployment of the system, we assume that devices for the users a, b,... and for the servers TGS, KDC are loaded with appropriate keys of level Max as part of some personalisation process. The administrator can then load long term keys $K_a, K_{a,TGS}$ etc. on to the user devices and the servers. They can now play out the protocol using just the working key commands $\mathsf{encrypt}$, $\mathsf{decrypt}$, $\mathsf{generateSecret}$, $\mathsf{generatePublic}$. Note that when implemented by our API the protocol messages will be tagged following our scheme, e.g. message 2 will appear as $\{tgt, 0, v', \mathsf{TGT}\}_{K_{TGS}}$, $\{K_{a,TGS}, (a, TGS, KDC), v, \mathsf{K_{A,TGS}}\}_{K_a}$ where TGT and $\mathsf{K_{A,TGS}}$ are elements in the miscellaneous field and v, v' are some validity times.

Suppose key $K_{a,TGS}$ is compromised. Then the administrator has two choices. One is to do nothing. Then he knows that after time $v + \Delta((a, TGS, KDC))$, where v is the expiry time of $K_{a,TGS}$, the system returns to a state where keys at level (a, TGS, KDC) are secure. Lower level keys like session keys $K_{a,b}$ might still be compromised, but new session keys will be safe. Alternatively he can signal to TGS and a to blacklist the key. In this case this level (a, TGS, KDC) becomes unusable. However, to preserve functionality, and administrator can add extra tags t_i to all levels, so level (a) becomes (a, t_1), level (a, TGS, KDC) becomes (a, TGS, KDC, t_1, t_2), and the session keys $K_{a,b}$ will have level $(a, b, TGS, KDC, t_1, t_2, t_3)$. If $K_{a,TGS}$ is compromised the administrator only needs to issue a new key with a new level $a, b, TGS, KDC, t_1, t'_2$ and the device can continue to issue tickets securely while level (a, TGS, KDC, t_1, t_2) is blacklisted.

4.5 Implementation in Java Card

In this section, we provide a quick overview of our implementation of this design in Java Card language. The full source code and detailed tutorials are available online at [Wie].

4.5.1 A Few Words about Java Card

Java Card is specially used for embedded devices. It is widely used in SIM and ATM cards. At the language level, Java Card is a subset of Java: all language constructs of Java Card exist in Java and behave identically. However, some common features of Java are not provided at runtime by many actual smart cards (in particular type int, which is the default type of a Java expression; and garbage collection of objects).

4.5.2 The Settings

In this implementation, the first step was to define the keys used by the API, as Javacard objects. To do this, we specified a new class, `APIKey`, which is basically made upon already pre-existing objects `DESKey`, representing keys that are used in a DES encryption scheme. As described in the previous sections, our keys need specific attributes like a level, the value of the key itself, and a validity date. The Figure 4.6 displays all the specification for our `APIKey` attributes. We added more attributes, like an initialization mark or the size.

```
public class APIKey implements DESKey{
        [...]
        // Attributes
        private boolean hasBeenInit; // Is key initialized?
        private short   size;        // Key's size (in bits)
        private byte    type;        // Key's type
        private byte[]  key;         // Key's value
        private byte    level;       // Key's level
        private short   max_clock;   // Key's validity date
        [...]
}
```

Figure 4.6: Specification of the attributes of an objet `APIKey` representing a key.

The Figure 4.7 depicts how the keys are initialized. The method takes as input several arguments. The two first ones are a table of bytes, `v`, and an index `ind`, which points where to start reading `v` to set the value of the key. The method also takes a level `l`, a type `t` and a validity date `timeMax` to set the corresponding attributes of the key. Finally, since the attributes have been set, the key is now marked as initialized.

The whole class `APIKey` has no real specificity of the Java Card language and is just a rather simple object with attributes and specific methods.

4.5.3 The Commands

On the other hand, the implementation of the API itself is rather specific to the Java Card language and uses specific code, which, however, seems to be quite common for API programming.

```
public void init3DES(byte[] v, short ind, byte l, byte t, short timeMax) {
        // Key's value set by content of v starting at index ind.
        this.setKey(v, ind);
        // Setting key's level.
        this.level = l;
        // Setting key's type.
        this.type = t;
        // Setting key's validity date.
        this.max_clock = timeMax;
        // Now key is initialized.
        this.hasBeenInit = true;
}
```

Figure 4.7: Method to initialize one key.

The API is always waiting for a message as input. The message comes with a header in which is specified the command that needs to be performed.

For example, Figure 4.8 displays a part of the code for the generateSecret command described in Section 4.2.2. The part of the code which is missing is a technical part that allows the API to extract all the information of the incoming message (and place it in a buffer).

Note that the generateSecret command is supposed to generate a (secret) key of a level specified by the user. To do so, the API first checks that the level specified by the user is correct. In our case, "correct" means that the level is not equal to 0, which represents public data, and strictly less than `LVL_MAX`, which is a variable representing the level of revocation keys.

After that, the API tests if there is enough memory to store this new key. This is done by the `findFreeKey()` method that either returns an error if the memory is full or an index to a free position. Then, we generate the value of the key. To do so, we need to generate a random seed, which is performed by creating a random generator and using it.

Finally, we can proceed to the initialization of the key by the `init3DES()` method which is given as argument the seed, the expected level (coming from the buffer), the type (also coming from the buffer) and the validity date, which is directly computed by the API, based on the specified level. At the end, the key is stored in the memory and the API returns to the user a message, which contains the handle pointing to this new key. Here, it is just the index of the table `keys[]` where the new entry has been stored.

For all the other commands, we refer the reader to [Wie]. The code may be a little tedious to understand but, hopefully, it is commented enough to understand how they are designed and implemented. The implementation mainly follows the specification of the different API commands we gave in Section 4.2.2.

4.6 Conclusion

We have presented our API for key management with revocation and proved its security properties. While the underlying assumptions for our system to be secure are not unrealistic, in future work we intend to weaken them further. To relax the assumption on level Max messages remaining confidential seems to require more cryptographic primitives. In particular, in order to ensure that the loss of long term keys does not lead to the loss of update messages we require perfect forward secrecy, which would seem to imply the use of asymmetric cryptography. Extending

```
private void generateKey(APDU apdu) {
        [...]
        // We test the expected level
        if (0 >= buffer[ISO7816.OFFSET_CDATA]
                        || buffer[ISO7816.OFFSET_CDATA] >= LVL_MAX) {
                ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
                return;
        }
        // Is it a free place in the memory?
        byte i = findFreeKey();
        // Key Generation
        byte[] alea = new byte[24];
        // Random Generator Creation
        RandomData random = RandomData
                .getInstance(RandomData.ALG_PSEUDO_RANDOM);
        // RG Initialization
        random.setSeed(buffer, (short) 0, (short) 24);
        // Generation of random
        random.generateData(alea, (short) 0, (short) 24);
        // Key's creation using the created random
        keys[i].init3DES(alea, (short) 0,
                // Specifying expected level
                buffer[ISO7816.OFFSET_CDATA],
                // Specifying expected type
                buffer[ISO7816.OFFSET_CDATA + 1],
                // Specifying expected validity date
                getMaxClock(buffer[ISO7816.OFFSET_CDATA]) );
        // The API returns the handle.
        buffer[0] = i;
        apdu.setOutgoingAndSend((short) 0, (short) 1);
}
```

Figure 4.8: Part of the generateSecret command.

the API to asymmetric encryption is the subject of a collaboration with Marion Daubignard. We also assume here that each device has a real time clock. This could be weakened by adding some sort of nonce based freshness test for each new key accepted, as required in the restricted version of the API on which our revocation scheme is based [CS09]. All our security proofs are made in the symbolic model, but it should be possible to extend them to the standard model of cryptography under the usual assumptions, i.e. IND-CCA2 or IND-CPA+INT-CTXT, for the authenticated encryption scheme, since we avoid problematic constructions such as key cycles.

Part III

Type-Systems for Proof-Automation

Chapter 5

Automated Analysis of Ballot-Secrecy through Typing

Contents

As we seen it through Chapter 2, proving equivalence-based properties in the case of complex electronic voting protocols is difficult and potentially error-prone. Automatic tools have been developed to avoid such complicated proofs but as we have described it in the introduction, they remain often insufficient for such protocols. One of the main reason of this is the fact they can not handle very well equational theories including AC-properties. With some exaggeration, if a theory includes an equation of the form $a + b = b + a$, tools come quickly to non-termination. This remains a major issue since several cryptographic primitives used in e-voting may require AC-properties, like homomorphic encryption.

On the other hand, as we stay with quite simple logical properties (associativity and commutativity of the addition), there are existing tools that can help us: SMT Solvers. Indeed, SMT (Satisfiability Modulo Theories) solvers are designed to solve decision problems of first-order logic formulas given specified theories and, even they can been overwhelmed with too complex formulas, e-voting protocols needs should remain simple enough. Such tools have also been used by type-systems to automatically and statically enforce authenticity properties or access control policies of protocols [Eig09].

Previous work already used type-systems to analyze security properties like eligibility or inalterability (no one can change someone's vote) in e-voting protocols (e.g. Civitas [CCM08]). These properties have the common point to have definitions that can be expressed using authorization policies that can be handle by type-systems. A very recent work from Cédric Fournet and *al* [BFG+14] present a new type-system, RF*, that can handle properties like unlikability, integrity or privacy, using a relational approach.

This chapter presents the way we build upon the work of Fournet et *al.* to provide a method to semi-automatically analyze ballot-secrecy for e-voting protocols. This approach is mainly illustrated by the example of Helios, an open-source, web-based voting system [Adi08]. In the first section, we describes the essentials of RF* (we refer the reader directly to [BFG+14] for further details). The Section 5.2 is an informal presentation of the voting protocol Helios. Then, we describe our method through the implementation of Helios in RF* through Section 5.3.

5.1 The RF* Type-System

First presented in [BFG+14], RF* is a relational extension of F*, which is a a typed, strict, higher-order programming language. Based on a relational Hoare logic for a higher-order, stateful, probabilistic language, it aims to allow automatic verification of relational properties (like unlinkability, integrity and privacy) for symbolic or computational implementations.

In this section, we present λ_p, which is a fraction of RF*, a minimal high-order language with statically allocated references and unbounded recursion. For a complete description, we refer to [BFG+14].

5.1.1 Syntax

The sets of types, contexts, values and expressions are given by the following grammars:

Type	$\mathbf{T}$	::=	$\mathbf{B} \mid \mathbf{T} \to \mathbf{T}$
Context	G	::=	$[] \mid \mathrm{G}, [x : \mathbf{T}]$
Value	v, u	::=	$c \mid x \mid o(v_1, \ldots, v_n) \mid \mathsf{fun}\ x : \mathbf{T} \to e$
Expression	e	::=	$v \mid e\ v \mid !r \mid r := v \mid \mathsf{flip} \mid \mathsf{pick}_i^j \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mid \mathsf{let}\ f x = e_1\ \mathsf{in}\ e_2$ $\mid \mathsf{if}\ v\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$

where x is a variables, r a reference and o a $\mathbf{B}$-sorted operator of the form $\mathbf{B}_1 \times \cdots \times \mathbf{B}_n \to \mathbf{B}_0$. The set of types $\mathbf{B}$ is supposed to contain the **unit**, **bool** (booleans) and **int** (integers) types. A reference r with base type $\mathbf{B}$ is denoted by $r : \mathsf{ref}\ \mathbf{B}$. A reference can be seen as a handle: it is possible to store content in it and you can get it back by de-referencement.

5.1.2 Typing

In addition to basic types, we can add refinements to detail the specified type. In RF*, we can specify relational refinement types. These refinements allow to talk about the two projections

$$[\textsc{App}]\ \frac{\mathcal{G} \vdash e_0 \sim e_1 : \mathcal{T} \to \mathcal{C} \qquad \mathcal{G} \vdash v_0 \sim v_1 : \mathcal{T}}{\mathcal{G} \vdash e_0\ v_0 \sim e_1\ v_1 : \mathcal{C}}$$

$$[\textsc{Let}]\ \frac{\mathcal{G} \vdash e_0 \sim e_1 : \{\Phi\}x : \mathcal{T}\{\Xi\} \qquad \mathcal{G}, x : \mathcal{T} \vdash e_0' \sim e_1' : \{\Xi\}y : \mathcal{U}\{\Psi\} \qquad x \notin \mathrm{FV}(\mathcal{U}, \Psi)}{\mathcal{G} \vdash \mathsf{let}\ x = e_0\ \mathsf{in}\ e_0' \sim \mathsf{let}\ x = e_1\ \mathsf{in}\ e_1' : \{\Phi\}y : \mathcal{U}\{\Psi\}}$$

Figure 5.1: Example of relational typing rules

(left and right) of a relational variable or reference. For example, we can specify the following:

$$x : \mathbf{B}\ \{x_{\mathcal{L}} = \mathsf{true} \wedge x_{\mathcal{R}} = \mathsf{false}\}.$$

This type says that value x is of type boolean and such that the left projection of x is equal to true and the right projection of x is equal to false.

Refinement types are either *relational types* (denoted by $\mathcal{T}, \mathcal{U}, \mathcal{G}$), which are used for relational typing of values; or *computation types* (denoted by $\mathcal{C}$), used for relational typing of expression. They are defined by the following grammar:

$$\mathcal{T}, \mathcal{U}, \mathcal{V} := \mathbf{bool} \mid (x : \mathcal{T}) \to \mathcal{C} \qquad \mathcal{C} := \{\Phi\}x : \mathcal{T}\{\Psi\}$$

where Φ and Ψ are relational assertions. A *relational context* $\mathcal{G}$ is a sequence of bindings $x : \mathcal{T}$ such that the bound variables are pairwise distinct.

[BFG+14] proposes a rich set rules that define the relational typing judgements for values ($\mathcal{G} \vdash v_1 \sim v_2 : \mathcal{T}$) and for expressions ($\mathcal{G} \vdash e_1 \sim\ e_2 : \mathcal{C}$). Two excerpts of relational ones are displayed on Figure 5.1.

5.1.3 Soundness

The following theorem states that all judgments of the logic are sound, w.r.t. their interpretations; it implies that typing can be used to verify probabilistic claims, thanks to the properties of lifting.

Theorem 5.1 (Soundness [BFG+14])**.**

- *If* $\mathcal{G} \vdash v_1 \sim v_2 : \mathcal{T}$*, then* $\mathcal{G} \models v_1 \sim v_2 : \mathcal{T}$*,*
- *If* $\mathcal{G} \vdash e_1 \sim e_2 : \mathcal{C}$*, then* $\mathcal{G} \models e_1 \sim e_2 : \mathcal{C}$*.*

We can explain this theorem that way: if two values v_1, v_2 (or expressions e_1, e_2) type-check in RF⋆ (that is $\mathcal{G} \vdash v_1 \sim v_2 : \mathcal{T}$, or $\mathcal{G} \vdash e_1 \sim e_2 : \mathcal{C}$), then, we can consider v_1 and v_2 (or e_1 and e_2) as observationally equivalent.

This allows us to apply RF⋆ to formal verification of e-voting protocols. In fact, to ensure ballot secrecy, we have to ensure that the two frames, which are all the messages seen by the attacker, obtained in the two executions, remain observationally equivalent. To do so, we will ensure, in RF⋆, that all the messages sent over the network are indistinguishable for the attacker in the two executions.

5.2 Presentation of Helios

Helios [Adi08] is the first web-based, open-audit e-voting system, closely related to Benaloh's simple verifiable voting protocol [Ben06]. Compared to the Norwegian design we studied in Chapter 2, Helios only involves an electronic Ballot box (B). Each voter (V) logs, using a computer and a web-browser, in order to submit his vote. Channels between voters and the Ballot box are assumed to be authenticated channels (obtained through password-based authentication). The protocol is mainly divided into two main phases: the submission phase, where voters are allowed to submit their ballots, and the counting phase, where the trusted authorities provide a verifiable tally of the submitted votes. Note that there exist several versions of Helios, based on the use of different cryptographic primitives. In this chapter, we consider Helios with homomorphic encryption.

5.2.1 Settings: Creation of an Election

Before the election, the only main step is the creation of the keys that will be used for encryption and decryption during the election. This is performed by the trustees (e.g. election authorities), who generate a new El-Gamal pair of keys (public and private). Only the public key is displayed for everyone (voters and administrator) and the private key is separated into shares that are given to trustees.

5.2.2 Submission Phase

During the voting phase, each voter uses the public key to perform an encryption of his vote. The encryption used in Helios is a randomized and homomorphic encryption. Moreover, each voter generates a zero-knowledge proof, which ensures that the ciphertext is a valid encryption of a valid vote option. Then, the voter sends his ballot (the ciphertext and the zero-knowledge proof) to the Ballot box. The latter checks the proof and, if the test succeeds, publishes the encrypted ballot on the webpage of the election next to the voter's id. Finally, the voter checks if his ballot has been correctly counted by the Ballot box.

5.2.3 Counting Phase

Once the voting phase is closed, the tally can be computed. The different trustees then check all the proofs. Once they are done, they compute the product of all ciphertexts and share their partial decryption of the final tally using their part of the secret key. They also provide a proof showing that they did a good job. The partial decryptions are put together to fully decrypt the tally and the final result is published, coming with a proof of correct decryption. In our example, we suppose that the Ballot box performs the decryption all by itself, for simplicity.

5.3 Modeling Helios in RF*

This section provides a specification of Helios in RF*, using the framework of Fournet et *al.* [BFG+14], partially presented in Section 5.1. We proceed top to bottom, starting with easiest things and moving slowly into details of the model and the cryptographic interface. All the different files of source code, partially presented here, are available at [Wie].

5.3.1 The Protocol

The whole code for RF* modeling of Helios voting system is represented in Figure 5.2. As we did it for the Norwegian voting protocol, one would have expected that the different players act concurrently. But RF* does not currently provide parallelism and we used cascaded function to model the overall progress of the protocol. Therefore, we are forced to specify an order between players. Here, Alice and Bob (in that order) vote first before letting the attacker submitting his vote. Then, the Ballot box is performing the tally and published the result.

Note. Enforcing the order between participants is not a real limitation. Indeed, in Chapter 2, we also considered that honest voters cast their ballots first in order to provide more information to the attacker. Moreover, we are not limiting the possibilities of the latter, since he can still send whatever he wants to the Ballot box.

```
(* -------------- Helios Process ------------- *)
val Helios: error -> in_Helios_t -> out_Helios_t

let Helios e arg1 =
        // Alice votes and outputs her ciphertext cA.
        let cA = Alice arg1 in

        let Helios_P2 arg2 =
                // Bob votes and outputs his ciphertext cB.
                let cB = Bob arg2 in

                let Helios_P3 cO =
                        // The attacker submits his vote cO
                        // and Ballot Box makes the tally.
                        let res = BallotBox cA cB cO e in
                                res in

                (cB,Helios_P3) in
        (cA,Helios_P2)
```

Figure 5.2: RF* code of Helios Protocol

In this code, Helios is specified to take two inputs, an object of type `error` (`e`) and another of type `in_Helios_t` (`arg1`). The first one is just a type `eq bytes` which corresponds to an error message from which the attacker is learning nothing and will be thrown away by the Ballot box if something goes wrong during its process (see Figure 5.4). The second one is also an `eq bytes` type and is just a dummy argument. The protocol is expected to output something of type `out_Helios_t` which is of type `eq bytes` since this refers to the result of the election and the attacker should see no differences between the results of the two execution of the process.

5.3.2 Honest Voters: Alice (and Bob)

In this section, we have a closer look to the behavior of the two honest voters: Alice and Bob. Since they behave in very similar ways, we only present the model of Alice. (For the model of Bob, we refer to the full source code, available at [Wie].)

Let us start with the implementation of Alice, which represents the operations Alice performs during the voting process. Once again, the function is provided a dummy argument `arg` which has no influence over its execution. As a first step, `v1` and `v2`, which are integers values, are transformed (for typing reasons) into bytes using function `Int2Bytes`. Then, `vA`, the vote casted by Alice, is chosen between `v1Byt` and `v2Byt`, meaning that, in the first execution of the process, we have `vA` = `v1Byt` and, in the second one, `vA` = `v2Byt`. This step allows us to model the necessary swapping of vote between the two honest voters. Once the choice is made, Alice can encrypt her vote thanks to an encryption function `encA` (defined a little bit further) and outputs her encrypted ballot.

```
(* -------------- Alice Interface -------------- *)
val Alice: eq bytes -> ca:cipher{| (FromAboth (L ca) (R ca))
                                && ( exists (mLa:bytes) (mRa:bytes).(
                                        (Encryptedboth mLa mRa (L ca) (R ca))
                                        && (Marshboth (L v1) (R v2) mLa mRa) )
                                ) |}

(* -------------- Alice Implementation -------------- *)
let Alice arg =
        let v1Byt = Int2Bytes v1 in
        let v2Byt = Int2Bytes v2 in
        let va = choice v1Byt v2Byt in
        let ca = encA va in
        ca
```

Figure 5.3: RF* code of Alice.

The interface needs a bit more explanation on its own. As we said it, Alice takes as input something irrelevant of type `eq bytes` and she outputs a value `cA` which is of type `cipher` (which corresponds to `eq bytes` but has been called this way to ease the understanding of the manipulated objects) with some refinement. The refinement says two things:

- `FromABoth (L cA) (R cA)` is a predicate that ensure that both `L cA` and `R cA`, which respectively corresponds to the value of `cA` in the first execution and in the second execution, are identified as actually coming from Alice.
- The second part assumes it exists two bytes values `mLa` (for the first execution) and `mRa` (for the second execution) such that `L ca` is an encryption of `mLa`, in the first execution, and `R ca` is an encryption of `mRa` in the second execution. This is coded through the predicate `EncryptedBoth`. Moreover, we should have that `mLa` is a byte corresponding to the integer `v1` and `mRa` to `v2`, which is written using the predicate `Marshboth`.

In a nutshell, `cA` is a ciphertext, indistinguishable from one execution to another, which is encrypting `v1` in the first execution and `v2` in the second one.

5.3.3 The Ballot Box

Now, we can move on the code for the Ballot box's implementation. The figure below describes the set of operations done it is called by the Helios process. The Ballot box takes, as inputs,

three ciphertexts coming respectively from Alice, Bob and the attacker. It also takes the error `e` (which has type `error`, i.e. `eq bytes`) that is thrown away if a problem is encountered.

The first three steps model the verification of the zero-knowledge proofs coming along with the ciphertexts. In this model, we choose to not represent directly zero-knowledge proofs (see Section 5.4 for more details) but we can still check them using the `check_validity` function over a given ciphertext. If the three tests succeed, then the Ballot box proceeds to two more checks. Indeed, in this version of Helios it has been proven [CS13] that we need to remove duplicate ballots to avoid a replay attack on ballot-secrecy.

```
(* -------------- Ballot Box Implementation -------------- *)
let BallotBox ca cb co e =
        let ba = check_validity ca in
        match ba with
        | false -> e
        | true ->
        let bb = check_validity cb in
        match bb with
        | false -> e
        | true ->
        let bo = check_validity co in
        match bo with
        | false -> e
        | true ->
        if (co = ca) then e else
        if (co = cb) then e else
        let cab = homomorphic ca cb in
        let cfinal = homomorphic cab co in
        let mfinal = decrypt cfinal in
        mfinal
```

Figure 5.4: RF* code for the Ballot box's implementation.

Once all the checks have been performed, the Ballot box can prepare the tally and, thus, puts all the different ciphertexts together using the homomorphic property of encryption. This is modeled by the `homomorphic` function, which takes two ciphertexts and combine them in one. Finally, the Ballot box decrypts the final tally, `cfinal`, and outputs the result, `mfinal`.

Concerning the interface of the Ballot box, we can see that it takes four inputs. The two first inputs are ciphertexts coming from Alice and Bob with the specified types coming from Alice (and Bob) interfaces. It also takes a ciphertext `co` from the attacker, which has type `eq bytes` because the same constructions must be sent in both executions. It also takes an input of type `error` and finally should output something of type `eq bytes`.

5.4 Specifying the Cryptographic Interface

In this section, we describe the RF* code that represents the core of our model: the cryptographic library. We detail here all the functions we have seen in the previous parts like the encryption function, the homomorphic function, etc.

```
(* -------------- Ballot Box Interface -------------- *)
val BallotBox: ca:cipher{| (FromAboth (L ca) (R ca))
                              && ( exists (mLa:bytes) (mRa:bytes).(
                                      (Encryptedboth mLa mRa (L ca) (R ca))
                                      && (Marshboth (L v1) (R v2) mLa mRa) )
                              ) |}
                        -> cb:cipher{| (FromBboth (L cb) (R cb))
                              && ( exists (mLb:bytes) (mRb:bytes).(
                                      (Encryptedboth mLb mRb (L cb) (R cb))
                                      && (Marshboth (L v2) (R v1) mLb mRb) )
                              ) |}
                        -> co:cipher
                        -> error
                        -> result
```

Figure 5.5: RF* code for the Ballot box's interface.

5.4.1 Encryption Function

To model encryption, we used one of the classical approaches performed in type-systems: seal-based libraries. A seal can be encoded in several ways, but, usually, it is a pair that consists of a sealing function and an unsealing function. The sealing function takes a plaintext m as an input. If the sealing function has never been called with this argument before, it generates a fresh value c and adds the pair (m, c) to a secret table, before returning c; if an entry containing m already exists, it just returns c. The unsealing function takes c and returns m if an entry is found in the same table. Without the unsealing function m can not be recovered since the table is only accessible using these specific sealing and unsealing functions.

To fully model homomorphic encryption, we build two tables (so two seals): T_{valid} and T_{all}. The first table, T_{valid}, reflects the ciphertexts that are "valid", i.e. which come with a zero-knowledge proof that will be implicitly present. For example, the encrypted ballots generated by the honest voters appears in that table. The second table, T_{all} is used for the homomorphic property as described in Section 5.4.3.

```
(* Seal reference for T(able) Valid *)
type seal_val_ref_entry_t =
        m:eq bytes * c:cipher{| Encryptedboth (L m) (R m) (L c) (R c)
                && ( Validboth (L c) (R c) )
                && ( (FromAboth (L c) (R c)) || (FromBboth (L c) (R c))
                        || ( (FromOboth (L c) (R c)) && (L m = R m) ) )
                && ( exists (zL:int) (zR:int).(Marshboth zL zR (L m) (R m)) )
                |}
```

Figure 5.6: RF* code for content of table T_{valid}.

The seal corresponding to the table T_{valid} is depicted in Figure 5.6. It contains pairs of plaintexts and corresponding ciphertexts. An entry in the table can be seen as follows:

```
((L m, R m), (L c, R c))
```

where we have the left and right values of the plaintext m and the ciphertext c. We also have a refinement on the ciphertext c. The predicate `EncryptedBoth`, implying that `L c` is the encryption of `L m` and the for `R c` with `R m`. The predicate `ValidBoth` represents the implicit zero-knowledge proof coming along with the ciphertext. Moreover, a ciphertext is tagged to come either from Alice, Bob or the attacker, which is represented by predicates `FromAboth`, `FromBboth` and `FromOboth`. We also include that the plaintexts should corresponds to some integers. (This predicate is needed since voting option are originally `int` (integers) but are sent over the network as `bytes`.)

```
(* Encryption for Alice *)
val encA: m:bytes{| (Marsh (L v1) (L m)) && (Marsh (R v2) (R m)) |}
                -> c:cipher{| (Encryptedboth (L m) (R m) (L c) (R c))
                        && (FromAboth (L c) (R c)) |}

(* Encryption for the Attacker *)
val encO: m:bytes{| L m = R m |} -> cipher
```

Figure 5.7: Encryption functions for Alice and the Attacker.

Then, we can write the interfaces for the encryption function. We designed three encryptions functions: one for Alice, one for Bob and one for the attacker. They represent the same encryption but we needed the separation because of the predicates `FromXboth` which are specific for each voter. The Figure 5.7 describes the encryption functions: `encA` of Alice (Bob has a very similar function `encB`) and `encO` for the attacker. The latter is very simple, it takes an `eq bytes` as a plaintext and return an `eq bytes` as a ciphertext.

For Alice, it is a bit more complicated. The function `encA` takes as an input a plaintext m, which is of type `bytes` and such that it should correspond to the integer `v1` in the first execution and `v2` in the second execution. The function returns a ciphertext of type `eq bytes` (`cipher`) such that the ciphertext is indeed an encryption of m (predicate `EncryptedBoth`). Moreover, the ciphertext is tagged as coming from Alice (predicate `FromAboth`).

5.4.2 Zero-Knowledge Proofs

```
(* Unseal function type for T(able) valid (used for validity checks) *)
val check_validity : c:cipher
-> b:bool{| (( (L b) = true ) && ( (R b) = true ))  => (
        (Validboth (L c) (R c))
        && (exists (m1:bytes) (m2:bytes) (z1:int) (z2:int).(
                (Encryptedboth m1 m2 (L c) (R c)) && (Marshboth z1 z2 m1 m2)
                && ( ( (FromAboth (L c) (R c)) )
                        || ( (FromBboth (L c) (R c)) )
                        || ( (FromOboth (L c) (R c)) && (m1 = m2) ) )
                ))
        )|}
```

Figure 5.8: Function check_validity used to verify implicit zero-knowledge proofs.

As we have seen in the previous section, zero-knowledge proofs are implicitly given using the

table T_{valid}. To check such proof, we define `check_validity` (see Figure 5.8) which is acting, in a way, like an unsealing function.

This function takes as input a ciphertext c and returns a boolean b, which is specified as follows. If, in the two executions, b is equal to `true`, then, it implies that the predicate `Validboth` holds for left and right value of c. Moreover, we know that it exists a plaintext m (with left and right values) such that c is an encryption of m. We also know that this m is a byte-interpretation of some integers. Finally, the ciphertext c is coming either from Alice, Bob or the attacker. In the case of the latter, we additionally knows that m must be an `eq bytes` and, thus, the left and right values are equal.

5.4.3 Homomorphic Property

Here comes maybe the more tricky part of this model: the `homomorphic` function that allows to combine two ciphertexts in one. The goal is to reflect the following simple equation that models homomorphic encryption:

$$\mathsf{enc}(x,k) * \mathsf{enc}(y,k) = \mathsf{enc}(x+y,k).$$

Adding two ciphertexts leads to another one, which contains the sum of the two plaintexts. Morally, this is what is coded in Figure 5.9.

```
(* Reseal function type for homomorphic encryption *)
val homomorphic : c1:cipher -> c2:cipher -> c:cipher{|
        forall (mL1:bytes) (mL2:bytes) (mR1:bytes) (mR2:bytes)
        (zL1:int) (zL2:int) (zR1:int) (zR2:int).(
                ( (Encryptedboth mL1 mR1 (L c1) (R c1))
                && (Encryptedboth mL2 mR2 (L c2) (R c2))
                && (Marshboth zL1 zR1 mL1 mR1)
                && (Marshboth zL2 zR2 mL2 mR2) )
                => (exists (mL:bytes) (mR:bytes).(
                        (Encryptedboth mL mR (L c) (R c))
                        && (Marshboth (zL1 + zL2) (zR1 + zR2) mL mR)
                ))
        )|}
```

Figure 5.9: Homomorphic function to combine ciphertexts.

The `homomorphic` function takes as input two ciphertexts `c1` and `c2` in order to output a ciphertext c. We know that for all two plaintexts `m1` and `m2` corresponding to the two ciphertexts such that these plaintexts are byte-version of existing integers `z1` and `z2`, then c is an encryption of a plaintext m such that m is a byte corresponding to an integer z verifying the fact that $z = \mathtt{z1} + \mathtt{z2}$.

All the ciphertexts created using the `homomorphic` function are stored into the table T_{all}. Indeed, this function is acting like a sealing function: for a given pair `c1` and `c2`, it generates (or outputs, if the entry already exists) a fresh random c, stores the corresponding entry in the table and outputs c.

5.4.4 Decryption Function

To decrypt the final tally, we implement the unsealing function for table T_{all}. The code is rather simple compared to previous examples. The `decrypt` function takes a cipher as an input, looks into T_{all} if there is a corresponding entry and outputs a plaintext m of type `bytes` such that m is the plaintext corresponding to c.

```
(* Unseal function type for T(able) all (will be used for decryption) *)
val decrypt : c:cipher
        -> m:bytes{| forall (z1:bytes) (z2:bytes). (
                (Encryptedboth z1 z2 (L c) (R c)) => ((z1 = L m) && (z2 = R m))
        ) |}
```

Figure 5.10: Decryption function.

5.4.5 Assumptions

Finally, to ensure ballot-secrecy in this Helios implementation (more precisely, to help the SMT solver to conclude), we needed to encode some properties that are verified during the protocol execution. We first assume that, for each ciphertext, there is one and only one corresponding plaintext. In other words, we are using deterministic decryption schemes. Another assumption is that Alice and Bob (honest voters) only vote once.

```
(* A single ciphertext only corresponds to one plaintext *)
assume forall
(mL1:bytes) (mL2:bytes) (mR1:bytes) (mR2:bytes) (c1:bytes) (c2:bytes).(
        ((Encryptedboth mL1 mR1 c1 c2)  && (Encryptedboth mL2 mR2 c1 c2))
                => ((mL1 = mL2) && (mR1 = mR2))    )

(* Voter A only vote once. *)
assume forall (c1:bytes) (c2:bytes).( ((FromA c1) && (FromA c2)) => (c1 = c2) )

(* Voter B only vote once. *)
assume forall (c1:bytes) (c2:bytes).( ((FromB c1) && (FromB c2)) => (c1 = c2) )
```

Figure 5.11: Assumptions translated into RF* code.

5.5 Results & Conclusion

This model of Helios made it through the RF* type-checker and, therefore, ensures that this implementation of the protocol satisfies ballot-secrecy according to what we showed in Section 5.1.

5.5.1 Future Work

Our work was a first step in a new approach considering type-systems for automatic verification of security properties for e-voting protocol. Unfortunately, RF*, quite new, is still difficult to use and we didn't succeed in implementing other protocols like the Norwegian one, discussed in

Chapter 2. Nevertheless, this is still encouraging since it is the starting point for providing a a general purpose library for type-based verification of e-voting protocols in collaboration with M. Maffei, F. Eigner and S. Kremer. The goal is to formalize different security properties like ballot-secrecy, individual and universal verifiability, eligibility, etc. in terms of type-systems. As we mentioned it in introduction of this chapter, some of them have already been studied, and this work aims to provide a methodology and some tools to use type-systems to (partially) automatically verify e-voting protocols. Since all properties are not based on equivalences, we would not need RF* for all of them and, typically, it would only be required for proving ballot-secrecy, receipt-freeness or coercion-resistance.

Other possible lines of work is to use the existing tool EasyCrypt [BGHB11], which, even if it is mainly used for computational verification, also deals with symbolic proofs and may be less "restrictive" than RF* (but the former has several years of existence, while the latter is brand new).

Conclusion & Perspectives

"There is no real ending.
It's just the place where you stop the story."

Frank Herbert

This thesis focuses on two advanced families of security protocols: electronic voting and APIs. In particular, we analyze existing and used e-voting protocols, and provide a new design of a secure API for symmetric keys management including revocation functionality. Although further works on these results have already been mentioned in the conclusion of the previous chapters, we recall here part of them, and present other possible leads.

E-Voting Protocols Analysis

The first part of this thesis presents two analyses of existing e-voting systems. Thanks to the applied pi-calculus model [AF01], we are able to provide a formal and detailed study of these protocols, showing their security in different corruption cases. For both of them, these formal analyses were, at our knowledge, the first formal proofs showing that these e-voting systems indeed satisfy ballot secrecy, and correctness in the case of the CNRS protocol.

Nevertheless, the study of the Norwegian system shows the limits of a hand-made proof. Even if we try to provide global tools (general lemmas) that can be reused in further analyses of different voting protocols using a similar scheme, the complexity of the Norwegian scheme represents, in terms of proofs, a long and error-prone work. Current tools can not be used when it comes to cryptographic primitives with AC-properties, especially when, like in the Norwegian, they are used together with a whole set of other cryptographic functions.

As we have already discussed it, a further step consists in developing a procedure for automatically checking for equivalences, which remains a difficult problem. New tools also appear, improving the state of art of automatic verification like Tamarin [MSCB13] and APTE [Che14]. But they seem to not support complex theories like the one used in the Norwegian protocol. An alternative step would be to develop a sound procedure that over-approximates the relation, losing completeness in the spirit of ProVerif [Bla05] but tailored to e-voting properties.

Instead of a full verification approach, one other lead would be to only assist automatically a part of the proof. In the Norwegian and CNRS protocol, we have seen that the exhibition and proof of the existence of a relation is somewhat cumbersome and not very difficult. A first-step could then be to be able to automatize this part. Given an applied pi-calculus specification of a protocol, it could be possible to be given, automatically, a relation and a proof, modulo equivalence results that would be discharged to a human. That would not solve the main issue since the real difficulty is in that part, but would still be helpful.

In Part III, we have shown that it might be possible to verify ballot secrecy using type-systems. We have also seen that there are currently some limitations but, in collaboration with

M. Maffei, F. Eigner and S. Kremer, we are aiming at providing a formal framework that could use type-systems in general (not only RF* as we did it for ballot secrecy) to ensure different security properties like verifiability, eligibility or correctness. Our goal is to provide new definitions of these security properties, based on observational equivalence as well as authorization policies, assumptions and assertions. Then, these definitions could be used to model an e-voting protocol, following an existing type-systems syntax (supporting the definition we try to prove), and the associated type-checker will perform the analysis automatically for us.

APIs Analysis

The Part II presents a new design for managing symmetric keys including a revocation functionality. It also seems that it is one of the first models coming along with a full formal security proof specifying the different security guarantees it provides.

One of the logical perspectives of future work is to extend this design to asymmetric keys, like the extension of the design presented by V. Cortier and G. Steel in [CS09] has been extended in [DLS14]. This is an ongoing project in collaboration with M. Daubignard. The major difficulty is to deal with asymmetric keys revocation. In practice, this is done using Certificate Revocation Lists (CRLs) [Aka13], a standard that is somewhat tricky to implement in APIs, especially because we need to decide, who issues the certificate for one specific key. The main issue is more a matter of design than a matter of effective security as we strongly believe that extending [CSW12] to asymmetric keys (or [DLS14] to revocation) should follow the same proof technique. Nevertheless, at this point, having an automatic prover would be welcome.

Another interesting further perspective would be, as with security protocols in general, to provide an automatic formal analysis of API designs. A current tool, Tookan [BCFS10], is already able to analyze implementations of these interfaces but it would be still interesting to provide an automatic verification at an earlier stage of development ensuring that the designs are worth the implementation. An extension of ProVerif, StatVerif [ARR11], has been developed to deal with protocols using global states, like APIs. One lead would be to see if it is possible to use this new tool to formalize and verify the property we proved in Part II.

Security APIs & E-Voting ?

Finally, the two families of protocols presented in this thesis are mainly separated as if there is no interactions between them. One of the main argument against electronic voting (especially protocols designed to vote using Internet) is that their security often relies on the fact that the voter's computer must remain safe. As far as we know, it is quite difficult to achieve this in reality because of all the different threats (malwares, viruses, fishing, etc.) that exist online.

On the other hand, manipulating secret data on untrusted machines is the goal of security APIs. Thus, one further perspective would be to see if there is any possibility to combine electronic voting with APIs to ensure security properties under weaker assumptions. Indeed, using these interfaces, we could be able to remove the fact that the voter's computer should remain free of malicious programs that could leak information to the attacker. The idea would be to use a secure token to perform the authentication and encryption on the voter's behalf.

Of course, we would need to specify the design of such global protocol that would combine two specific ones and proving that it satisfies security properties (e.g. ballot secrecy) is not guaranteed to be easy. There is still a chance that such design could be analyzed by parts and reach a general result using composability.

Part IV

Appendix

Appendix A

Norwegian E-Voting Protocol: Lemmas for Static Equivalence

We recall in this chapter all the different results that have been proved useful for the main proofs of propositions and theorems detailed in Chapter 2.

A.1 Notations

Definition A.1. *Let us define and remind the following notations:*

$$
\begin{aligned}
\tilde{\omega} =& \mathsf{a}_1, \mathsf{a}_3, \mathsf{id}_1, \mathsf{id}_2, \mathsf{t}_1, \mathsf{t}_2, \mathsf{id}_\mathsf{R},\\
\theta_\delta^b =& \{^{\mathsf{vk}(\mathsf{id}_\mathsf{k})}/_{\mathsf{idp}_\mathsf{k}}, ^{\mathsf{s}(\mathsf{id}_\mathsf{k})}/_{\mathsf{s}_\mathsf{k}} \mid \mathsf{k} = 1..n\} \mid \{^{\mathsf{vk}(\mathsf{id}_\mathsf{R})}/_{\mathsf{idp}_\mathsf{R}}\} \mid \{^{\mathsf{pk}(\mathsf{a}_\mathsf{k})}/_{\mathsf{g}_\mathsf{k}} \mid \mathsf{k} = 1..3\} \mid\\
& \{^{\mathsf{ball}_\mathsf{k}}/_{ba_k}, ^{\mathsf{ball}_\mathsf{k}}/_{ba'_k}, ^{\mathsf{ok}}/_{conf_k} \mid \mathsf{k} = 1..2\} \mid\\
& \{^{\mathsf{ok}}/_{go_r^k}, ^{\mathsf{ok}}/_{sy_k}, ^{\mathsf{d}(\mathsf{p}(\mathsf{id}_\mathsf{k}),\mathsf{dec}(\Pi_2(x_k),\mathsf{a}_3))}/_{x_r^k} \mid \mathsf{k} = 3..n\} \mid\\
& \{^{\mathsf{sign}(\mathsf{hash}(\Pi_1(x_k)),\mathsf{id}_\mathsf{R})}/_{sr_k}, ^{\mathsf{d}(\mathsf{p}(\mathsf{id}_\mathsf{k}),\mathsf{dec}(\Pi_2(x_k),\mathsf{a}_3))}/_{rec_k} \mid \mathsf{k} = 1..n\} \mid\\
& \{^{\langle \mathsf{idp}_\mathsf{k}, \mathsf{hash}(\Pi_1(x_k))\rangle}/_{hr_k}, ^{\mathsf{dec}(d_{\delta(k)},\mathsf{a}_1)}/_{res_k} \mid \mathsf{k} = 1..n\},\\
\theta_\delta =& \{^{\mathsf{vk}(\mathsf{id}_\mathsf{k})}/_{\mathsf{idp}_\mathsf{k}}, ^{\mathsf{s}(\mathsf{id}_\mathsf{k})}/_{\mathsf{s}_\mathsf{k}} \mid \mathsf{k} = 1..n\} \mid \{^{\mathsf{vk}(\mathsf{id}_\mathsf{R})}/_{\mathsf{idp}_\mathsf{R}}\} \mid \{^{\mathsf{pk}(\mathsf{a}_\mathsf{k})}/_{\mathsf{g}_\mathsf{k}} \mid \mathsf{k} = 1..3\} \mid\\
& \{^{\mathsf{penc}(\mathsf{v}_\mathsf{k},\mathsf{t}_\mathsf{k},\mathsf{g}_1)}/_{\mathsf{e}_\mathsf{k}}, ^{\mathsf{pfk}_1(\mathsf{id}_\mathsf{k},\mathsf{t}_\mathsf{k},\mathsf{v}_\mathsf{k},\mathsf{e}_\mathsf{k})}/_{\mathsf{p}_\mathsf{k}}, ^{\mathsf{sign}(\langle \mathsf{e}_\mathsf{k},\mathsf{p}_\mathsf{k}\rangle,\mathsf{id}_\mathsf{k})}/_{\mathsf{si}_\mathsf{k}} \mid \mathsf{k} = 1..2\} \mid\\
& \{^{\mathsf{sign}(\mathsf{hash}(\Pi_1(x_k)),\mathsf{id}_\mathsf{R})}/_{sr_k}, ^{\mathsf{d}(\mathsf{p}(\mathsf{id}_\mathsf{k}),\mathsf{dec}(\Pi_2(x_k),\mathsf{a}_3))}/_{rec_k} \mid \mathsf{k} = 1..n\} \mid\\
& \{^{\mathsf{dec}(d_{\delta(k)},\mathsf{a}_1)}/_{res_k} \mid \mathsf{k} = 1..n\},\\
\sigma_\mathsf{L} =& \{^{\mathsf{a}}/_{\mathsf{v}_1}, ^{\mathsf{b}}/_{\mathsf{v}_2}\},\ \sigma_\mathsf{R} = \{^{\mathsf{b}}/_{\mathsf{v}_1}, ^{\mathsf{a}}/_{\mathsf{v}_2}\},
\end{aligned}
$$

$$
\begin{aligned}
\tilde{\sigma} =& \{^{M_\mathsf{k}}/_{x_k}, ^{U_\mathsf{k}}/_{d_k}, ^{W_\mathsf{k}}/_{hb_k} \mid \mathsf{k} = 1..n\} \mid \{^{N_\mathsf{k}}/_{x_b^k} \mid \mathsf{k} = 1..2\},\\
\hat{\sigma} =& \{^{M_\mathsf{k}\alpha}/_{x_k}, ^{U_\mathsf{k}\alpha}/_{d_k}, ^{W_\mathsf{k}\alpha}/_{hb_k} \mid \mathsf{k} = 1..n\} \mid \{^{N_\mathsf{k}\alpha}/_{x_b^k} \mid \mathsf{k} = 1..2\}\ \textit{with}\\
\alpha =& \{^{\langle \mathsf{idp}_\mathsf{k},\mathsf{e}_\mathsf{k},\mathsf{p}_\mathsf{k},\mathsf{si}_\mathsf{k}\rangle}/_{ba_k}, ^{ba_k}/_{ba'_k}, ^{\mathsf{ok}}/_{conf_k} \mid \mathsf{k} = 1..2\} \mid\\
& \{^{\langle \mathsf{idp}_\mathsf{k}, \mathsf{hash}(\Pi_1(x_k))\rangle}/_{hr_k} \mid \mathsf{k} = 1..n\} \mid \{^{\mathsf{ok}}/_{go_r^k}, ^{\mathsf{ok}}/_{sy_k}, ^{rec_k}/_{x_r^k} \mid \mathsf{k} = 3..n\}.
\end{aligned}
$$

A.2 Starting Lemmas

Lemma A.1. *Using the notations of Definition A.1, we have:*

$$\nu\tilde{\omega}.\theta_\delta^b\tilde{\sigma}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{t_\delta}^b\tilde{\sigma}\sigma_R \iff \nu\tilde{\omega}.\theta_\delta\hat{\sigma}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{t_\delta}\hat{\sigma}\sigma_R.$$

Proof. The proof is done using Lemma 2.3 to remove the redundant information contained in x_r^p and ba'_p, Lemma 2.2 to expand ba_1 and ba_2, and, finally, Lemma 2.3 to remove the different ok stored in the frame (go_r^k, sy_k and $conf_k$) that does not influence static equivalence. Using Lemma 2.3, we also can remove hr_k since they are free terms.

□

Lemma A.2. *Using notations of Definition A.1, for* $\mathsf{i} \in \{1,2\}$, $\mathsf{j} \in \{1,2,3\}$ *and any free term* U, *if* M *is of the form* $\mathsf{a_j} + U$, $\mathsf{t_i} * U$, $\mathsf{id_i}$ *or* $\mathsf{p}(\mathsf{id_i})$, *then* $\nu\tilde{\omega}.\theta_\delta \not\vdash M$.

Proof. We define four properties:

1. Let N be a deducible term such that $\exists\, p$ such that $N|_p = \mathsf{a_i} + U$ for $\mathsf{i} \in \{1,3\}$ and any term U. There are two cases:
   - $U = \epsilon$ (with $\mathsf{a_i} + \epsilon = \mathsf{a_i}$) and $p = p'.2$ such that $N|_{p'} = \mathsf{f}(N', \mathsf{a_i})$ with $\mathsf{f} \in \{\mathsf{dec}, \mathsf{renc}\}$.
   - $p = p'.1.q$ such that $N|_{p'} =_{AC} \mathsf{pk}(\mathsf{a_i}+U')$ for some U' and $\forall q' < q$, $\mathsf{head}(N|_{p'.1.q'}) = +$.
2. Let N be a deducible term such that $\exists\, p$ such that $N|_p = \mathsf{t_i} * U$ for $\mathsf{i} \in \{1,2\}$ and any term U. There are two cases :
   - $p = p'.2.q$ such that $N|_{p'} =_{AC} \mathsf{penc}(P_1, \mathsf{t_i} * U, K_1)$ and $\forall q' < q$, $head(N|_{p'.2.q'}) = *$.
   - $p = p'.2.q$ such that $N|_{p'} =_{AC} \mathsf{pfk}_1(P_1, \mathsf{t_i} * U, P_2, P_3)$ and $\forall q' < q$, $head(N|_{p'.2.q'}) = *$.
3. Let N be a deducible term such that $\exists\, p$ such that $N|_p = \mathsf{id_i}$ for $\mathsf{i} \in \{1,2\}$. There are five cases:
   - $p = p'.1$ and $N|_{p'} = \mathsf{vk}(\mathsf{id_i})$.
   - $p = p'.1$ and $N|_{p'} = \mathsf{p}(\mathsf{id_i})$.
   - $p = p'.1$ and $N|_{p'} = \mathsf{s}(\mathsf{id_i} i)$.
   - $p = p'.1$ and $N|_{p'} = \mathsf{pfk}_1(\mathsf{id_i}, P_1, P_2, P_3)$.
   - $p = p'.2$ and $N|_{p'} = \mathsf{sign}(N', \mathsf{id_i})$.
4. Let N be a deducible term such that $\exists\, p$ such that $N|_p = \mathsf{p}(\mathsf{id_i})$ for $\mathsf{i} \in \{1,2\}$. Then, $p = p'.1$ and $N|_{p'} = \mathsf{d}(\mathsf{p}(\mathsf{id_i}), M)$.

We can see that these properties imply the undeducability of $\mathsf{a_i} + U$, $\mathsf{t_i} * U$, $\mathsf{id_i}$ and $\mathsf{p}(\mathsf{id_i})$. Let us prove these properties by induction on the number of steps needed to deduce N.

Base Case: All terms in the frame θ_δ verify these properties.

Induction Hypothesis: Let $M_1, \ldots, M_k$ be terms in normal form, deducible in $i-1$ steps verifying the properties. We prove that $f(M_1, \ldots, M_k)\!\downarrow$ also verifies them. There are two main cases:

- If $f(M_1, \ldots, M_k)$ is in normal form, then it is obvious that the result is true.

- If it is not in normal form, then, since $M_1, \ldots, M_k$ are in normal form, the reduction occurs in head.
  - If the applied rule is different from (E-4), (E-5) and (E-6), then the result is a subterm of $M_1, \ldots, M_k$ (or ok) and it is verifying the properties, using the induction hypothesis.
  - If the rule (E-4) is applied, then $\mathsf{dec}(M_1, M_2) \longrightarrow \mathsf{blind}(T_1, T_2)$ with T_1 and T_2 subterms of M_1, M_2 and are verifying the properties by induction hypothesis. Then, $\mathsf{blind}(T_1, T_2)$ verify the properties.
  - If the rule (E-5) is applied, then $\circ(M_1, M_2) \longrightarrow \mathsf{penc}(T_1 \diamond T_2, R_1 * R_2, P)$ with T_1, T_2, R_1, R_2, P subterms of M_1, M_2, all verifying the properties by induction hypothesis. Thus, the result, in normal form, is verifying the properties too.
  - Finally, if the rule (E-6) is applied, then $\mathsf{renc}(M_1, M_2) \longrightarrow$ $\mathsf{penc}(T, R, \mathsf{pk}(P_1 + P_2))$ with T, R, P_1 and P_2 subterms of M_1 and M_2. Thanks to the induction hypothesis, we can see that the the result is satisfying the property. Indeed, whatever the case where a_i is a subterm of P_1 or P_2, $\exists\, P'$ such that $P_1 + P_2 =_{AC} \mathsf{a_i} + P'$.

□

A.3 Base Case for Static Equivalence

Lemma A.3. *We consider the following frame:*

$$\theta_0 = \{^{\mathsf{vk}(\mathsf{id_k})}/_{\mathsf{idp_k}}, {}^{\mathsf{s}(\mathsf{id_k})}/_{\mathsf{s_k}} \mid \mathsf{k} = 1..n\} \mid \{^{\mathsf{vk}(\mathsf{id_R})}/_{\mathsf{idp_R}}\} \mid \{^{\mathsf{pk}(\mathsf{a_k})}/_{\mathsf{g_k}} \mid \mathsf{k} = 1..3\} \mid$$
$$\{^{\mathsf{penc}(\mathsf{v_k},\mathsf{t_k},\mathsf{g_1})}/_{\mathsf{e_k}}, {}^{\mathsf{pfk_1}(\mathsf{id_k},\mathsf{t_k},\mathsf{v_k},\mathsf{e_k})}/_{\mathsf{p_k}}, {}^{\mathsf{sign}(\langle \mathsf{e_k},\mathsf{p_k}\rangle,\mathsf{id_k})}/_{\mathsf{si_k}} \mid \mathsf{k} = 1..2\}.$$

Then, we have: $\nu\tilde{\omega}.\theta_0\sigma_L \approx_s \nu\tilde{\omega}.\theta_0\sigma_R$.

Proof. Note that the adversary can arbitrarily combine ciphertexts from the frame with ciphertexts in the frame or freshly constructed ciphertexts, thus we enrich the frame θ_0 with any such combination of ciphertexts. Formally, for any $\alpha_1, \alpha_2 \in \mathbb{N}$ and free terms K, P, R, we define $C_{\alpha_1,\alpha_2,K,P,R}$ as follows :

$$\mathsf{penc}(P \overset{2}{\underset{i=1}{\diamond}} v_i^{\alpha_i}, R \overset{2}{\underset{i=1}{*}} r_i^{\alpha_i}, \mathsf{pk}(\mathsf{a_1} + K)).$$

We define the extended frame $\widehat{\varphi}$ as follows:

$$\begin{aligned}\widehat{\varphi} = \nu\tilde{\omega}.(\theta_0 | \{^{C_{\alpha_1,\alpha_2,K,P,R}}/_{x_{\alpha_1,\alpha_2,K,P,R}} \mid \alpha_1, \alpha_2 \in \mathbb{N},\ \alpha_1 + \alpha_2 \neq 1 \\ \text{and terms } K, P, R \text{ such that } (\mathsf{fn}(K) \cup \mathsf{fn}(P) \cup \mathsf{fn}(R)) \cap \tilde{\omega} = \emptyset \\ \text{and } \mathsf{fv}(K, P, R) \subseteq \mathsf{dom}(\varphi_0).\}) \\ = \nu\tilde{\omega}.\widehat{\theta}.\end{aligned}$$

Note that $C_{1,0,\epsilon,\epsilon,\epsilon}$ is equal to $\mathsf{e_1}$ and $C_{0,1,\epsilon,\epsilon,\epsilon}$ is equal to $\mathsf{e_2}$ that is why they are not added to the frame, we also have that $\widehat{\varphi}$ is infinite. Using Lemma 2.3, we know that, in order to prove Lemma A.3, it is sufficient to show that: $\widehat{\varphi}\sigma_L \approx_s \widehat{\varphi}\sigma_R$. We introduce the following two claims.

Claim 1. *Let M, N be two terms such that $(\mathsf{fv}(M) \cup \mathsf{fv}(N)) \subseteq \mathsf{dom}(\widehat{\varphi})$ and $\mathsf{fn}(M, N) \cap \tilde{\omega} = \emptyset$. If $M\widehat{\varphi}\sigma =_{AC} N\widehat{\varphi}\sigma$, for some $\sigma \in \{\sigma_L, \sigma_R\}$, then $M\widehat{\varphi} =_{AC} N\widehat{\varphi}$.*

Claim 2. *Let M be a term such that $\mathsf{fv}(M) \subseteq \mathsf{dom}(\widehat{\varphi})$ and $\mathsf{fn}(M) \cap \tilde{\omega} = \emptyset$. If $M\widehat{\varphi}\sigma \longrightarrow U$ for some $\sigma \in \{\sigma_L, \sigma_R\}$, then there exists N such that $\mathsf{fn}(N) \cap \tilde{\omega} = \emptyset$, $U =_{AC} N\widehat{\varphi}\sigma$ and $M\widehat{\varphi}\sigma' \longrightarrow N\widehat{\varphi}\sigma'$ for any $\sigma' \in \{\sigma_L, \sigma_R\}$.*

The above claims allow us to conclude our proof. Let M, N be two terms such that $\mathsf{fn}(M, N) \cap \tilde{\omega} = \emptyset$, $\mathsf{fv}(M, N) \subseteq \mathsf{dom}(\widehat{\varphi})$ and $M\widehat{\varphi}\sigma_L =_E N\widehat{\varphi}\sigma_L$. Thus $(M\widehat{\varphi}\sigma_L)\!\downarrow =_{AC} (N\widehat{\varphi}\sigma_L)\!\downarrow$. Applying repeatedly Claim 2, we deduce that there exists M' such that $(M\widehat{\varphi}\sigma_L)\!\downarrow =_{AC} M'\widehat{\varphi}\sigma_L$ and $M\widehat{\varphi}\sigma_R \longrightarrow^* M'\widehat{\varphi}\sigma_R$. Similarly, there exists N' such that $(N\widehat{\varphi}\sigma_L)\downarrow =_{AC} N'\widehat{\varphi}\sigma_L$ and $N\widehat{\varphi}\sigma_R \longrightarrow^* N'\widehat{\varphi}\sigma_R$. From $M'\widehat{\varphi}\sigma_L =_{AC} N'\widehat{\varphi}\sigma_L$ and Claim 1, we deduce $M'\widehat{\varphi} =_{AC} N'\widehat{\varphi}$. Therefore, $M'\widehat{\varphi}\sigma_R =_{AC} N'\widehat{\varphi}\sigma_R$ and thus $M\widehat{\varphi}\sigma_R =_E N\widehat{\varphi}\sigma_R$.

Proof of Claim 1: Assume by contradiction that there exist M, N two terms such that $M\widehat{\varphi}\sigma =_{AC} N\widehat{\varphi}\sigma$, for some $\sigma \in \{\sigma_L, \sigma_R\}$ and $M\widehat{\varphi} \neq_{AC} N\widehat{\varphi}$. Consider M, N two minimal terms that satisfy this property.

- M and N are both variables or ok. If $M = \mathsf{ok}$, then $N = \mathsf{ok}$ since we can not have $M\widehat{\varphi}\sigma =_{AC} N\widehat{\varphi}\sigma$ with N variable and this case leads to a contradiction. Thus, we have $x\widehat{\varphi}\sigma =_{AC} y\widehat{\varphi}\sigma$ and $x\widehat{\varphi} \neq_{AC} y\widehat{\varphi}$ with $x, y \in \mathsf{dom}(\widehat{\varphi})$ and $x \neq y$. We have $\mathsf{head}(x\widehat{\varphi}\sigma) \in \{\mathsf{penc}, \mathsf{pfk}_1, \mathsf{pk}, \mathsf{s}, \mathsf{sign}, \mathsf{vk}\}$. But, by construction of $\widehat{\varphi}$, σ does not change randomnesses used in penc or in pfk_1 nor the secret ids in pk, s, sign or vk, thus, randomnesses or ids uniquely determine the variable, which implies $x = y$ and a contradiction.

- M is a variable or ok and $N = \mathsf{f}(N_1, \ldots, N_p)$. If $M = \mathsf{ok}$, since $N\widehat{\varphi}\sigma = \mathsf{f}(N_1\widehat{\varphi}\sigma, \ldots, N_p\widehat{\varphi}\sigma)$, we have a contradiction since $M\widehat{\varphi}\sigma \neq_{AC} N\widehat{\varphi}\sigma$ in any case. If M is a variable, we have $\mathsf{head}(M\widehat{\varphi}\sigma) \in \{\mathsf{penc}, \mathsf{pfk}_1, \mathsf{pk}, \mathsf{s}, \mathsf{sign}, \mathsf{vk}\}$. If $\mathsf{head}(M\widehat{\varphi}\sigma) \in \{\mathsf{penc}, \mathsf{pfk}_1, \mathsf{sign}\}$, then, according to Lemma A.2, we have a contradiction upon deducibility of randomnesses t_1, t_2 and secret ids id_1, id_2. If $\mathsf{head}(M\widehat{\varphi}\sigma) \in \{\mathsf{pk}, \mathsf{s}, \mathsf{vk}\}$ we can also exclude cases where $M \in \{\mathsf{idp}_1, \mathsf{idp}_2, \mathsf{s}_1, \mathsf{s}_2, \mathsf{idp}_\mathsf{R}, \mathsf{g}_1, \mathsf{g}_3\}$ using again Lemma A.2. Nevertheless, in all other cases, we have $N = \mathsf{f}(N_1)$ with $\mathsf{f} \in \{\mathsf{pk}, \mathsf{s}, \mathsf{vk}\}$ and since σ does not affect secret ids and secret key, we must have $M\widehat{\varphi} = N\widehat{\varphi}$. Contradiction.

- $M = \mathsf{f}(M_1, \ldots, M_p)$ and $N = \mathsf{g}(N_1, \ldots, N_q)$. We have that $M\widehat{\varphi}\sigma = \mathsf{f}(M_1\widehat{\varphi}\sigma, \ldots, M_p\widehat{\varphi}\sigma)$ $=_{AC} N\widehat{\varphi}\sigma = \mathsf{g}(N_1\widehat{\varphi}\sigma, \ldots, N_q\widehat{\varphi}\sigma)$. We must have $\mathsf{f} = \mathsf{g}$. If f is not an AC-symbol, i.e. in $\{\circ, +, \diamond, *\}$, we have immediately $p = q$ and there must be $i \in [\![1, p]\!]$ such that $M_i\widehat{\varphi} \neq_{AC} N_i\widehat{\varphi}$ but $M_i\widehat{\varphi}\sigma =_{AC} N_i\widehat{\varphi}\sigma$. In that case we will have a contradiction since M_i and N_i will be smaller than M and N. If f is an AC-symbol, e.g. $\circ$, assume we have $M = \overset{p}{\underset{i=1}{\circ}} M_i$, where $\mathsf{head}(M_i) \neq \circ$, and $N = \overset{q}{\underset{i=1}{\circ}} N_i$, where $\mathsf{head}(N_i) \neq \circ$. Then, we must have $p = q$ again and $i, j \in [\![1, p]\!]$ such that $M_i\widehat{\varphi} \neq_{AC} N_j\widehat{\varphi}$ but $M_i\widehat{\varphi}\sigma =_{AC} N_j\widehat{\varphi}\sigma$, which leads, again, to a contradiction.

Proof of Claim 2: This result is proved by inspection of the rewriting rules. More precisely, assume that $M\widehat{\varphi}\sigma \to U$ for some $\sigma \in \{\sigma_L, \sigma_R\}$. It means that there exists a rewriting rule $l \to r$, a substitution θ and a position p such that $M\widehat{\varphi}\sigma|_p =_{AC} l\theta$. p cannot occur below M since $\widehat{\varphi}\sigma$ is in normal form. If $M|_p = l\theta'$ for some θ' then we conclude that we can rewrite M as expected. The only interesting case is thus when $M|_p$ is not an instance of l but $M\widehat{\varphi}\sigma|_p$ is. By inspection of the rules, $l \to r$ can only correspond to one of the equations (E-3) to (E-10).

- The case of equations (E-3) or (E-4) is ruled out by the fact that $a_1 + K$ with free K is not deducible from $\nu\tilde{\omega}.\theta_\delta$ (and thus from $\nu\tilde{\omega}.\theta_0$) according to Lemma A.2 and that $\widehat{\varphi}$ is just an extension of θ_0 including only deducible terms from that one.

- For the case of Equation (E-5), we have several possibilities.
  - If $M|_p = \mathsf{penc}(M_1, M_2, M_3) \circ \mathsf{penc}(M_4, M_5, M_6)$, then since we must have $M_3\widehat{\varphi}\sigma =_{AC} M_6\widehat{\varphi}\sigma$ in order to have a reduction, we have, using Claim 1 that $M_3\widehat{\varphi} =_{AC} M_6\widehat{\varphi}$ then $M|_p\widehat{\varphi} =_E \mathsf{penc}(M_1 \diamond M_3, M_2 * M_4, M_3)\widehat{\varphi}$ and we conclude.
  - If $M|_p = x \circ \mathsf{penc}(M_1, M_2, M_3)$ with $x \in \mathsf{dom}(\widehat{\varphi})$, then we have that $x = y_{\alpha_1,\alpha_2,K,P,R}$ with some $\alpha_1, \alpha_2 \in \mathbb{N}$ and free terms K, P and R and we must have $\mathsf{pk}(\mathsf{a_1}+K)\widehat{\varphi}\sigma =_{AC} M_3\widehat{\varphi}\sigma$. In that case, we have two cases: $M_3 = \mathsf{g_1}$ and $K\widehat{\varphi}\sigma =_{AC} \epsilon$ the null term or $M_3 = \mathsf{g_3}$ and $K\widehat{\varphi}\sigma =_{AC} \mathsf{a_2}$, otherwise we have a contradiction with the fact that $\mathsf{a_1}$ is not deducible (Lemma A.2). In both cases, we have $M|_p\widehat{\varphi} =_E y_{\alpha_1,\alpha_2,K,P\diamond M_1,R*M_2}\widehat{\varphi}$ and we conclude.
  - If $M|_p = x_1 \circ x_2$ with $x_1, x_2 \in \mathsf{dom}(\widehat{\varphi})$, then we have that $x_1 = y_{\alpha_1,\alpha_2,K,P,R}$ and $x_2 = y_{\alpha'_1,\alpha'_2,K',P',R'}$ with $\alpha_1, \alpha_2, \alpha'_1, \alpha'_2 \in \mathbb{N}$ and free terms K, P, R, K', P' and R'. (Note that we can note $\mathsf{e_i} = y_{\delta_{1i},\delta_{2i},\epsilon,\epsilon,\epsilon}$ with δ_{ij} the Kronecker symbol.) Since $M|_p\widehat{\varphi}\sigma$ is reducing, we must have $(\mathsf{a_1} + K)\widehat{\varphi}\sigma =_{AC} (\mathsf{a_1} + K')\widehat{\varphi}\sigma$ which implies $K\widehat{\varphi}\sigma =_{AC} K'\widehat{\varphi}\sigma$. Thus, using Claim 1, we have $K\widehat{\varphi} =_{AC} K'\widehat{\varphi}$. Then, we use that $M|_p\widehat{\varphi} =_E y_{\alpha_1+\alpha'_1,\alpha_2+\alpha'_2,K,P\diamond P',R*R'}\widehat{\varphi}$ to conclude.
- In the case of Equation (E-6), the two non-obvious cases are either the case when $M|_p = \mathsf{renc}(x, U)$ or $M|_p = \mathsf{renc}(\mathsf{penc}(M_1, M_2, x), U)$ with $x \in \mathsf{dom}(\widehat{\varphi})$ and U a free term such that $\mathsf{fn}(U) \cap \tilde{\omega} = \emptyset$.
  - In the first case, we must have that $x = y_{\alpha_1,\alpha_2,K,P,R}$ for some α_1, $\alpha_2 \in \mathbb{N}$ and free terms K, P and R (otherwise there will be no reduction). Then, we have that $M|_p\widehat{\varphi} =_E y_{\alpha_1,\alpha_2,K+U,P,R}\widehat{\varphi}$ which allows us to conclude.
  - In the second case, we have that $x \in \{\mathsf{g_1}, \mathsf{g_2}, \mathsf{g_3}\}$. If $x = \mathsf{g_1}$, we have $M|_p\widehat{\varphi} =_E y_{0,0,U,M_1,M_2}\widehat{\varphi}$ and we conclude. If $x = \mathsf{g_2}$, we have $M|_p\widehat{\varphi} =_E N\widehat{\varphi}$ with $N = \mathsf{penc}(M_1, M_2, \mathsf{pk}(\mathsf{a_2} + U))$ and we conclude. Finally, if $x = \mathsf{g_3}$, we have $M|_p\widehat{\varphi} =_E y_{0,0,\mathsf{a_2}+U,M_1,M_2}\widehat{\varphi}$ and we conclude.
- For the case of Equation (E-7), the result is straightforward since there is nothing in $\widehat{\varphi}$ starting with the function symbol blind.
- In the case of Equation (E-8), we have $M = \mathsf{checksign}(M_1, M_2, M_3)$.
  - If M_3 is a variable, then we must have $M_3 = \mathsf{si_1}$ (or $\mathsf{si_2}$). In that case, $M_1 = \langle \mathsf{e_1}, \mathsf{p_1} \rangle$ (or $\langle \mathsf{e_2}, \mathsf{p_2} \rangle$) and $M_2 = \mathsf{idp_1}$ (or $\mathsf{idp_2}$) otherwise there is no reduction since $\mathsf{t_1}$, $\mathsf{t_2}$, $\mathsf{id_1}$ and $\mathsf{id_2}$ are not deducible according to Lemma A.2. But then we have $M\widehat{\varphi} \to \mathsf{ok}$ and we conclude.
  - If M_3 is not a variable, $M = \mathsf{checksign}(M_1, M_2, \mathsf{sign}(M_3, M_4))$. If M_2 is not a variable, then $M_2 = \mathsf{vk}(M'_2)$ and since $M\widehat{\varphi}\sigma$ is reducing, we have $M_1\widehat{\varphi}\sigma =_{AC} M_3\widehat{\varphi}\sigma$ and $M'_2\widehat{\varphi}\sigma =_{AC} M_4\widehat{\varphi}\sigma$. Then, using Claim 1, we have $M_1\widehat{\varphi} =_{AC} M_3\widehat{\varphi}$ and $M'_2\widehat{\varphi} =_{AC} M_4\widehat{\varphi}$. But, then we have $M\widehat{\varphi} \to \mathsf{ok}$ and we can conclude. If M_2 is variable, then $M_2 = \mathsf{idp_i}$ with $i \in [\![3, n]\!]$ otherwise we have a contradiction with Lemma A.2. Thus, we have $M_4\widehat{\varphi}\sigma =_{AC} \mathsf{id_i}$ and we conclude using the same argument as in the case where M_2 is not a variable.
- In the case of Equation (E-9), we have $M = \mathsf{checkpfk_1}(M_1, M_2, M_3)$.

– If M_3 is a variable, then $M_3 = \mathsf{p}_1$ (or p_2) then, using Lemma A.2, we must have that $M_1 = \mathsf{idp}_1$ (or idp_2) and $M_2 = \mathsf{e}_1$ (or e_2) but then, we have that $M\widehat{\varphi} \to \mathsf{ok}$ and we conclude.

– If M_3 is not a variable, then $M_3 = \mathsf{pfk}_1(M'_1, M'_2, M'_3, M'_4)$. Since $M\widehat{\varphi}\sigma$ is reducing, we have $M_2\widehat{\varphi}\sigma =_{AC} M'_4\widehat{\varphi}\sigma$ and $M_1\widehat{\varphi}\sigma =_{AC} \mathsf{vk}(M'_1)\widehat{\varphi}\sigma$. Using Claim 1, $M_2\widehat{\varphi} =_{AC} M'_4\widehat{\varphi}$ and $M_1\widehat{\varphi} =_{AC} \mathsf{vk}(M'_1)\widehat{\varphi}$ which implies that $M\widehat{\varphi} \to \mathsf{ok}$ and, again, we conclude.

- Finally, case of Equation (E-10) with $M = \mathsf{checkpfk}_2(M_1, M_2, M_3, M_4)$. According to the definition of $\widehat{\varphi}$, M_4 can not be a variable. Then $M_4 = \mathsf{pfk}_2(M'_1, M'_2, M'_3, M'_4, M'_5)$. Since $M\widehat{\varphi}\sigma$ is reducing, we must have that $M_1\widehat{\varphi}\sigma =_{AC} M'_1\widehat{\varphi}\sigma$, $M_2\widehat{\varphi}\sigma =_{AC} M'_4\widehat{\varphi}\sigma$ and $M_3\widehat{\varphi}\sigma =_{AC} M'_5\widehat{\varphi}\sigma$. Using Claim 1, these equalities holds for $\widehat{\varphi}$, but then, we conclude that $M\widehat{\varphi} \to \mathsf{ok}$ and we conclude.

□

A.4 Additional Lemmas

We remind the following notations:

$$\begin{aligned}
\theta_{\mathsf{init}} &= \{^{\mathsf{vk}(\mathsf{id_k})}/_{\mathsf{idp_k}}, {}^{\mathsf{s}(\mathsf{id_k})}/_{\mathsf{s_k}} \mid \mathsf{k} = 1..n\} \mid \{^{\mathsf{vk}(\mathsf{id_R})}/_{\mathsf{idp_R}}\} \mid \{^{\mathsf{pk}(\mathsf{a_k})}/_{\mathsf{g_k}} \mid \mathsf{k} = 1..3\},\\
\theta_0 &= \theta_{\mathsf{init}} \mid \{^{\mathsf{penc}(\mathsf{v_k},\mathsf{t_k},\mathsf{g}_1)}/_{\mathsf{e_k}}, {}^{\mathsf{pfk}_1(\mathsf{id_k},\mathsf{t_k},\mathsf{v_k},\mathsf{e_k})}/_{\mathsf{p_k}}, {}^{\mathsf{sign}(\langle \mathsf{e_k},\mathsf{p_k}\rangle,\mathsf{id_k})}/_{\mathsf{si_k}} \mid \mathsf{k} = 1..2\},\\
\theta_k &= \theta_{k-1} \mid \{^{\mathsf{sign}(\mathsf{hash}(\Pi_1(M_\mathsf{k})),\mathsf{id_R})}/_{sr_k}, {}^{\mathsf{d}(\mathsf{p}(\mathsf{id_k}),\mathsf{dec}(\Pi_2(M_\mathsf{k}),\mathsf{a}_3))}/_{rec_k}\},\\
\hat{\sigma}_i^j &= \{^{M_\mathsf{k}\alpha}/_{x_k} \mid \mathsf{k} = 1..i\} \mid \{^{N_\mathsf{k}\alpha}/_{x_b^k} \mid \mathsf{k} = 1..\mathsf{min}(\mathsf{i},2)\} \mid \{^{U_\mathsf{k}\alpha}/_{d_k}, {}^{W_\mathsf{k}\alpha}/_{hb_k} \mid \mathsf{k} = 1..j\}.
\end{aligned}$$

Lemma A.4. *Let $\phi = \nu\tilde{\omega}.\theta_n$ and $M = f(M_1, \ldots, M_k)$ with $f \in \{\mathsf{vk}, \mathsf{penc}, \mathsf{pfk}_1, \mathsf{sign}\}$. Then, $(M\phi)\!\downarrow = f((M_1\phi)\!\downarrow, \ldots, (M_k\phi)\!\downarrow)$.*

Proof. Let $M = f(M_1, \ldots, M_k)$. Since vk, penc, pfk_1 and sign do not appear in head in any rule of E, we must have $(M\phi)\!\downarrow = f((M_1\phi)\!\downarrow, \ldots, (M_k\phi)\!\downarrow)$ and we conclude.

□

Lemma A.5. *Let $\phi = \nu\tilde{\omega}.\theta_n$ and $M = f(M_1, \ldots, M_k)$ a minimal recipe of $(M\phi)\!\downarrow$ with $f \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$. Then, $(M\phi)\!\downarrow = f((M_1\phi)\!\downarrow, \ldots, (M_k\phi)\!\downarrow)$.*

Proof. Let us prove this by induction on the depth of M.

Base Case: $M = f(M_1, \ldots, M_k)$ with $M_1, \ldots, M_k$ variables or names.

- If $f \in \{\mathsf{fst}, \mathsf{snd}\}$, since $\nexists\, x \in \mathsf{dom}(\phi)$ such that $\mathsf{head}(x\phi) = \mathsf{pair}$, we conclude immediately.
- If $f = \mathsf{dec}$ and $M = \mathsf{dec}(M_1, M_2)$, then $M_1 \in \{\mathsf{e}_1, \mathsf{e}_2\}$ but if there is a reduction, it means that we have $M_2\phi = \mathsf{a}_1$ which is impossible as a_1 is not deducible by Lemma A.2.
- If $f = \mathsf{unblind}$, since $\nexists\, x \in \mathsf{dom}(\phi)$ such that $\mathsf{head}(x\phi) = \mathsf{blind}$, we conclude immediately.

Induction Hypothesis: We suppose that all terms M with a depth $\leq n$ satisfy the property. Let $M = f(M_1, \ldots, M_k)$ of depth equal to $n+1$ with M_i of depths $\leq n$. If $f((M_1\phi)\!\downarrow, \ldots, (M_k\phi)\!\downarrow)$ is in normal form, we conclude. Suppose that $f((M_1\phi)\!\downarrow, \ldots, (M_k\phi)\!\downarrow)$ is not in normal form.

- If $f \in \{\mathsf{fst}, \mathsf{snd}\}$ and $M = f(M_1)$. The case where M_1 is a variable is excluded. Then $M_1 = g(M'_1, \ldots, M'_k)$ with $g \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{pair}, \mathsf{snd}, \mathsf{unblind}\}$.
  - If $g \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$, using the induction hypothesis, we have $(M_1\phi)\!\downarrow = g((M'_1\phi)\!\downarrow, \ldots, (M'_k\phi)\!\downarrow)$ and, thus, $(M\phi)\!\downarrow = f((M_1\phi)\!\downarrow)$.
  - If $g = \mathsf{pair}$ we have a contradiction with the minimality of M.
- If $f = \mathsf{dec}$ and $M = \mathsf{dec}(M_1, M_2)$. The case where M_1 is a variable is excluded (see Base case). Then $M_1 = g(M'_1, \ldots, M'_k)$ with $g \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{penc}, \mathsf{renc}, \mathsf{snd}, \mathsf{unblind}, \circ\}$.
  - If $g \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$, we conclude thanks to the induction hypothesis.
  - If $g = \mathsf{penc}$, there is a contradiction with the minimality of M.
  - If $g = \mathsf{renc}$ and $M_1 = \mathsf{renc}(M'_1, M'_2)$. The case where M'_1 is a variable is still excluded (see Base case). Then $M'_1 = g_1(M''_1, \ldots, M''_k)$ with $g_1 \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{penc}, \mathsf{snd}, \mathsf{unblind}, \circ\}$ where all cases, except $g_1 = \circ$ lead to a contradiction using the induction hypothesis or minimality of M. If $g_1 = \circ$ and $M'_1 = \circ(M''_1, M''_2)$, we can still exclude the case where M''_1 is a variable and then $M''_1 = g_2(M'''_1, \ldots, M'''_k)$ with $g_2 \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{penc}, \mathsf{renc}, \mathsf{snd}, \mathsf{unblind}\}$. Again, the only viable case is $g_2 = \mathsf{renc}$. Then, repeating this argument (a finite number of time since depths are finite) we have that $M_1 = \mathsf{renc}(\circ(\mathsf{renc}(\cdots \circ (n_1, n_2))))$ or $M_1 = \mathsf{renc}(\circ(\mathsf{renc}(\ldots \mathsf{renc}(n_1, n_2))))$ with n_1 and n_2 of length 1. But then we have a contradiction since n_1 and n_2 can not be names (then there are no reduction) or variables (see Base case).
  - Finally, the case when $g = \circ$. Using a similar argument close to the one for the previous case, we have that $M_1 = \circ(\mathsf{renc}(\cdots \circ (m_1, m_2)))$ or $M_1 = \circ(\mathsf{renc}(\ldots \mathsf{renc}(m_1, m_2)))$ with m_1 and m_2 of length 1 and we conclude in the same way.
- If $f = \mathsf{unblind}$ and $M = \mathsf{unblind}(M_1, M_2)$. The case where M_1 is a variable is excluded (see Base case). Then $M_1 = g(M'_1, \ldots, M'_k)$ with $g \in \{\mathsf{blind}, \mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$.
  - If $g \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$, we conclude thanks to the induction hypothesis.
  - If $g = \mathsf{blind}$, there is a contradiction with the minimality of M.

□

Lemma A.6. *Let $\phi = \nu\tilde{\omega}.\theta_n$ and M a minimal recipe of $(M\phi)\!\downarrow = f(M_1, \ldots, M_k)$ with $f \in \{\mathsf{vk}, \mathsf{pfk}_1, \mathsf{sign}\}$. Then M is a variable or $M = f(M'_1, \ldots, M'_k)$.*

Proof. Suppose, by contradiction, that M is not a variable and $M = f(M_1, \ldots, M_p)$ and s.t. $(M\phi)\!\downarrow = g(N_1, \ldots, N_q)$ with $g \in \{\mathsf{pfk}_1, \mathsf{sign}, \mathsf{vk}\}$ and $f \neq g$. This would imply that $\exists\ l \to r$ a reduction rule and θ a substitution such that $\mathsf{head}(l\theta) = f$ and $\mathsf{head}(r\theta) = g$. Then, it must be the case that $f \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$. But then, we have a contradiction, according to Lemma A.5.

□

A.5 Shape of Valid Ballots

Definition A.2. *We define $|.| : T(\Sigma, \mathcal{X}, \mathcal{N}) \to \mathbb{N}$ recursively as follows:*

- $|u| = 1$ *for u a constant or a variable,*

- $|f(u_1,\ldots,u_n)| = \sum |u_i|$ *if* $f \in \{+,*,\circ,\diamond\}$.
- $|f(u_1,\ldots,u_n)| = 1 + \sum |u_i|$ *otherwise.*

We also define $L : T(\Sigma,\mathcal{X},\mathcal{N}) \to \mathbb{N}\times\mathbb{N}$ *which is defined as* $L(M) = (||M||,|M|)$ *where* $||.|| : T(\Sigma,\mathcal{X},\mathcal{N}) \to \mathbb{N}$ *is defined as follows with* $\#_\circ(M)$ *the number of* $\circ$ *symbols in* M*:*

- $||u|| = 0$ *for* u *a constant or a variable,*
- $||f(u_1,\ldots,u_n)|| = \begin{cases} \#_\circ(f(u_1,\ldots,u_n)) + \sum ||u_i||, & f \in \{\mathsf{penc},\mathsf{renc}\} \\ \sum ||u_i||, & \text{otherwise.} \end{cases}$

Definition A.3. *Let* $id \in \{\mathsf{id_1},\ldots,\mathsf{id_n}\}$*. A term* M *is said to be a* id-valid ballot *if* $\phi_{\mathsf{B}}(id,M)$ *is verified, equivalently:*

$$\begin{cases} M & = & \langle M_1,M_2,M_3,M_4\rangle \\ M_1 & =_E & \mathsf{vk}(id) \\ \mathsf{checksign}(\langle M_2,M_3\rangle,M_1,M_4) & =_E & \mathsf{ok} \\ \mathsf{checkpfk_1}(M_1,M_2,M_3) & =_E & \mathsf{ok} \end{cases}.$$

We first give the general shape of the two first ballots submitted by honest voters on which we have extra information.

Lemma A.7. *We consider* $i \in \{1,2\}$ *and* M *a free term such that* $\mathsf{fv}(M) \subseteq \mathsf{dom}(\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L)$ *and* $M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ *be an* id_i*-valid ballot. We suppose that* $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$*. Then, we have, for* $\sigma \in \{\sigma_L,\sigma_R\}$ *:*

$$M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \langle \mathsf{idp_i},\mathsf{e_i},\mathsf{p_i},\mathsf{si_i}\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma.$$

Proof. Let $i \in \{1,2\}$ and M a free term s.t. $\mathsf{fv}(M) \subseteq \mathsf{dom}(\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L)$ and $M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ is a id_i-valid ballot. Let M' minimal in size - according to the measure of length L defined in Definition A.2 - such that: $M'\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$.

According to Definition A.3, we have $M'\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \langle P_1,P_2,P_3,P_4\rangle$.

- Let suppose that M' is a variable. Since $\nexists\ x \in \mathsf{dom}(\theta_{i-1})$ such that $x\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ is a id_i-valid ballot, there is a contradiction.
- Thus, $M' = f(M_1,\ldots,M_n)$ with $f \in \{\mathsf{dec},\mathsf{fst},\mathsf{pair},\mathsf{snd},\mathsf{unblind}\}$ since only equations (E-1), (E-2), (E-3) and (E-7) can lead to $\langle P_1,P_2,P_3,P_4\rangle$.
  - If $f \in \{\mathsf{dec},\mathsf{fst},\mathsf{snd},\mathsf{unblind}\}$, using Lemma A.5, we have a contradiction with the fact that $\mathsf{head}(M'\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L) = \mathsf{pair}$.
  - If $f = \mathsf{pair}$, then $M' = \langle M_1,M''\rangle$, with some free M_1. By repeating this reasoning, we get that $M' = \langle M_1,M_2,M_3,M_4\rangle$, with some free M_i for $i \in [\![1,4]\!]$. Thus we have :

$$\begin{aligned} M'\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L &=_E \langle M_1,M_2,M_3,M_4\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \\ &=_E M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L. \end{aligned}$$

  Since $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$, then, for $\sigma \in \{\sigma_L,\sigma_R\}$:

$$M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \langle M_1,M_2,M_3,M_4\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma.$$

Moreover, using again Definition A.3, we have that:

$$M_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{vk}(\mathsf{id_i}),$$
$$M_3\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{pfk}_1(\mathsf{id_i}, N_1, N_2, M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L),$$
$$M_4\hat{\sigma}_{i-1}\sigma^0_{i-1}\sigma_L =_E \mathsf{sign}(\langle M_2, M_3\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L, \mathsf{id_i}).$$

According to Lemma A.6, we know that M_1, M_3 and M_4 must be variables or constructed terms. But since, according Lemma A.2, $\mathsf{id_i}$ is not deducible, M_1, M_3 and M_4 must be variables. In that case, we must have $M_1 = \mathsf{idp_i}$, $M_3 = \mathsf{p_i}$ and $M_4 = \mathsf{si_i}$. And, according to the fact that $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$, we have, for $\sigma \in \{\sigma_L, \sigma_R\}$ and for some free M_2:

$$M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \langle \mathsf{idp_i}, M_2, \mathsf{p_i}, \mathsf{si_i}\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma.$$

Now, since $M_3 = \mathsf{p_i}$ and according to Definition A.3, we must have that $M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{penc}(\mathsf{v_i}, \mathsf{t_i}, \mathsf{g_1})$ with M_2 free. Thus, M_2 is a variable or $M_2 = f(M'_1, \ldots, M'_k)$ with $f \in \{\circ, \mathsf{dec}, \mathsf{fst}, \mathsf{penc}, \mathsf{renc}, \mathsf{snd}, \mathsf{unblind}\}$:

- If $f \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$, using Lemma A.5, we get a contradiction on the fact that $\mathsf{head}(M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L) = \mathsf{penc}$.
- If $M_2 = \mathsf{penc}(M'_1, M'_2, M'_3)$ with free terms M'_1, M'_2 and M'_3, then using Lemma A.4 we must have $M'_2 = \mathsf{t_i}$ which is in contradiction with Lemma A.2 and the fact that $\mathsf{t_i}$ is not deducible.
- If $M_2 = \mathsf{renc}(M'_1, M'_2)$ with M'_1 and M'_2 free terms. We must have $M'_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{penc}(M''_1, M''_2, \mathsf{pk}(M''_3))$ leading to the fact that $\mathsf{a_1} =_{AC} M''_3 + M'_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ where M'_2 is free which is a contradiction with Lemma A.2.
- If $M_2 = \overset{p}{\underset{k=1}{\circ}} M'_k$ with M'_k free and $\mathsf{head}(M'_k) \neq \circ$ for $k \in [\![1, p]\!]$. To have a reduction, we must have, for $k \in [\![1, p]\!]$, that $M'_k\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{penc}(M^1_k, M^2_k, \mathsf{pk}(M^3_k))$, and since $\mathsf{e_1}$ and $\mathsf{e_2}$ are the only variables leading to a penc-headed term, we must have one M'_k which is not a variable, otherwise we would have $\mathsf{t_1}^q * \mathsf{t_2}^s = \mathsf{t_1}$ with $q+s = p \geq 2$. Due to the study of previous cases, we must have that $M'_k = \mathsf{penc}(M''_1, M''_2, M''_3)$ or $\mathsf{renc}(\mathsf{penc}(M''_1, M''_2, M''_3), M''_4)$, but this leads to $M''_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L * U = \mathsf{t_1}$ with free M''_2 which leads to a contradiction due to Lemma A.2.

Thus, we have M_2 variable and $M_2 = \mathsf{e_1}$, which allows us to conclude the proof, using equivalence between the two frames.

□

We know can give the general shape of the ballots submitted by the intruder, under the condition they are accepted by the Ballot box.

Lemma A.8. *Using previous notations, let $i \in [\![3, n]\!]$ and M a free term s.t. $\mathsf{fv}(M) \subseteq \mathsf{dom}(\theta_{i-1}\hat{\sigma}^0_{i-1})$ and $M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ is an id_i-valid ballot. We suppose that $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$. Then, for $\sigma \in \{\sigma_L, \sigma_R\}$:*

- $M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \langle M_1, M_2, M_3, M_4\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ *with free* $M_1, \ldots, M_4$,
- $M_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{vk}(\mathsf{id_i})\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$,
- $M_3\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{pfk}_1(\mathsf{id_i}, N_1, N_2, N_3)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ *with free* $N_1, \ldots, N_3$,

- $M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{penc}(N_2, N_1, U)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ *with free* U *or* $U =_{AC} \mathsf{pk}(\mathsf{a_p} + U')$ *with free* U' *and* $\mathsf{a_p} \in \{\mathsf{a_1}, \mathsf{a_3}\}$.

Proof. Let $i \in \{3, \ldots, n\}$. Let M such that $M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ is a id_i-valid ballot. Let M' minimal in size - according to the measure of length L defined in Definition A.2 - such that :

$$M'\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \qquad (\dagger)$$

Using Definition A.3, we have $M'\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L = \langle P_1, P_2, P_3, P_4\rangle$.

- Let suppose that M'_i is a variable. Since $\nexists\ x \in \mathsf{dom}(\theta_{i-1})$ such that $x\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ is a id_i-valid ballot, there is a contradiction.

- Thus, $M' = f(M_1, \ldots, M_p)$ with $f \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{pair}, \mathsf{snd}, \mathsf{unblind}\}$ since only equations (E-1), (E-2), (E-3) and (E-7) can lead to $\langle P_1, P_2, P_3, P_4\rangle$.

  - If $f \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$, using Lemma A.5, we have a contradiction with the fact that $\mathsf{head}(M'_i\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L) = \mathsf{pair}$.
  - If $f = \mathsf{pair}$, then $M' = \langle M_1, M''\rangle$, with some free M_1. By repeating this reasoning, we get that $M' = \langle M_1, M_2, M_3, M_4\rangle$, with some free terms M_1, M_2, M_3 and M_4. Thus we have :

$$\begin{aligned} M'\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L &=_E \langle M_1, M_2, M_3, M_4\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \\ &=_E M\theta_{i-1}\sigma^0_{i-1}\sigma_L. \end{aligned}$$

  Since $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$, then, for $\sigma \in \{\sigma_L, \sigma_R\}$:

$$M\theta_{i-1}\sigma^0_{i-1}\sigma =_E \langle M_1, M_2, M_3, M_4\rangle\theta_{i-1}\sigma^0_{i-1}\sigma.$$

Definition A.3 implies that $M_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{vk}(\mathsf{id_i})$. Since $\mathsf{id_i}$ is deducible (it is not restricted), we have here a public test and the equivalence is sufficient to show that $M_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{vk}(\mathsf{id_i})\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ for $\sigma \in \{\sigma_L, \sigma_R\}$.
Moreover, using Definition A.3, we also have that $M_3\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{pfk}_1(\mathsf{id_i}, P_1, P_2, P_3)$.

- M_3 cannot be a variable since there is no $x \in \mathsf{dom}(\theta_{i-1})$ such that $x\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L = \mathsf{pfk}_1(\mathsf{id_i}, P_1, P_2, P_3)$ with $i \geq 3$.

- Thus, $M_3 = f(N_1, \ldots, N_p)$ with $f \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{pfk_1}, \mathsf{snd}, \mathsf{unblind}\}$ since only equations (E-1), (E-2), (E-3) and (E-7) lead to $\mathsf{pfk}_1(\mathsf{id_i}, P_1, P_2, P_3)$.

  - If $f \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$, using Lemma A.5, we have a contradiction with the fact that $\mathsf{head}(M_3\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L) = \mathsf{pfk_1}$.
  - If $f = \mathsf{pfk_1}$, then $M_3 = \mathsf{pfk}_1(\mathsf{id_i}, N_1, N_2, N_3)$, with some free N_1, N_2, N_3. Since $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$, we have, for $\sigma \in \{\sigma_L, \sigma_R\}$:

$$M_3\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{pfk}_1(\mathsf{id_i}, N_1, N_2, N_3)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma.$$

Finally, Definition A.3 gives us that, for some term U, we have:

$$M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{penc}(N_2\theta_{i-1}\sigma_{i-1}\sigma_L, N_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L, U).$$

- If M_2 is a variable, then $M_2 \in \{\mathsf{e}_1, \mathsf{e}_2\}$. In that case, we would have $N_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{t}_1$ (or t_2) with free N_1 which would mean that t_1 (or t_2) is deducible which is in contradiction with Lemma A.2.
- Thus, $M_2 = f(V_1, \ldots, V_p)$ with $f \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{penc}, \mathsf{renc}, \mathsf{snd}, \mathsf{unblind}, \circ\}$ since only equations (E-1) to (E-3) and (E-5) to (E-7) lead to $\mathsf{penc}(P_1, P_2, P_3)$.
  - If $f \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$, using Lemma A.5, we have a contradiction with the fact that $\mathsf{head}(M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L) = \mathsf{penc}$.
  - If $f = \mathsf{renc}$ i.e. $M_2 = \mathsf{renc}(V_1, V_2)$. If V_1 is variable, then $V_1 \in \{\mathsf{e}_1, \mathsf{e}_2\}$ and we would have a contradiction with Lemma A.2 since we would have $N_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{t_i}$ and $\mathsf{t_i}$ would be deducible. Then $V_1 = g(V'_1, \ldots, V'_p)$ with $g \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{penc}, \mathsf{renc}, \mathsf{unblind}, \circ\}$ since $\mathsf{head}(V_1\theta_{i-1}\sigma^0_{i-1}\sigma_L) = \mathsf{penc}$.
    - If $g \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$, using Lemma A.5, we have a contradiction with the fact that $\mathsf{head}(V_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L) = \mathsf{penc}$.
    - If $g = \mathsf{renc}$ and $V_1 = \mathsf{renc}(V'_1, V'_2)$, we have a contradiction with the minimality of M_2 since $\mathsf{renc}(V'_1, V'_2 + V_2)$ is a smaller recipe than $\mathsf{renc}(\mathsf{renc}(V'_1, V'_2), V_2)$ for L.
    - If $g = \circ$ and $V_1 = V'_1 \circ V'_2$, we also have a contradiction with the minimality of M_2 since $\mathsf{renc}(V'_1, V_2) \circ \mathsf{renc}(V'_2, V_2)$ is a smaller recipe than $\mathsf{renc}(V'_1 \circ V'_2, V_2)$ according to the Definition A.2 of the measure L. Indeed, since $M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ is reducing, we have $(V'_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L)\downarrow = \mathsf{penc}(U_1, U'_1, U''_1)$ and $(V'_2\theta_{i-1}\sigma^0_{i-1}\sigma_L)\downarrow = \mathsf{penc}(U_2, U'_2, U''_2)$ with $U''_1 = U''_2$ implying that the two recipes lead to the same term.
    - If $g = \mathsf{penc}$ and $V_1 = \mathsf{penc}(V'_1, V'_2, V'_3)$. We have two cases :
      - If V'_3 is a variable, then $V'_3 \in \{\mathsf{g}_1, \mathsf{g}_2, \mathsf{g}_3\}$ and $M_2 = \mathsf{renc}(\mathsf{penc}(V'_1, V'_2, \mathsf{g_p}), V_2)$ with free terms V'_1, V'_2 and V_2. In that case, we have :

$$\begin{aligned} M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L &=_E \mathsf{renc}(\mathsf{penc}(V'_1, V'_2, \mathsf{g_p}), V_2)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L(\ddagger) \\ &=_E \mathsf{penc}(V'_1, V'_2, \mathsf{pk}(\mathsf{a_p} + V_2))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \end{aligned}$$

        Thanks to the fact that $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$ and $(\ddagger)$, we also have that:

$$\begin{aligned} M_2\theta_{i-1}\hat{\sigma}_{i-1}\sigma_R &=_E \mathsf{renc}(\mathsf{penc}(V'_1, V'_2, \mathsf{g_p}), V_2)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R \\ &=_E \mathsf{penc}(V'_1, V'_2, \mathsf{pk}(\mathsf{a_p} + V_2))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R \end{aligned}$$

        Then, we have, for $\sigma \in \{\sigma_L, \sigma_R\}$:

$$M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{penc}(V'_1, V'_2, \mathsf{pk}(\mathsf{a_p} + V_2)))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma.$$

        Since

$$M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{penc}(N_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L, N_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L, U)$$

        we have that $V'_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E N_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ and $V'_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E N_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$. Using the fact that $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$, these equalities hold with σ_R. Finally, for $\sigma \in \{\sigma_L, \sigma_R\}$ and free V_2:

$$M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{penc}(N_2, N_1, \mathsf{pk}(\mathsf{a_p} + V_2))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma.$$

        (With a_2 deducible, $\mathsf{pk}(\mathsf{a}_2 + V_2)$ can be seen as a free U.)

· If $V_3' = h(V_1'', \ldots, V_q'')$ with $h \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{pk}, \mathsf{snd}, \mathsf{unblind}\}$, we conclude easily with a contradiction when $h \neq \mathsf{pk}$ using Lemma A.5. If $h = \mathsf{pk}$ then there is contradiction with minimality of M_2 since $\mathsf{penc}(V_1', V_2', \mathsf{pk}(V_1'' + V_2))$ is smaller than $\mathsf{renc}(\mathsf{penc}(V_1', V_2', \mathsf{pk}(V_3'')), V_2)$.

– If $f = \mathsf{penc}$ i.e. $M_2 = \mathsf{penc}(V_1, V_2, V_3)$ with free terms V_1, V_2, V_3, we immediately conclude that, for $\sigma \in \{\sigma_L, \sigma_R\}$:

$$M_2\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma =_E \mathsf{penc}(V_1, V_2, V_3)\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma.$$

Using that $V_1\theta_{i-1}\hat{\sigma}_{i-1}^0\Sigma =_E N_2\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma$ and $V_2\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma =_E N_1\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma$, we have, with free V_3 and $\sigma \in \{\sigma_L, \sigma_R\}$:

$$M_2\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma =_E \mathsf{penc}(N_2, N_1, V_3)\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma.$$

– If $f = \circ$ i.e. $M_2 = V_1 \circ V_2$. Then, we can assume, without loss of generality, that $\mathsf{head}(V_1) \neq \circ$. Indeed, if $V_1 = V_1' \circ V_1''$, then we write $M_2 = V_1' \circ (V_1'' \circ V_2)$. Then, we have $M_2 = V_1 \circ \left(\circ_{j=2}^{p} V_j\right)$ s.t. $\forall\ j,\ \mathsf{head}(V_j) \neq \circ$. If $\exists\ j$ such that V_j is a variable, then $V_j \in \{\mathsf{e}_1, \mathsf{e}_2\}$ and $N_1\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma_L = r_k + U$ (where $k \in \{1, 2\}$) with free N_1 which is a contradiction with Lemma A.2 since $r_k + U$ is not deducible. Then, $\forall\ j \in [\![1, n]\!]$, $V_j = h_j(V_1^j, \ldots, V_q^j)$ with $h_j \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{penc}, \mathsf{renc}, \mathsf{unblind}\}$ since $\mathsf{head}(V_j\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma_L) = \mathsf{penc}$ otherwise $\mathsf{head}(V\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma_L) \neq \mathsf{penc}$. Now, according to Lemma A.5, we have that $h_j \in \{\mathsf{penc}, \mathsf{renc}\}$ for $j \in [\![1, n]\!]$. Then:

$$M_2 = \left(\overset{p_1}{\underset{j=1}{\circ}} \mathsf{penc}(V_1^j, V_2^j, V_3^j)\right) \circ \left(\overset{p_2}{\underset{j=1}{\circ}} \mathsf{renc}(V_4^j, V_5^j)\right).$$

If $p_1 \geq 2$, then we have a contradiction with the minimality of M_2 since $\mathsf{penc}(V_1^1 \diamond V_1^2, V_2^1 * V_2^2, V_3^1)$ is a smaller recipe than $\mathsf{penc}(V_1^1, V_2^1, V_3^1) \circ \mathsf{penc}(V_1^2, V_2^2, V_3^2)$ according to Definition A.2. Thus $p_1 \in \{0, 1\}$. Moreover, using the case $f = \mathsf{renc}$ we also must have that $V_4^j = \mathsf{penc}(W_1^j, W_2^j, W_3^j)$ with W_3^j a variable.

∗ Suppose that $p_1 = 1$, then $W_3^1\theta_{i-1}\hat{\sigma}_{i-1}\sigma_L$ is deducible. In order for M_2 to be reduced, we must have the same keys in both W_3^1 and $\circ_{j=1}^{p_2}\mathsf{renc}(V_4^j, V_5^j)$. But since $V_4^j = \mathsf{penc}(W_1^j, W_2^j, W_3^j)$ with W_3^j a variable, then $W_3^j \in \{\mathsf{g}_1, \mathsf{g}_2, \mathsf{g}_3\}$, and we must have $W_3^j = \mathsf{g}_2$ otherwise this would imply that a_1 or a_3 is deducible which is contradiction with Lemma A.2. Since $\forall j \in [\![1, n]\!]$ we have $W_3^j = \mathsf{a}_2$, we also have that $\forall j \in [\![1, n]\!]$, $V_5^j = V_5^1$ (with V_5^1 the minimal recipe in all the V_5^j) otherwise we will not have $\mathsf{pk}(\mathsf{a}_2 + V_5^j)\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma_L =_E \mathsf{pk}(\mathsf{a}_2 + V_5^1)\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma_L$ and there will be no reduction. Then, finally, we have:

$$M_2 = \left(\overset{p_2}{\underset{j=1}{\circ}} \mathsf{renc}(\mathsf{penc}(W_1^j, W_2^j, \mathsf{g}_2), V_5^1)\right) \circ \mathsf{penc}(V_1^1, V_2^1, \mathsf{pk}(\mathsf{a}_2 + V_5^1)).$$

Then, we have :

$$\begin{aligned} M_2\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma_L = &\left[\circ_{j=1}^{p_2}\mathsf{renc}(\mathsf{penc}(W_1^j, W_2^j, \mathsf{g}_2), V_5^1)\right. \\ &\left. \circ \mathsf{penc}(V_1^1, V_2^1, \mathsf{pk}(\mathsf{a}_2 + V_5^1))\right] \theta_{i-1}\hat{\sigma}_{i-1}^0\sigma_L \ (\star) \\ =_E\ & \mathsf{penc}(\diamond_{j=1}^{p_2} W_1^j \diamond V_1^1, *_{j=1}^{p_2} W_2^j * V_2^1, \mathsf{pk}(\mathsf{a}_2 + V_5^1))\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma_L. \end{aligned}$$

Thanks to the fact that $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$ and $(\star)$, we also have that:

$$\begin{aligned}M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R = &\Big[\circ_{j=1}^{p_2}\mathsf{renc}(\mathsf{penc}(W_1^j, W_2^j, \mathsf{g}_2), V_5^1)\\ &\circ \mathsf{penc}(V_1^1, V_2^1, \mathsf{pk}(\mathsf{a}_2 + V_5^1))\Big]\,\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R\ (\star)\\ =_E\ &\mathsf{penc}(\diamond_{j=1}^{p_2} W_1^j \diamond V_1^1, *_{j=1}^{p_2} W_2^j * V_2^1, \mathsf{pk}(\mathsf{a}_2 + V_5^1))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R.\end{aligned}$$

Then, we have, for $\sigma \in \{\sigma_L, \sigma_R\}$:

$$M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{penc}(V_1', V_2', \mathsf{pk}(\mathsf{a}_2 + V_3'))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma.$$

∗ If $p_1 = 0$, we have $M = \circ_{j=1}^{p_2}\mathsf{renc}(\mathsf{penc}(W_1^j, W_2^j, W_3^j), V_5^j)$ with W_3^j variables. To have a reduction, we still need to have that $\forall p, q \in [\![1, n]\!]\ \mathsf{pk}(\mathsf{a_p}+V_5^p) = \mathsf{pk}(\mathsf{a_q}+V_5^q)$ where $\mathsf{a_p}, \mathsf{a_q} \in \{\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3\}$. That leads us to $M_2 = \circ_{j=1}^{p_2}\mathsf{renc}(\mathsf{penc}(W_1^j, W_2^j, \mathsf{g_k}), V_5^1)$ with $k \in \{1, 2, 3\}$ and free W_1^j, W_2^j, V_5^1 (V_5^1 is the minimal recipe among all V_5^j).

Then, we have :

$$\begin{aligned}&M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L\\ &\quad= \circ_{j=1}^{p_2}\mathsf{renc}(\mathsf{penc}(W_1^j, W_2^j, \mathsf{g_k}), V_5^k)\theta_{i-1}\sigma^0_{i-1}\sigma_L\ (\dagger\dagger)\\ &\quad=_E \mathsf{penc}(\diamond_{j=1}^{p_2} W_1^j, *_{j=1}^{p_2} W_2^j, \mathsf{pk}(\mathsf{a_k} + V_5^1))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L.\end{aligned}$$

Thanks to the fact that $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$ and $(\dagger\dagger)$, we also have that:

$$\begin{aligned}&M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R\\ &\quad= \circ_{j=1}^{p_2}\mathsf{renc}(\mathsf{penc}(W_1^j, W_2^j, \mathsf{g_k}), V_5^1)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R\\ &\quad=_E \mathsf{penc}(\diamond_{j=1}^{p_2} W_1^j, *_{j=1}^{p_2} W_2^j, \mathsf{pk}(\mathsf{a_k} + V_5^1))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R.\end{aligned}$$

Then, we have, for $\sigma \in \{\sigma_L, \sigma_R\}$:

$$M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{penc}(V_1', V_2', \mathsf{pk}(\mathsf{a_k} + V_3'))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma.$$

Then, in both cases ($p_1 = 1$ or $p_1 = 0$), we have, for $\sigma \in \{\sigma_L, \sigma_R\}$, there is $i \in [\![1, 3]\!]$ such that:

$$M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{penc}(V_1', V_2', \mathsf{pk}(\mathsf{a_k} + V_3'))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma.$$

Since $M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{penc}(N_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L, N_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L, U)$ then $V_1'\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E N_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ and $V_2'\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E N_1\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$. Using the fact that $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$, these equalities hold with σ_R. Finally, we have, for $\sigma \in \{\sigma_L, \sigma_R\}$:

$$M_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{penc}(N_2, N_1, \mathsf{pk}(\mathsf{a_k} + V_3'))\theta_{i-1}\sigma^0_{i-1}\sigma.$$

Again, $\mathsf{pk}(\mathsf{a}_2 + V_3')$ can be seen as a free term U since a_2 is deducible.

□

The next lemma gives the general shape of a ballot submitted by the intruder provided the fact that it is accepted by the Receipt generator in the first execution and shows that it is also accepted in the second execution.

Lemma A.9. *We consider $i \in [\![1,n]\!]$ and M a free term such that $\mathsf{fv}(M) \subseteq \mathsf{dom}(\theta_{i-1}\hat{\sigma}^0_{i-1})$ and such that $\phi_{\mathsf{R}}(\mathsf{idp_i}, M)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{ok}$ with:*

$$\begin{aligned}\phi_{\mathsf{R}}(id,x) = (x = \langle x_1,x_2,x_3\rangle) \wedge (x_1 = \langle y_1,y_2,y_3,y_4\rangle) \wedge (y_1 = id)\\ \wedge (\mathsf{checksign}(\langle y_2,y_3\rangle, y_1, y_4)) \wedge (\mathsf{checkpfk_1}(y_1,y_2,y_3))\\ \wedge (\mathsf{checkpfk_2}(y_1,y_2,x_2,x_3))\,.\end{aligned}$$

Assuming that $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$, then, for $\sigma \in \{\sigma_L, \sigma_R\}$:

- $\phi_{\mathsf{R}}(\mathsf{idp_i}, M)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{ok}$,
- $M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \langle P,Q,R\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ *for free terms P, Q and R.*
- *For $i \in \{1,2\}$, we have:*
  - $P\theta_{i-1}\hat{\sigma}a^0_{i-1}\sigma =_E \langle \mathsf{idp_i}, \mathsf{e_i}, \mathsf{p_i}, \mathsf{si_i}\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$
  - $Q\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{blind}(\mathsf{renc}(\mathsf{e_i}, Q_1), Q_2)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ *with Q_1 and Q_2 free terms.*
- *For $i \in [\![3,n]\!]$, we have:*
  - $P\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \langle P_1,P_2,P_3,P_4\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ *with $P_1,\ldots,P_4$ free terms,*
  - $Q\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{blind}(\mathsf{penc}(Q_1,Q_2,U),Q_3)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ *with free Q_1, Q_2, Q_3, U or $U =_{AC} \mathsf{pk}(\mathsf{a_p} + U')$ with free U' and $\mathsf{a_p} \in \{\mathsf{a_1}, \mathsf{a_3}\}$.*

Proof.

- Using the fact that $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$ and since $\phi_{\mathsf{R}}(\mathsf{idp_i}, M)$ is a public test, we have that the first property holds obviously.
- Now, using that $\phi_{\mathsf{R}}(\mathsf{idp_i}, M)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{ok}$, we have that, $M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \langle M_1,M_2,M_3\rangle$. By repeating the same reasoning as we did in the proof of Lemma A.8, we have that $M\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \langle P,Q,R\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ with free term P, Q and R for $\sigma \in \{\sigma_L, \sigma_R\}$.
- Let $i \in \{1,2\}$.
  - Since $\phi_{\mathsf{R}}(\mathsf{idp_i}, M)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{ok}$, we have that $P\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ must be an id_i-valid ballot. Then, we use Lemma A.7 to conclude on the form of $P\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$.
  - We also have that $\mathsf{checkpfk_2}(\mathsf{idp_i}, \mathsf{e_i}, Q, R)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{ok}$, thus $R\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E \mathsf{pfk_2}(R_1,R_2,R_3,R_4,R_5)$ with $R_1 = \mathsf{vk}(\mathsf{id_i})$ and $R_4 = \mathsf{e_i}\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$. We consider R' minimal in size (for the measure L) such that $R\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L =_E R'\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$.
    - R' cannot be a variable since there is no variable $x \in \mathsf{dom}(\theta_{i-1})$ such that $\mathsf{head}(x\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L) = \mathsf{pfk_2}$.
    - $R' = f(R_1,\ldots,R_p)$ with $f \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{pfk_2}, \mathsf{snd}, \mathsf{unblind}\}$ since only equations (E-1), (E-2), (E-3) and (E-7) can lead to a term of the form $\mathsf{pfk_2}(R_1,R_2,R_3,R_4,R_5)$.
      - If $f \in \{\mathsf{dec}, \mathsf{fst}, \mathsf{snd}, \mathsf{unblind}\}$, using Lemma A.5, we have a contradiction with the fact that $\mathsf{head}(R'\theta_{i-1}\hat{\sigma}a^0_{i-1}\sigma_L) = \mathsf{pfk_2}$.
      - If $f = \mathsf{pfk_2}$, then $R' = \mathsf{pfk_2}(R_1,R_2,R_3,R_4,R_5)$, with some free R_j for $j \in [\![1,5]\!]$. Since we have that $(R =_E \mathsf{pfk_2}(\mathsf{idp_i}, R_1, R_2, e_i, R_3))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ and $\nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L \approx_s \nu\tilde{\omega}.\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R$, we conclude that:

$$(R =_E \mathsf{pfk_2}(\mathsf{idp_i}, R_1, R_2, \mathsf{e_i}, R_3))\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_R.$$

Thus, for $\sigma \in \{\sigma_L, \sigma_R\}$, we have:

$$R\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{pfk}_2(\mathsf{idp_i}, R_1, R_2, \mathsf{e_i}, R_3)\theta_{i-1}\hat{\sigma}a^0_{i-1}\sigma.$$

Moreover, we also have, according to Equation E-10, and repeating the same development as we just did, that:

$$Q\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{blind}(\mathsf{renc}(\mathsf{e_i}, R_1), R_2)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma.$$

- Let $i \in [\![3, n]\!]$.
  - Since $\phi_{\mathsf{R}}(\mathsf{idp_i}, M)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{ok}$, we have that $P\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma_L$ is an id_i-valid ballot. Using Lemma A.8, we have $P\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \langle P_1, P_2, P_3, P_4\rangle\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ with free terms P_j for $j \in [\![1, 4]\!]$ and $\sigma \in \{\sigma_L, \sigma_R\}$.
  - According to Lemma A.8, we also get that $P_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{penc}(P'_1, P'_2, U)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ for free terms P'_1, P'_2 and free term U or $U =_{AC} \mathsf{pk}(\mathsf{a_p} + U')$ with free U' and $\mathsf{a_p} \in \{\mathsf{a_1}, \mathsf{a_3}\}$. Now, according to Equation E-10 and by repeating the same reasoning as in the previous case in this situation, we can deduce that $R\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{pfk}_2(\mathsf{vk}(\mathsf{id_i}), R_1, R_2, R_3, R_4)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$ with free R_i for $i \in [\![1, 4]\!]$ and $\sigma \in \{\sigma_L, \sigma_R\}$. This equation also provides that $Q\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{blind}(\mathsf{renc}(P_2, R_1), R_2)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$. Now, since $P_2\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{penc}(P'_1, P'_2, U)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$, we can conclude that:

$$Q\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma =_E \mathsf{blind}(\mathsf{penc}(Q_1, Q_2, U), Q_3)\theta_{i-1}\hat{\sigma}^0_{i-1}\sigma$$

with free Q_1, Q_2, Q_3 and free U or $U =_{AC} \mathsf{pk}(\mathsf{a_p} + U')$ with free U' and $\mathsf{a_p} \in \{\mathsf{a_1}, \mathsf{a_3}\}$.

□

A.6 Moving to Final Static Equivalence

We first show that we can add the signature coming from the Receipt generator in the frame.

Lemma A.10. *Using notations from Definition A.1, for $i \in [\![0, n]\!]$, we have:*

$$\nu\tilde{\omega}.\theta_i\hat{\sigma}^0_i\sigma_L \approx_s \nu\tilde{\omega}.\theta_i\hat{\sigma}^0_i\sigma_R.$$

Proof. The case $i = 0$ is proved by Lemma A.3.

Induction Step: Assume now that, for any $i \geq 0$, we have that $\nu\tilde{\omega}.\theta_i\hat{\sigma}^0_i\sigma_L \approx_s \nu\tilde{\omega}.\theta_i\hat{\sigma}^0_i\sigma_R$ holds. We will show that this formula also holds for $i + 1$.

We consider the term $S_{i+1} = \mathsf{sign}(U_{i+1}, \mathsf{id_R})$ with $U_{i+1} = \mathsf{hash}(\Pi_1(M_{i+1}))$ and free M_{i+1}. Since $\mathsf{id_R}$ is restricted and not deducible according to Lemma A.2, we must have that $S_{i+1}\theta_i\hat{\sigma}^0_i\sigma$ does not appear in $\theta_i\hat{\sigma}^0_i\sigma$ for $\sigma \in \{\sigma_L, \sigma_R\}$. We also have that $\{^{\mathsf{vk}(\mathsf{id_R})}/_{\mathsf{idp_R}}\} \in \theta_i$, thus, using Lemma 2.8, we have, with $\theta^s_i = \theta_i \mid \{^{S_{i+1}}/_{sr_{i+1}}\}$:

$$\nu\tilde{\omega}.\theta^s_i\hat{\sigma}^0_{i+1}\sigma_L \approx_s \nu\tilde{\omega}.\theta^s_i\hat{\sigma}^0_{i+1}\sigma_R.$$

Now we consider $R_{i+1} = \mathsf{d}(\mathsf{p}(\mathsf{id_{i+1}}), \mathsf{dec}(\Pi_2(M_{i+1}), \mathsf{a_3}))$ and, according to this definition, we consider several cases:

- Either $i + 1 \in \{1, 2\}$, then, according to Lemma A.2, $\mathsf{p}(\mathsf{id_{i+1}})$ is not deducible for $\theta^s_i\hat{\sigma}^0_i\sigma$ and so is R_{i+1} for $\sigma \in \{\sigma_L, \sigma_R\}$. Moreover, $R_{i+1}\theta^s_i\hat{\sigma}^0_i\sigma$ is neither a subterm of $\theta^s_i\hat{\sigma}^0_i\sigma$ for $\sigma \in \{\sigma_L, \sigma_R\}$, nor deducible from it, then, since d is a destructor in E, we can use Lemma 2.7 to conclude.

- Or $i{+}1 \geq 3$. Using Lemma A.9, we have $\Pi_2(M_{i+1})\theta_i^s\hat{\sigma}_i^0\sigma =_E \mathsf{blind}(\mathsf{penc}(P_1, P_2, U), P_3)\theta_i^s\hat{\sigma}_i^0\sigma$ with free terms P_1, P_2, P_3 and free U or $U =_{AC} \mathsf{pk}(\mathsf{a_p} + U')$ with free U' and $\mathsf{a_p} \in \{\mathsf{a_1}, \mathsf{a_3}\}$ for $\sigma \in \{\sigma_L, \sigma_R\}$. We will consider two different subcases:

  - $U\theta_i^s\hat{\sigma}_i^0\sigma \neq_E \mathsf{pk}(\mathsf{a_3})$ for $\sigma \in \{\sigma_L, \sigma_R\}$. we introduce the following claims:

    Claim 1. $\forall\ x \in \mathsf{dom}(\theta_i^s\hat{\sigma}_i^0\sigma),\ x\theta_i^s\hat{\sigma}_i^0\sigma \neq_E \mathsf{dec}(M, \mathsf{a_3})$.

    Proof. Left to reader's discretion. □

    Claim 2. *If* $\theta_i^s\hat{\sigma}_i^0\sigma \vdash R_{i+1}\theta_i^s\hat{\sigma}_i^0\sigma$ *then* $R_{i+1}\theta_i^s\hat{\sigma}_i^0\sigma \in \mathsf{St}(\theta_i^s\hat{\sigma}_i^0\sigma)$.

    Proof. Left to reader's discretion. □

    Claim 3. $R_{i+1}\theta_i^s\hat{\sigma}_i^0\sigma \notin \mathsf{St}(\theta_i^s\hat{\sigma}_i^0\sigma)$.

    Proof. Left to reader's discretion. □

    According to Claim 2 and 3, we have that $R_{i+1}\theta_i^s\hat{\sigma}_i^0\sigma$ is neither a subterm of $\theta_i^s\hat{\sigma}_i^0\sigma$ nor appearing in it nor deducible from it, for $\sigma \in \{\sigma_L, \sigma_R\}$. We also have that, in normal form, $R_{i+1}\theta_i^s\hat{\sigma}_i^0\sigma =_E \mathsf{d}(\mathsf{p}(\mathsf{id_{i+1}}), \mathsf{dec}(\mathsf{blind}(\mathsf{penc}(P_1, P_2, U), P_3)\theta_i^s\hat{\sigma}_i^0\sigma, \mathsf{a_3}))$. Then, since d is a destructor in E, we can use Lemma 2.7 with previous equivalence to conclude.
  - $U\theta_i^s\hat{\sigma}_i^0\sigma =_E \mathsf{pk}(\mathsf{a_3})$ for one $\sigma \in \{\sigma_L, \sigma_R\}$ then, using that $\theta_i^s\hat{\sigma}_i^0\sigma_L \approx_s \theta_i^s\hat{\sigma}_i^0\sigma_R$, we have $U\theta_i^s\hat{\sigma}_i^0\sigma =_E \mathsf{g_3}\theta_i^s\hat{\sigma}_i^0\sigma$ for $\sigma \in \{\sigma_L, \sigma_R\}$. In that case, we have $R_{i+1}\theta_i^s\hat{\sigma}_i^0\sigma =_E \mathsf{d}(\mathsf{p}(\mathsf{id_{i+1}}), \mathsf{blind}(P_1, P_3))\theta_i^s\hat{\sigma}_i^0\sigma$ with free P_1 and P_3. Thus, we can use Lemma 2.3 with previous equivalence to conclude.

□

Finally, we show that we can add the final results outputted by the Decryption device to the frame provided that the Auditor has endorsed it.

Lemma A.11. *Let us consider* $M_1, \ldots, M_n$ *free terms s.t.* $\forall\ i \in [\![1, n]\!]$, $\phi_{\mathsf{R}}(\mathsf{idp_{i+1}}, M_{i+1})\theta_i\hat{\sigma}_i^0\sigma_L = \mathsf{ok}$ *with* $\mathsf{fv}(M_{i+1}) \subseteq \mathsf{dom}(\theta_i) \cup \mathsf{dom}(\hat{\sigma}_i^0) \setminus \{x_b^i\}$. *We suppose that* $\forall i \in [\![1, n]\!]$, $\nu\tilde{\omega}.\theta_i\hat{\sigma}_i^0\sigma_L \approx_s \nu\tilde{\omega}.\theta_i\hat{\sigma}_i^0\sigma_R$. *We also consider* $U_1, \ldots, U_n$ *and* $W_1, \ldots, W_n$ *be free terms such that:*

- $\mathsf{fv}(U_{k+1}) \subseteq \mathsf{dom}(\theta_n) \cup \{\mathsf{dom}(\hat{\sigma}_n^0)\} \cup \{d_1, \ldots, d_k\}$ *and* $\mathsf{fv}(W_{k+1}) \subseteq \mathsf{dom}(\theta_n) \cup \{\mathsf{dom}(\hat{\sigma}_n^k)\}$,
- $\phi_{\mathsf{A}}(\mathsf{hash}(\langle U_1, \ldots, U_n\rangle), hb_r^1, \ldots, hb_r^n, W_1, \ldots, W_n)\theta_n\hat{\sigma}_n^n\sigma_L = \mathsf{ok}$, *with the corresponding notation:* $hb_r^i = \langle \mathsf{idp_i}, \mathsf{hash}(\Pi_1(M_i))\rangle$.

Then, we have, for $\sigma \in \{\sigma_L, \sigma_R\}$, $\forall 1 \leq i \leq n$*:*

$$U_i\theta_n\hat{\sigma}_n^n\sigma =_E \mathsf{e_i}\theta_n\hat{\sigma}_n^n\sigma,\ \textit{for } i = 1, 2.$$

Moreover, there exist free terms U_1', U_2' *and free term* U_3' *or* $U_3' = \mathsf{pk}(\mathsf{a_k} + U)$ *with free term* U*:*

$$U_i\theta_n\hat{\sigma}_n^n\sigma =_E \mathsf{penc}(U_1', U_2', U_3')\theta_n\hat{\sigma}_n^n\sigma,\ \textit{for } i \in [\![3, n]\!].$$

Proof. Since $\nu\tilde{\omega}.\theta_n\hat{\sigma}_n^0\sigma_L \approx_s \nu\tilde{\omega}.\theta_n\hat{\sigma}_n^0\sigma_R$, then $\nu\tilde{\omega}.\theta_n\hat{\sigma}_n^n\sigma_L \approx_s \nu\tilde{\omega}.\theta_n\hat{\sigma}_n^n\sigma_R$ and according to the fact that ϕ_{A} is a public test, we also know that, for $\sigma \in \{\sigma_L, \sigma_R\}$:

$$\phi_{\mathsf{A}}(\mathsf{hash}(\langle U_1, \ldots, U_n\rangle), hb_r^1, \ldots, hb_r^n, W_1, \ldots, W_n)\theta_n\hat{\sigma}_n^n\sigma = \mathsf{ok}.$$

This implies that, according to the definition of ϕ_{A}, that:

$$\mathsf{hash}(\langle U_1, \ldots, U_n\rangle)\theta_n\hat{\sigma}_n^n\sigma =_E \mathsf{hash}(\langle \Pi_2(W_1), \ldots, \Pi_2(W_n)\rangle)\theta_n\hat{\sigma}_n^n\sigma$$
$$\mathsf{hash}(\Pi_1(M_i))\theta_n\hat{\sigma}_n^n\sigma =_E \mathsf{hash}(W_i)\theta_n\hat{\sigma}_n^n\sigma, \text{ for } i \in [\![1, n]\!]$$

From these two equations, we can deduce that $\forall i \in [\![1, n]\!]$, we have:

$$U_i\theta_n\hat{\sigma}_n^n\sigma =_E \Pi_2(\Pi_1(M_i))\theta_n\hat{\sigma}_n^n\sigma.$$

According to the construction of θ_i and $\hat{\sigma}_i^j$, we have that, $\forall i \in [\![0, n]\!]$, $\theta_i = \theta_n \mid_{\mathsf{dom}(\theta_i)}$ and $\hat{\sigma}_i^0 = \hat{\sigma}_n^0 \mid_{\mathsf{dom}(\hat{\sigma}_i^0)}$. In particular, we also have that, $\forall i \in [\![1, n]\!]$, $\mathsf{fv}(M_i) \subseteq \mathsf{dom}(\theta_{i-1})$ and $\mathsf{fv}(M_i\theta_{i-1}) \subseteq \mathsf{dom}(\hat{\sigma}_{i-1}^0)$. Thus, we have:

$$\forall i \in [\![1, n]\!],\ M_i\theta_n\hat{\sigma}_n^0\sigma =_E M_i\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma.$$

But, according to Lemma A.9, we know that $\Pi_1(M_i)\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma$ is an id_i-valid ballot. Thus, using Lemma A.7 for $i \in \{1, 2\}$ and Lemma A.8 for $i \in [\![3, n]\!]$ we have that:

$$\Pi_2(\Pi_1(M_i))\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma =_E \begin{cases} \mathsf{e_i}\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma \text{ for } i \in \{1, 2\} \\ \mathsf{penc}(N, P, Q)\theta_{i-1}\hat{\sigma}_{i-1}^0\sigma, \text{ for } i \in [\![3, n]\!], \end{cases}$$

with N and P free terms and Q a free term or $Q = \mathsf{pk}(\mathsf{a_k} + Q')$ with Q' a free term and such that $\mathsf{fv}(N) \cup \mathsf{fv}(P) \cup \mathsf{fv}(Q) \subseteq \mathsf{fv}(M_i)$. According to this and the fact that $\mathsf{fv}(M_i\theta_n) \subseteq \mathsf{dom}(\hat{\sigma}_n^0)$ and that, for $i \in [\![0, n]\!]$, $\hat{\sigma}_n^i = \hat{\sigma}_n^n \mid_{\mathsf{dom}(\hat{\sigma}_n^i)}$, we have that the previous result holds for $\hat{\sigma}_n^n$ instead of $\hat{\sigma}_n^0$ and we conclude:

$$U_i\theta_n\hat{\sigma}_n^n\sigma =_E \begin{cases} \mathsf{e_i}\theta_n\hat{\sigma}_n^n\sigma \text{ for } i \in \{1, 2\} \\ \mathsf{penc}(N, P, Q)\theta_n\hat{\sigma}_n^n\sigma, \text{ for } i \in [\![3, n]\!]. \end{cases}$$

□

Appendix B

Long Abstract - Résumé de la thèse

This chapter is a detailed summary, in French, of this thesis. It does not contain any new information that has not been already mentioned in the previous chapters and is designed for those who are not familiar with English.

B.1 Introduction

Depuis plusieurs années, l'informatique a pris de l'importance dans de nombreux domaines, en particulier celui des communications. Ordinateurs, téléphones portables, tablettes, etc., tous ces appareils sont devenus incontournables. Et s'ils nous permettent d'effectuer des opérations bancaires ou administratives en ligne pour nous faciliter la vie, ils se voient également confier des informations censées rester secrètes telles que des numéros de compte, de cartes de crédit ou même détails d'ordre privé. Ces informations peuvent se révéler dangereuses si elles tombent entre de mauvaises mains. De plus, même si cette technologie est bien connue du public, ses utilisateurs ne sont pas forcément conscients des risques encourus, comme, par exemple, le vol d'identité (voir l'étude du CREDOC[4] et les recommandations du CIFAS[5]), qui se développe grâce aux informations privées souvent peu, voire pas du tout, protégées une fois en ligne.

L'existence de ces menaces impliquent donc de devoir garantir la sécurité des informations échangées sur les différents réseaux. Dans ce but, il existe des *protocoles cryptographiques* (ou *protocoles de sécurité*). Ce sont des programmes qui assurent la sécurité des communications (c.-à-d. le secret d'une information privée), et font souvent appel à la cryptographie. Bien entendu, il ne suffit pas de dire qu'un protocole est sûr, il faut s'en assurer, et le vérifier, avant son utilisation. En effet, si, après avoir mis un protocole en fonction, on découvre que celui-ci possède une faille exploitable, les conséquences peuvent être importantes, selon la valeur des informations qu'il était censé protéger.

B.1.1 Protocoles cryptographiques

Dans le monde physique, échanger de manière sécurisée des données entre deux ou plusieurs participants exigerait des enveloppes scellées, des boîtes cadenassées, voire des pièces insonorisées. Dans le monde numérique, on utilise des protocoles cryptographiques.

[4] Une présentation de ce rapport est disponible à l'adresse suivante : http://protegezvotreidentite.files.wordpress.com/2009/11/credoc-usurpation-didentite-en-france.pdf.

[5] Chiffres disponibles à l'adresse : http://www.cifas.org.uk/identity_fraud.

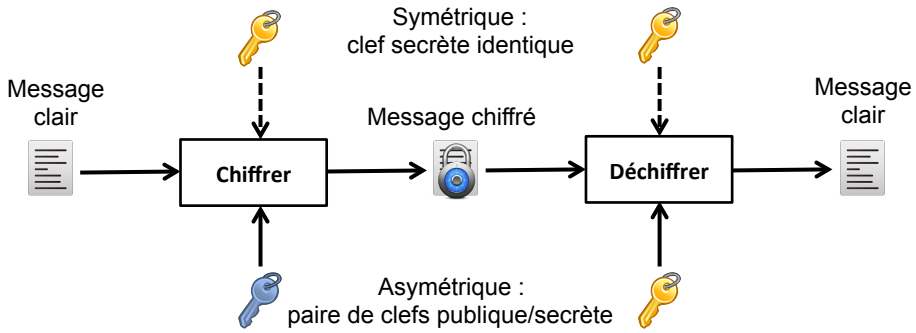


Figure B.1: Schémas de chiffrement symétrique et asymétrique.

La cryptographie existe depuis plusieurs millénaires et a beaucoup évolué depuis. Il existe bon nombre d'algorithmes de chiffrements, que l'on peut classer en deux catégories : symétrique ou asymétrique (voir Figure B.1). Les lecteurs curieux pourront se référer aux livres de B. Schneier [Sch95] et D. Kahn [Kah96] pour davantage de détails sur les schémas existants et une histoire détaillée de la cryptographie.

Depuis des années, les cryptographes cherchent à créer l'algorithme cryptographique parfait (c.-à-d. incassable). Mais, même si un tel algorithme existait, cela ne garantirait pas la sécurité des communications pour autant. La cryptographie n'est qu'un outil qui se doit d'être utilisé convenablement. Un *protocole* est une sorte de manuel, qui décrit la façon dont les algorithmes cryptographiques doivent être utilisés afin d'accomplir une tâche spécifique. Dans le domaine des communications, un protocole définira l'ensemble des opérations permettant l'authentification mutuelle entre deux utilisateurs ou l'établissement d'un canal sécurisé, etc. Il y a autant de protocoles que de tâches à accomplir et parfois plusieurs protocoles pour un même objectif (pour des raisons d'efficacité, d'optimisation, de charge serveur, etc.). Cette thèse portera sur deux familles spécifiques de protocoles cryptographiques : le vote électronique et les API.

B.1.2 Vote électronique

Voter n'est pas anodin et les élections peuvent avoir des impacts politiques et économiques. Par conséquent, il est nécessaire de garantir leur bon déroulement. Le vote-papier, qui utilise des urnes transparentes, des bulletins et des enveloppes, fournit de fortes garanties aux votants mais prsente quelques désavantages, notamment un décompte plutôt fastidieux et la mobilisation de moyens humains importants. La question est alors de savoir si la technologie actuelle peut permettre de faire mieux.

Protocoles de vote électronique

Les premières tentatives datent des années 1900, avec les machines à leviers utilisées aux États-Unis [JS12]. La mécanique remplaçait alors le papier pour accélérer les processus de vote et de décompte. Malheureusement, ces machines étaient facilement modifiables et de nombreux cas de fraudes ont été recensés. La technologie aidant, la mécanique est remplacée par l'électronique et les premières machines à voter électroniques apparaissent dès 1996 aux États-Unis et sont depuis très largement utilisées. En 2005, l'Estonie est le premier pays à organiser des élections nationales où les votants utilisent Internet pour soumettre leur vote.

Il existe deux variantes du vote électronique. La première concerne tous les protocoles qui sont utilisés directement au bureau de vote, tels que les machines à voter, et sont directement

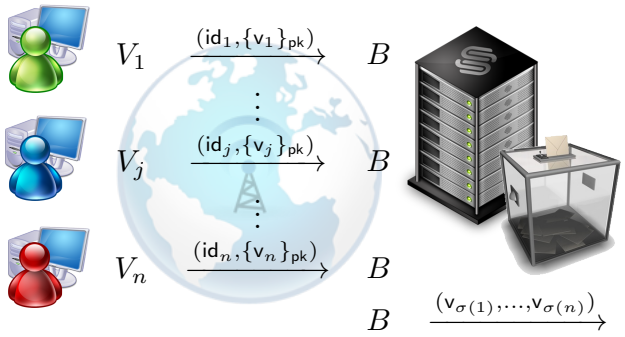


Figure B.2: Échanges de messages durant l'exécution du protocole $\mathsf{E}_{\mathsf{toy}}$.

surpervisés par des représentants gouvernementaux ou des autorités locales indépendantes. La seconde permet aux votants d'exprimer leur choix d'où ils le souhaitent, du moment qu'ils ont accès à Internet et un ordinateur. Cette thèse est consacrée particulièrement à cette seconde catégorie de protocoles. L'exemple $\mathsf{E}_{\mathsf{toy}}$ décrit dans la Figure B.2 illustre le fonctionnement de ces protocoles. La phase de vote consiste, pour chaque votant, à chiffrer son vote avec la clef publique de l'élection puis à soumettre son vote à l'urne virtuelle grâce à son navigateur. Une fois le vote clos, l'urne (un programme s'exécutant sur un serveur), déchiffre les bulletins avec la clef secrète et révèle publiquement le résultat, qui est une permutation de tous les votes déchiffrés. À noter que, même si ce protocole paraît simpliste, le principe d'avoir une *boîte noire* collectant les votes, les déchiffrant, puis publiant le résultat de l'élection, est utilisé dans certains systèmes actuels. Bien entendu, de tels protocoles n'offrent que peu de garanties de sécurité puisqu'il faut faire entièrement confiance à l'urne. C'est pour affaiblir cette hypothèse de confiance que de nombreux protocoles de vote électronique font appel à plusieurs entités avec des rôles spécifiques (collecter les votes, déchiffrer, etc.).

Propriétés de sécurité

Bien que théoriquement plus pratique que le vote-papier, le vote électronique se doit cependant de garantir a minima les mêmes propriétés de sécurité. Durant l'analyse d'un protocole, une des difficultés est de définir proprement les propriétés que l'on veut démontrer.

L'une des plus importantes est l'*anonymat du vote* : l'impossibilité, pour une personne malveillante, de découvrir le vote d'une personne durant une élection. Il existe une définition formelle de cette propriété [DKR09]. Intuitivement, on considère deux exécutions du protocole dans lesquelles deux votants honnêtes échangent leurs votes. Si on ne peut pas avoir de différences entre les deux exécutions, alors le protocole vérifie la propriété. D'autres définitions existent, dans des modèles différents, et ont été utilisées [KTV11, BPW12] dans l'analyse de protocoles. Dans cette thèse, on retiendra la définition de [DKR09].

Il existe d'autres propriétés, telles que la *résistance à la coercition* [ALBD04, DKR09] ou la *vérifiabilité*, individuelle ou universelle [KRS10]. Ces propriétés et d'autres encore sont présentées et définies dans la littérature [KTV11, FOO92, JLY98].

Attaques

Si les protocoles de vote électronique utilisent souvent beaucoup de primitives cryptographiques, cela n'implique pas qu'ils sont sûrs pour autant. L'exemple fourni précédemment, $\mathsf{E}_{\mathsf{toy}}$, est

attaquable. Pour le démontrer, nous prendrons trois votants, V_{A} et V_{B}, Alice et Bob, deux votants honnêtes, et V_{C}, Charlie, une personne malhonnête dont le but est de découvrir comment les autres ont voté. On supposera également que Charlie contrôle le réseau, une hypothèse courante lors de l'analyse de protocoles.

L'attaque se déroule de la manière suivante. Pendant la phase de vote, Alice et Bob agissent normalement et soumettent leurs bulletins $(id_{\mathsf{A}}, \{v_{\mathsf{A}}\}_{\mathsf{pk}})$ et $(id_{\mathsf{B}}, \{v_{\mathsf{B}}\}_{\mathsf{pk}})$. Charlie, qui a accès aux communications voit ces bulletins sans être capable de les déchiffrer, puisqu'il ne possède pas la clef secrète. Mais il peut récupérer le vote chiffré d'Alice (soit $\{v_{\mathsf{A}}\}_{\mathsf{pk}}$) et le soumettre comme si c'était le sien. Finalement, durant la phase de décompte, l'urne déchiffrera et publiera le résultat $\{v_{\mathsf{A}}, v_{\mathsf{A}}, v_{\mathsf{B}}\}$ permettant ainsi à Charlie de savoir comment Alice a voté.

Cette simple attaque montre que la cryptographie ne suffit pas à assurer la sécurité d'un protocole. Toutefois, plus ceux-ci sont complexes, plus il est difficile de trouver une faille au premier coup d'œil. C'est pour cela qu'il est nécessaire d'analyser ces protocoles afin de s'assurer qu'ils vérifient les propriétés recherchées.

Protocoles existants

Le vote électronique suscite beaucoup d'intérêt depuis plusieurs années et il existe dans la littérature de nombreux protocoles qui fournissent souvent différents niveaux de sécurité. On pourra mentionner FOO [FOO92], Civitas [JCJ05, CCM08], Helios [Adi08], Belenios [CGGI13], Caveat Coercitor [GRBR13], qui sont des protocoles de vote à distance (par Internet), mais aussi Prêt-à-Voter [CRS05], Three-Ballots [Riv06], Scantegrity [SFCC11], qui sont des protocoles directement mis en place aux bureaux de vote.

B.1.3 API

Avec l'évolution des technologies, les systèmes embarqués se sont multipliés. On peut les retrouver dans nos téléphones ou nos véhicules et ils sont souvent accompagnés de composants hardwares sécurisés qui vont effectuer les opérations cryptographiques ou assurer un stockage sûr pour des informations secrètes. Néanmoins, ces composants ne garantissent pas à eux seuls la sécurité de ces systèmes. Il est nécessaire de définir des interfaces permettant d'accéder de manière sécurisée à ces composants. C'est le rôle des API (Interfaces de Programmation).

API de sécurité

Une *API de sécurité* est une liste de commandes, conçue pour permettre à une machine-hôte non nécessairement digne de confiance, d'accéder à un composant fiable d'une manière entièrement sécurisée. C'est une passerelle entre deux mondes avec un niveau de confiance différent. Tout comme un protocole, une API a pour but d'assurer des propriétés de sécurité, quel que soit l'ordre dans lequel ses commandes sont exécutées comme, par exemple, éviter la fuite d'informations secrètes.

Pour illustrer cela, on considère l'exemple d'un composant sécurisé conçu pour stocker des clefs utilisées pour du chiffrement et du déchiffrement. La mémoire du composant est représentée par un tableau dont chaque entrée est étiquetée d'un identifiant et contient une clef. Les identifiants permettent aux utilisateurs de manipuler les clefs sans connaître leurs valeurs. Par exemple, lorsqu'une clef est reçue (chiffrée) par le composant, la valeur de la clef est stockée et l'utilisateur ne reçoit qu'un identifiant correspondant à cette nouvelle clef.

On considère l'interface A_{toy} qui permet à un utilisateur d'utiliser deux commandes : encrypt et decrypt. Comme le montre la Figure B.3, l'appel de la fonction de chiffrement (encrypt) prend,

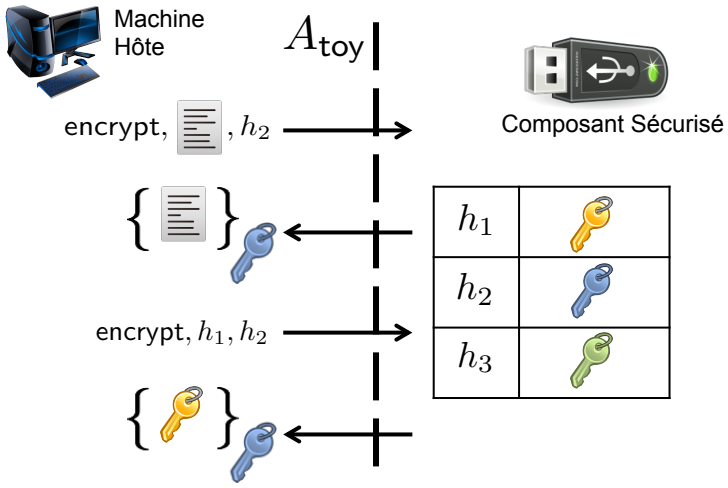


Figure B.3: La commande encrypt de A_{toy}, exemple d'API.

comme arguments, ce qu'il faut chiffrer (des données ou un identifiant) et l'identifiant pointant vers la clef qu'il faut utiliser pour chiffrer. La commande de déchiffrement (decrypt) fonctione de manière similaire avec un message chiffré et un identifiant pointant vers la clef à utiliser. En connectant le composant sécurisé à une machine-hôte, un utilisateur peut se servir d'une clef stockée dans le composant pour effectuer des chiffrements et des déchiffrements.

Les API sont utilisées dans de nombreux domaines et il existe de nombreux exemples, que ce soit académiques, open-source ou propriétaires. Comme, notamment, l'IBM CCA (Common Cryptographic Architecture) [IBM06], Trusted Platform Modules [Rya09], ou le standard de cryptographie à clefs publiques PKCS#11 [Inc04].

Attaques

Tout comme les protocoles, les API peuvent être vulnérables aux attaques, et peut-être même plus puisque, dans le cas des API, il n'y a pas de séquence prédéfinie de messages à échanger. Aussi, un utilisateur, malhonnête ou non, peut donc éxecuter autant de commandes qu'il le souhaite, dans n'importe quel ordre.

L'exemple A_{toy} est problématique. Il n'y a aucune restriction sur ce qui peut être chiffré ou déchiffré. En particulier, il est facile de construire une attaque qui conduit à la fuite d'une clef. Un utilisateur peut simplement demander à chiffrer une clef en utilisant son identifiant (encrypt, h_1, h_2) et obtenir le chiffrement C de la clef stockée dans h_1. Il demande ensuite le déchiffrement de C (decrypt, C, h_2) et récupère ainsi la clef en clair.

Plusieurs attaques ont été découvertes sur des API déjà déployées, principalement sur des implémentations de PKCS#11 [Clu03, DKS08, BCFS10] car il est le plus utilisé et étudié. Des failles ont également été découvertes sur l'API d'IBM [BA01]. Une preuve que les API également doivent être analysées avant d'être déployées.

B.1.4 Analyse de protocoles cryptographiques

Comme nous l'avons vu précédemment, la vérification des protocoles est cruciale. Afin de réaliser de telles analyses, la première étape consiste à représenter le système et son environnement.

Modèles Existants

Il existe deux grandes familles de modèles : *calculatoire* et *symbolique*. Chacune des deux approches possède ses avantages et ses inconvénients. Cette thèse est davantage consacrée à l'approche symbolique. Dans ce modèle, les messages sont vus comme des termes et les primitives cryptographiques sont représentées par des symboles de fonctions. On peut alors abstraire l'ensemble du protocole et les capacités de l'intrus sont définies par un ensemble de règles. Il existe un nombre important de modèles symboliques que ce soit avec des *systèmes de contraintes* [MS01, CLS03] ou des *clauses de Horn* [Wei99, Bla01, CLC03]. Une grande partie de cette thèse est basée sur un autre modèle symbolique, basé sur le *pi-calcul appliqué* [AF01].

Résultats existants

L'analyse des protocoles cryptographiques dans le modèle symbolique est un problème difficile et vérifier des propriétés de sécurité se révèle rapidement indécidable [DLMS99]. Les résultats de décidabilité posent souvent des limitations, notamment sur le nombre de session [RT01]. Il existe des résultats sur les *propriétés de traces* qui sont comparables à des problèmes d'accessibilité : est-il possible de trouver une exécution (d'un protocole) qui révèle un secret ? Plusieurs outils permettent d'étudier automatiquement ces propriétés comme ProVerif [Bla01], AVISPA [Vig06] ou Scyther [Cre08]. Dans cette thèse, les propriétés de sécurité étudiées s'expriment en termes d'équivalence, comme, par exemple, l'anonymat du vote qui est défini comme l'absence de différences entre deux exécutions (plus formellement, il s'agit d'équivalence observationelle). Il existe plusieurs résultats de décidabilité [Hüt02, Bau05, CD09, CR05] pour ces propriétés. Plusieurs outils permettent également, dans certaines limites, d'analyser automatiquement ces propriétés, comme SPEC [TD10], AKiSs [cC01] or APTE [Che14]. ProVerif a également été étendu afin de pouvoir démontrer certaines équivalences [BAF08]. Concernant les API, il existe plusieurs modèles souvent créés spécifiquement en fonction de l'API en se basant sur des clauses de Horn ou des règles de réécriture. Des outils comme StatVerif [ARR11] ou Tamarin [MSCB13] émergent petit à petit pour automatiser ces analyses.

Limitations

Malgré tous les résultats existants et les progrès constants dans le domaine de la vérification des protocoles cryptographiques, il y a encore des limitations, en particulier pour le vote électronique. En effet, le vote électronique utilise un nombre conséquent de primitives cryptographiques différentes (chiffrement homorphique, déchiffrement distribué, preuves à divulgation de connaissance nulle, etc.), souvent utilisées de concert. L'utilisation simultanée de toutes ces fonctions rend la vérification automatique avec les outils existants impossible et l'étude de ces protocoles doit souvent être faite à la main. L'analyse des API est également complexe. Là où les protocoles sont des échanges de messages bien définis, les API sont une liste de fonctions qu'il est possible d'appeler dans n'importe quel ordre. Qui plus est, ces commandes utilisent souvent des paramètres communs rendant l'analyse modulaire quasiment impossible. De plus, l'absence de formalisme des propriétés de sécurité et l'utilisation d'états globaux rendent également l'analyse difficile. Tout cela implique de développer des approches spécifiques pour analyser les API.

B.1.5 Contributions et publications

Cette thèse est divisée en trois parties distinctes. La première partie (Section B.2), présente des études formelles de protocoles de vote électroniques basées sur l'analyse symbolique et le

pi-calcul appliqué. La deuxième partie (Section B.3), est centrée sur la définition d'une API de sécurité pour la gestion de clefs symétriques incluant une fonctionnalité de révocation. Enfin, la troisième partie (Section B.4), décrit une nouvelle approche pour l'automatisation des preuves d'anonymat du vote pour les protocoles de vote électronique, qui utilise les systèmes de types.

Les contributions présentées dans les deux premières parties de cette thèse ont fait l'objet de publications dans les conférences suivantes: POST'12 [CW12] pour l'analyse du protocole norvégien, Vote-ID'13 [ACW13] pour l'étude du protocole du CNRS et CCS'12 [CSW12] pour la présentation d'un nouveau design d'API. Elles représentent des travaux en collaboration avec Mathilde Arnaud d'une part, et Graham Steel d'autre part. La dernière partie n'a pas fait l'objet de publication pour le moment mais est également un travail issu d'une collaboration avec Matteo Maffei, Fabienne Eigner et Steve Kremer.

B.2 Analyse formelle de protocoles de vote électronique

Cette partie se consacre à l'étude de deux protocoles de vote utilisés dans deux contextes différents. Le premier est un système permettant à ses utilisateurs de voter par internet, le second a été créé afin de servir lors délections de comité, avec un nombre restreint de votants tous présents dans la même pièce. Nous étudions les propriétés de sécurité de ces deux protocoles à l'aide du pi-calcul appliqué [AF01] dont la syntaxe est détaillée dans le Chapitre 1.

B.2.1 Protocole de vote norvégien

Testé en septembre 20011 et 2013 pour des élections municipales en Norvège [Gov], ce protocole a été mis en place dans dix municipalités et a été utilisé par environ 28 000 votants en 2011 et 70 000 en 2013. Dans cette thèse, on propose une modélisation de ce protocole en pi-calcul appliqué ainsi qu'une étude de l'anonymat du vote sous plusieurs scénarios de corruption.

Le protocole de vote norvégien [Gjø10] est constitué de quatre entités : une urne (B), un générateur de reçus (R), un service de déchiffrement (D) et un auditeur (A). Chaque votant (V) peut se connecter grâce à un ordinateur (P) afin de soumettre son vote. Le protocole en lui-même peut se découper en trois phases : initilisation, soumission et décompte.

Ces phases étant purement descriptives, nous invitons le lecteur à se référer à la Section 2.1 de cette thèse pour voir le fonctionnement du protocole en détail. D'un point de vue informel, B va recueillir le vote du votant et R va fournir une preuve au votant que son vote a bel et bien été pris en compte. La spécificité de ce protocole est qu'il fournit deux confirmations au votant permettant de s'assurer qu'il n'y a pas de problème. La première, que son vote a été pris en compte, via un retour sur son ordinateur. La seconde, que c'est le bon vote qui a été enregistré, via un code envoyé par SMS, par R, à comparer avec une liste reçue avant l'élection par courrier. Durant la phase de décompte, A vérifie qu'il n'y a pas d'irrégularités entre les autres entités (B, R et D) et autorise (ou non) le déchiffrement et la publication des résultats par D.

Afin d'étudier ce protocole, ce dernier est modélisé à l'aide du pi-calcul appliqué. Les différentes modélisations peuvent être trouvées dans la section 2.2. On y trouvera l'ensemble des primitives cryptographiques et leurs propriétés (théorie équationnelle) ainsi que la modélisation des entités du protocole en pi-calcul.

Anonymat du vote

Cette thèse présente plusieurs résultats sur l'anonymat du vote dans le protocole norvégien. On énoncera ici des versions simplifiées des différents théorèmes évoqués dans la section 2.3.

Théorème B.1. *On considère le scénario où toutes les entités administratives, sauf B, sont honnêtes et où l'ensemble des votants, sauf deux, sont corrompus. Dans ce scénario, le protocole satisfait toujours la propriété d'anonymat du vote.*

De ce théorème, en découle un autre, qui est le suivant :

Théorème B.2. *On considère le scénario où toutes les entités administratives sont honnêtes et où l'ensemble des votants, sauf deux, sont corrompus. Dans ce scénario, le protocole satisfait toujours la propriété d'anonymat du vote.*

Avec une étude sur un modèle simplifié et à l'aide de ProVerif, on réalise une étude du cas où R ou A est corrompu. Pour finir, on démontre assez facilement que dans les cas où D est corrompu, ainsi que les cas où B et R, ou B et A, sont corrompus, alors il est possible de trouver une attaque contre l'anonymat du vote.

B.2.2 Protocole de vote en réunion du CNRS

Conçu par la section 07 du CNRS, ce système de vote utilise des boîtiers électronique dans le cadre d'un vote en réunion (comité, conseil d'administration, etc.). Ce protocole se distingue par son absence de cryptographie afin qu'il puisse être compréhensible par le plus grand nombre et qu'il soit ainsi plus facile de lui faire confiance contrairement à d'autres solutions du type "boîte noire" dans lesquelles l'utilisateur doit avoir confiance sans avoir de réelles preuves que cela fonctionne correctement.

Cette thèse présente une analyse complète du système de vote proposé en présentant trois variantes de celui-ci avant de les formaliser en pi-calcul appliqué. A l'aide de ces modèles, nous étudions les propriétés d'anonymat et d'exactitude du vote sous différents scénarios de corruption. La propriété d'exactitude du vote fait référence au fait que le résultat de l'élection reflète bien la volonté des votants honnêtes et que leurs votes apparaissent correctement dans le décompte final. Les différentes variantes sont présentées dans la section 3.2, mais se basent sur la même infrastructure avec une urne connectée à des boîtiers, un pour chaque votant, ainsi qu'un écran où les résultats sont affichés. L'assemblée possède également un membre qui jouera le rôle de l'assesseur (par ex. le président ou un membre désigné pour la séance).

Propriétés de sécurité

Après avoir modélisé (section 3.3) les différentes versions du protocole en pi-calcul appliqué, on peut démontrer le théorème suivant concernant la propriété d'anonymat du vote (section 3.4) :

Théorème B.3. *Dans le cas où les entités du protocole (urne, assesseur et écran) sont honnêtes et tous les votants, sauf deux, sont malhonnêtes, alors le protocole du CNRS (dans deux de ses variantes) satisfait à la propriété d'anonymat du vote.*

Pour l'exactitude du vote, on propose d'abord une définition de la propriété. (La définition formelle se trouve dans la section 3.5.)

Définition B.1. *Dans le cas où il y a n votants honnêtes parmi les m votants participant à l'élection, on dit que le protocole satisfait l'exactitude du vote si le résultat publié contient l'ensemble des n votes des votants honnêtes.*

On démontre alors le théorème suivant :

Théorème B.4. *Dans le cas où les entités du protocole, sauf l'urne, sont honnêtes, le protocole du CNRS (dans deux de ses variantes) satisfait à la propriété de l'exactitude du vote.*

B.3 Analyse formelle d'API

Cette partie est dédiée à l'élaboration et l'étude d'une API de sécurité pour la gestion de clefs symétriques stockées sur des composants sécurisés. L'intérêt de cette interface est qu'elle propose une fonction de révocation, une fonctionnalité souvent difficile à mettre en place tout en garantissant un niveau de sécurité optimal [EG02, Kar09, MM11].

Le design de l'API se base sur une hiérarchie de niveaux entre les différentes clefs qui forme un ordre partiel. Chaque clef est associée à un niveau. De cette manière, si deux clefs k_1 et k_2 avec des niveaux respectifs l_1 et l_2 sont telles que $l_1 < l_2$, alors il est possible de chiffrer k_1 avec k_2. Dans tous les autres cas, l'opération de chiffrement est impossible. En d'autres termes, il n'est possible de chiffrer qu'avec une clef de niveau strictement supérieur. On associe également à chaque clef une date de validité (liée à son niveau) de sorte qu'une fois la date dépassée, il n'est normalement plus possible d'utiliser la clef pour chiffrer ou déchiffrer. La description complète de l'API et de ses commandes est prsentée dans la section 4.2.

En plus du design de l'API elle-même, cette thèse propose une analyse formelle de celle-ci afin d'en énoncer et démontrer les propriétés de sécurité. Cette analyse formelle se base sur l'utilisation d'états globaux et de règles de transitions. De cette analyse, on distingue trois résultats majeurs. (Ces résultats sont présentés dans la section 4.4.)

Théorème B.5. *Si une clef est valide (c.-à-d. sa date de validité n'a pas expiré) et telle que son niveau n'est pas inférieur au niveau d'une clef perdue (connue de l'attaquant) alors cette clef est non déductible par l'attaquant.*

La hiérarchie des clefs permet également d'empêcher l'intrus d'accéder depuis une clef corrompue à d'autres clefs d'un niveau supérieur ou équivalent. En combinaison avec la durée de validité de chaque clef, on obtient le résultat suivant :

Théorème B.6. *Si le système est compromis à un niveau l et que l'intrus ne parvient pas à corrompre de clefs d'un niveau strictement supérieur à l, alors, au bout d'un certain temps, le système ne sera plus compromis que jusqu'au niveau $l-1$. (i.e. on gagne un niveau après que la dernière clef de niveau l corrompue ait expirée.)*

Enfin, le dernier résultat concerne la révocation. En effet, plutôt que d'attendre que le système se répare de lui-même, on peut procéder à une révocation et ainsi sécuriser immédiatement le système en révoquant la clef corrompue de l'ensemble des appareils.

B.4 Automatisation de preuves avec les systèmes de types

La dernière partie de cette thèse est consacrée à une nouvelle approche destinée à automatiser les preuves formelles de sécurité sur les protocoles de vote électronique. Comme on l'a vu dans l'introduction et dans cette thèse, ces preuves se révèlent souvent particulièrement délicates à effectuer à la main et gagneraient à être automatisées comme c'est déjà le cas pour certains protocoles cryptographiques.

Ces travaux se basent sur un système de types présenté récemment dans [BFG+14], RF*, qui permet la vérification de propriétés basées sur des équivalences, comme l'anonymat du vote. Cette thèse fournit l'application de ce système de types à un protocole de vote connu, Helios [Adi08], pour ainsi vérifier automatiquement - sur une version simplifiée cependant - la propriété d'anonymat du vote. On fournit donc une modélisation de ce protocole dans le langage de RF*, mais on pose également des bases plus générales afin de pouvoir réappliquer cette méthodologie à d'autres systèmes de vote.

B.5 Conclusion

Cette thèse s'est principalement intéressée à deux familles avancées de protocoles de sécurité : le vote électronique et les API. En particulier, nous avons analysé des protocoles de vote existants et présenté un nouveau design d'API de sécurité tout en fournissant une preuve formelle des propriétés de ce dernier. Enfin, nous avons également posé les premiers jalons afin d'automatiser la vérification des protocoles de vote électronique en utilisant les systèmes de types.

Bibliography

[ABF07] Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just Fast Keying in the Pi Calculus. *ACM Transactions on Information and System Security (TISSEC)*, 2007.

[ACW13] Mathilde Arnaud, Véronique Cortier, and Cyrille Wiedling. Analysis of an Electronic Boardroom Voting System. In *4th International Conference on E-Voting and Identify (Vote-ID)*, 2013.

[Adi08] Ben Adida. Helios: Web-based Open-Audit Voting. In *17th USENIX Security Symposium*, 2008.

[AF01] Martin Abadi and Cédric Fournet. Mobile Values, New names, and Secure Communication. In *28th ACM Symposium on Principles of Programming Languages (POPL)*, 2001.

[Aka13] Peter E. Yee Akayla. RFC 6818: Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. http://tools.ietf.org/html/rfc6818, 2013.

[ALBD04] Riza Aditya, Byoungcheon Lee, Colin Boyd, and Ed Dawson. An Efficient Mixnet-Based Voting Scheme Providing Receipt-Freeness. In *First International Conference of Trust and Privacy in Digital Busines*, 2004.

[AR00] Martín Abadi and Phillip Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). In *First International Conference of Theoretical Computer Science (IFIP)*, 2000.

[ARR11] Myrto Arapinis, Eike Ritter, and Mark Dermot Ryan. StatVerif: Verification of Stateful Processes. In *24th IEEE Computer Security Foundations Symposium (CSF)*, 2011.

[BA01] Mike Bond and Ross Anderson. API-Level Attacks on Embedded Systems. *IEEE Computer Magazine*, 2001.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated Verification of Selected Equivalences for Security Protocols. *J. Log. Algebr. Program.*, 2008.

[Bau05] Mathieu Baudet. Deciding Security of Protocols Against Off-line Guessing Attacks. In *ACM Conference on Computer and Communications Security (CCS)*, 2005.

[BC08] Bruno Blanchet and Avik Chaudhuri. Automated Formal Analysis of a Protocol for Secure File Sharing on Untrusted Storage. In *29th IEEE Symposium on Security and Privacy (S&P)*, 2008.

[BCFS10] Matteo Bortolozzo, Matteo Centenaro, Riccardo Focardi, and Graham Steel. Attacking and Fixing PKCS#11 Security Tokens. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[BCP+11] David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. Adapting Helios for Provable Ballot Secrecy. In *16th European Symposium on Research in Computer Security (ESORICS)*, 2011.

[BDK07] Michael Backes, Markus Dürmuth, and Ralf Küsters. On Simulatability Soundness and Mapping Soundness of Symbolic Cryptography. In *27th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2007.

[Ben06] Josh Benaloh. Simple verifiable elections. In *USENIX/ACCURATE Electronic Voting Technology Workshop*, 2006.

[BFG+14] Gilles Barthe, Cédric Fournet, Benjamin Grégoire, Pierre-Yves Strub, Nikhil Swamy, and Santiago Zanella Béguelin. Probabilistic Relational Verification for Cryptographic Implementations. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2014.

[BGHB11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-Aided Security Proofs for the Working Cryptographer. In *31st Annual Cryptology Conference on Advances in Cryptology (CRYPTO)*, 2011.

[Bla01] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW)*, 2001.

[Bla05] Bruno Blanchet. An Automatic Security Protocol Verifier based on Resolution Theorem Proving (invited tutorial). In *20th International Conference on Automated Deduction (CADE)*, 2005.

[Bla08] Bruno Blanchet. *Vérification Automatique de Protocoles Cryptographiques: Modèle Formel et Modèle Calculatoire.* Mémoire d'habilitation à diriger des recherches, Université Paris-Dauphine, 2008.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2012.

[cC01] Ştefan Ciobâcă. *Automated Verification of Security Protocols with Applications to Electronic Voting.* PhD thesis, Laboratoire Spécification et Vérification, ENS Cachan, France, 2001.

[CC09] C. Cachin and N. Chandran. A Secure Cryptographic Token Interface. In *Computer Security Foundations Symposium (CSF)*, 2009.

[CCD13] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. From Security Protocols to Pushdown Automata. In *40th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2013.

[CCD14] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Typing Messages for Free in Security Protocols: the Case of Equivalence Properties. In *25th International Conference on Concurrency Theory (CONCUR)*, 2014.

[CCLD11] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. Trace Equivalence Decision: Negative Tests and Non-determinism. In *18th ACM Conference on Computer and Communications Security (CCS)*, 2011.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a Secure Voting System. In *IEEE Symposium on Security and Privacy (S&P)*, 2008.

[CD09] Véronique Cortier and Stéphanie Delaune. A Method for Proving Observational Equivalence. In *22nd Computer Security Foundations Symposium (CSF)*, 2009.

[CGGI13] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. A Generic Construction for Voting Correctness at Minimum Cost - Application to Helios. *IACR Cryptology ePrint Archive*, 2013.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *International Conference on the Theory and Application of Cryptographic Techniques (EuroCrypt)*, 1997.

[Che14] Vincent Cheval. APTE: An Algorithm for Proving Trace Equivalence. In *20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2014.

[CKS07] Véronique Cortier, Gavin Keighren, and Graham Steel. Automatic Analysis of the Security of XOR-Based Key Management Schemes. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2007.

[CLC03] Hubert Comon-Lundh and Véronique Cortier. New Decidability Results for Fragments of First-Order Logic and Application to Cryptographic Protocols. In *14th International Conference on Rewriting Techniques and Applications (RTA)*, 2003.

[CLC08] Hubert Comon-Lundh and Véronique Cortier. Computational Soundness of Observational Equivalence. In *ACM Conference on Computer and Communications Security (CCS)*, 2008.

[CLS03] Hubert Comon-Lundh and Vitaly Shmatikov. Intruder Deductions, Constraint Solving and Insecurity Decisionin Presence of Exclusive Or. In *18th IEEE Symposium on Logic in Computer Science (LICS)*, 2003.

[Clu03] Jolyon Clulow. On the Security of PKCS#11. In *Cryptographic Hardware and Embedded Systems (CHES)*, 2003.

[CM06] J. Courant and J.-F. Monin. Defending the Bank with a Proof Assistant. In *6th International Workshop on Issues in the Theory of Security (WITS)*, 2006.

[Com] Computer-Assisted Cryptography Group (INRIA/IMDEA). EasyCrypt Webpage. https://www.easycrypt.info/.

[CR05] Yannick Chevalier and Michaël Rusinowitch. Combining Intruder Theories. In *32nd International Colloquium on Automata, Languages and Programming (ICALP)*, 2005.

[Cre08] Cas J. F. Cremers. Unbounded Verification, Falsification, and Characterization of Security Protocols by Pattern Refinement,. In *ACM Conference on Computer and Communications Security*, 2008.

[CRS05] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A Practical Voter-Verifiable Election Scheme. In *10th European Symposium on Research in Computer Security (ESORICS)*, 2005.

[CS09] Véronique Cortier and Graham Steel. A Generic Security API for Symmetric Key Management on Cryptographic Devices. In *14th European Symposium on Research in Computer Security (ESORICS)*, 2009.

[CS13] Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 2013.

[CSW12] Véronique Cortier, Graham Steel, and Cyrille Wiedling. Revoke and Let Live: a Secure Key Revocation API for Cryptographic Devices. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[CW12] V. Cortier and C. Wiedling. A formal analysis of the Norwegian E-voting protocol. In *First International Conference on Principles of Security and Trust (POST)*, 2012.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying Privacy-type Properties of Electronic Voting Protocols. *Journal of Computer Security*, 2009.

[DKS08] Stéphanie Delaune, Steve Kremer, and Graham Steel. Formal Analysis of PKCS#11. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF)*, 2008.

[DLMS99] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[DLS14] Marion Daubignard, David Lubicz, and Graham Steel. A Secure Key Management Interface with Asymmetric Cryptography. In *Third International Conference on Principles of Security and Trust (POST)*, 2014.

[EG02] Laurent Eschenauer and Virgil D. Gligor. A Key Management Scheme for Distributed Sensor Networks. In *9th ACM Conference on Computer and Communications Security (CCS)*, 2002.

[Eig09] Fabienne Eigner. Type-Based Verification of Electronic Voting Systems. Master's thesis, Saarland University, 2009.

[FHF07] A. Feldman, A. Halderman, and E. Felten. Security Analysis of the Diebold AccuVote-TS Voting Machine. In *USENIX/ACCURATE Electronic Voting Technology Workshop (EVT)*, 2007.

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *Workshop on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, 1992.

[FS09] S. Fröschle and G. Steel. Analysing PKCS#11 Key Management APIs with Unbounded Fresh Data. In *Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS)*, 2009.

[Gjø10] Kristian Gjøsteen. Analysis of an Internet Voting Protocol. Cryptology ePrint Archive, Report 2010/380, 2010.

[Gov] Norwegian Government. Web Page of the Norwegian Government on the Deployment of E-voting. http://www.regjeringen.no/en/dep/krd/prosjekter/e-vote-2011-project.html.

[GRBR13] Gurchetan S. Grewal, Mark Dermot Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. Caveat Coercitor: Coercion-Evidence in Electronic Voting. In *IEEE Symposium on Security and Privacy (S&P)*, 2013.

[Gro04] Jens Groth. Efficient Maximal Privacy in Boardroom Voting and Anonymous Broadcast. In *Financial Cryptography*, 2004.

[HRZ10] Feng Hao, Peter Y. A. Ryan, and Piotr Zielinski. Anonymous Voting by Two-round Public Discussion. *IET Information Security*, 2010.

[Hüt02] Hans Hüttel. Deciding Framed Bisimilarity. *Electr. Notes Theor. Comput. Sci.*, 2002.

[IBM06] IBM. CCA Basic Services Reference and Guide. http://www-03.ibm.com/security/cryptocards/pdfs/bs327.pdf, 2006.

[Inc04] RSA Security Inc. PKCS#11: Cryptographic Token Interface Standard, June 2004.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-Resistant Electronic Elections. In *ACM Workshop on Privacy in the Electronic Society*, 2005.

[JLY98] Wen-Sheng Juang, Chin-Laung Lei, and Pei-Ling Yu. A Verifiable Multi-Authorities Secret Election Allowing Abstaining from Voting. *International Computer Symposium*, 1998.

[JS12] Douglas W. Jones and Barbara Simons. *Broken Ballots: Will Your Vote Count?* Center For The Study Of Language And Information, 2012.

[Kah96] David Kahn. *The Codebreakers : the Story of Secret Writing.* Scribner, New York, NY, USA, 1996.

[Kar09] F. Kargl. Sevecom Baseline Architecture. Deliverable D2.1-App.A for EU Project Sevecom, 2009.

[KR05] Steve Kremer and Mark Ryan. Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. In *14th European Symposium on Programming (ESOP)*, 2005.

[KRS10] Steve Kremer, Mark Ryan, and Ben Smyth. Election Verifiability in Electronic Voting Protocols. In *15th European Symposium on Research in Computer Security (ESORICS)*, 2010.

[KSR10] Petr Klus, Ben Smyth, and Mark D. Ryan. ProSwapper: Improved equivalence verifier for ProVerif. http://www.bensmyth.com/proswapper.php, 2010.

[KT08] Ralf Küsters and Tomasz Truderung. Reducing Protocol Analysis with XOR to the XOR-free Case in the Horn Theory Based Approach. *CoRR*, 2008.

[KT09] R. Küsters and T. Truderung. An Epistemic Approach to Coercion-Resistance for Electronic Voting Protocols. In *IEEE Symposium on Security and Privacy (S&P 2012))*, 2009.

[KTV10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and Relationship to Verifiability. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[KTV11] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. In *IEEE Symposium on Security and Privacy (S&P)*, 2011.

[KTV12] R. Küsters, T. Truderung, and A. Vogt. Clash Attacks on the Verifiability of E-Voting Systems. In *IEEE Symposium on Security and Privacy (S&P)*, 2012.

[LBD+04] Byoungcheon Lee, Colin Boyd, Ed Dawson, Kwangjo Kim, Jeongmo Yang, and Seungjae Yoo. Providing Receipt-Freeness in Mixnet-Based Voting Protocols. In *6th International Conference on Information Security and Cryptology (ICISC)*, 2004.

[Lev10] Frederic Levy. SAM and Key Management Functional Presentation. Available from http://www.calypsostandard.net/, December 2010.

[Liu11] Jia Liu. A Proof of Coincidence of Labeled Bisimilarity and Observational Equivalence in Applied Pi Calculus. http://lcs.ios.ac.cn/~jliu/papers/LiuJia0608.pdf, 2011.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 1996.

[LR92] Dennis Longley and S. Rigby. An Automatic Search for Security Flaws in Key Management Schemes. *Computers & Security*, 1992.

[MM11] Sebastian Mödersheim and Paolo Modesti. Verifying SeVeCom using set-based abstraction. In *IWCMC*, 2011.

[MS01] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security (CCS)*, 2001.

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *25th International Conference on Computer Aided Verification (CAV)*, 2013.

[NYHR05] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The kerberos network authentication service - rfc 4120. Available at http://tools.ietf.org/html/rfc4120, July 2005.

[OMA+99] Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka, and Tatsuaki Okamoto. An Improvement on a Practical Secret Voting Scheme. In *Second International Information Security Workshop (ISW)*, 1999.

[Riv06] Ronald L. Rivest. The ThreeBallot Voting System. Unpublished draft. Available at http://people.csail.mit.edu/rivest/pubs/Riv06c.pdf, October 1, 2006.

[RT01] Michaël Rusinowitch and Mathieu Turuani. Protocol Insecurity with Finite Number of Sessions is NP-Complete. In *14th IEEE Computer Security Foundations Workshop (CSFW)*, 2001.

[Rya09] Mark Dermot Ryan. Introduction to the TPM 1.2. Technical report, University of Birmingham, 2009.

[SB06] G. Steel and A. Bundy. Attacking Group Protocols by Refuting Incorrect Inductive Conjectures. *Journal of Automated Reasoning*, 2006.

[Sch95] Bruce Schneier. *Applied Cryptography (2nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[SFCC11] Alan T. Sherman, Russell A. Fink, Richard Carback, and David Chaum. Scantegrity III: Automatic Trustworthy Receipts, Highlighting Over/Under Votes, and Full Voter Verifiability. In *Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE*, 2011.

[TD10] Alwen Tiu and Jeremy E. Dawson. Automating Open Bisimulation Checking for the Spi Calculus. In *3rd IEEE Computer Security Foundations Symposium (CSF)*, 2010.

[THG99] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security*, 1999.

[Tru11] Trusted Computing Group. *TPM Main Part 3 Commands*, level 2 revision 116 edition, March 2011. Specification Version 1.2.

[Vig06] Luca Viganò. Automated Security Protocol Analysis With the AVISPA Tool. *Electr. Notes Theor. Comput. Sci.*, 2006.

[Wei99] Christoph Weidenbach. Towards an Automatic Analysis of Security Protocols in First-Order Logic. In *16th International Conference on Automated Deduction, CADE*, 1999.

[Wey11] B. Weyl. Secure On-board Architecture Specification. Deliverable D3.2 for EU Project EVITA, http://evita-project.org/Deliverables/EVITAD3.2.pdf, August 2011.

[Wie] Cyrille Wiedling. Source Files for ProVerif, JavaCard and RF* Code *(Under Construction)*. http://www.loria.fr/∼wiedling/Resources/resources.html.

[WWH+10] S. Wolchok, E. Wustrow, J. A. Halderman, H. K. Prasad, A. Kankipati, S. K. Sakhamuri, V. Yagati, and R. Gonggrijp. Security Analysis of India's Electronic Voting Machines. In *17th ACM Conference on Computer and Communications Security (CCS)*, 2010.

[YW07] Xukai Zouz Yong Wan, Byrav Ramamurthy. KeyRev: An Effifcient Key Revocation Scheme for Wireless Sensor Networks. In *IEEE International Conference on Communications (ICC)*, 2007.

Abstract

For a long time now, formal methods have been successfully used to analyze security protocols. Several tools have even been developed to tackle automatically different proof techniques and, therefore, to ease the verification of such protocols. However, when it comes to electronic voting and APIs, current tools tend to reach their limits because they can't handle some cryptographic primitives (e.g. homomorphic encryption), or the security properties (e.g. ballot secrecy), involved in those protocols.

In this thesis, we first work on two cases studies of existing and deployed systems: a Norwegian e-voting protocol and a CNRS boardroom voting protocol. These two protocols are analyzed using the applied pi-calculus model and we discuss in details about their security properties, in different corruption scenarios. Even though this part provides several reusable results, it also shows the complexity to prove them by hand and, therefore, underlying a real need for automation in those proofs.

The third part of this thesis focuses on a possible lead in direction of this needed automation: type-systems. Indeed, we build upon a recent work describing a new type-system designed to deal with equivalence properties, in order to apply this on the verification of equivalence-based properties in electronic voting like ballot-secrecy. We present an application of this method through the example of Helios, a well-known, web-based, and open-audit, e-voting system.

Another family of advanced security protocols are APIs: secure interfaces devoted to allow access to some information stored into a secured trusted hardware without leaking it outside. Such protocols are not immune to attacks; in fact, recent work seems to show the opposite. In the second part of this thesis, we provide a new design for APIs, including a revocation functionality, which is not always considered. In addition, we include a fully formal analysis of this API allowing that a malicious combination of API's commands does not leak any key, even when the adversary may brute-force some of them.

Keywords: Formal Methods, Security, Protocols, E-Voting, APIs, Type-Systems

Résumé

Les méthodes formelles ont depuis longtemps fait leurs preuves dans l'études des protocoles de sécurité. Il existe même plusieurs outils qui permettent d'automatiser la vérification de ces protocoles. Toutefois, ceux-ci se montrent encore parfois dans l'incapacité d'analyser certains protocoles, que ce soit à cause de certaines primitives cryptographiques employées ou des propriétés que l'on cherche à démontrer.

Dans la première partie de cette thèse, on étudie deux systèmes existants: un protocole de vote par internet Norvégien et un protocole pour les votes en réunion du CNRS. A l'aide du pi-calcul appliqué, nous analysons des garanties de sécurité qu'ils proposent et ce, au travers de différents scénarios de corruption. Mais malgré les résultats réutilisables obtenus, ces preuves démontrent également la difficulté de les effectuer à la main.

C'est pourquoi la troisième partie de cette thèse est consacrée à une nouvelle piste dans l'automatisation de telles preuves: les systèmes de types. En effet, inspirés par le développement récent d'un système de types permettant de traiter des propriétés d'équivalence, nous l'avons utilisé afin de démontrer des propriétés comme l'anonymat dans le vote électronique. Nous avons appliqué cette méthode à un exemple: Helios, un système de vote par internet bien connu.

Il existe une autre famille de protocoles de sécurité : les APIs. Ces interfaces permettent l'utilisation d'informations stockées dans des dispositifs sécurisés sans qu'il soit normalement possible de les en extraire. Des travaux récents montrent que de telles interfaces sont également vulnérables. La deuxième partie de cette thèse présente un nouveau design d'API, incluant une fonctionnalité de révocation, rarement présente dans les solutions existantes. Nous démontrons, par une analyse entièrement formelle, qu'aucune combinaison de commandes ne permet de faire fuir des clefs sensées rester secrètes, même si l'adversaire parvient à en brute-forcer certaines.

Mots-clés: Méthodes Formelles, Sécurité, Protocoles, Vote Electronique, APIs, Systèmes de Types.