Design and formal analysis of security protocols, an application to electronic voting and mobile payment

Alicia Filipiak

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Conception et analyse formelle de protocoles de sécurité, une application au vote électronique et au paiement mobile

THÈSE

présentée et soutenue publiquement le 23 mars 2018

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

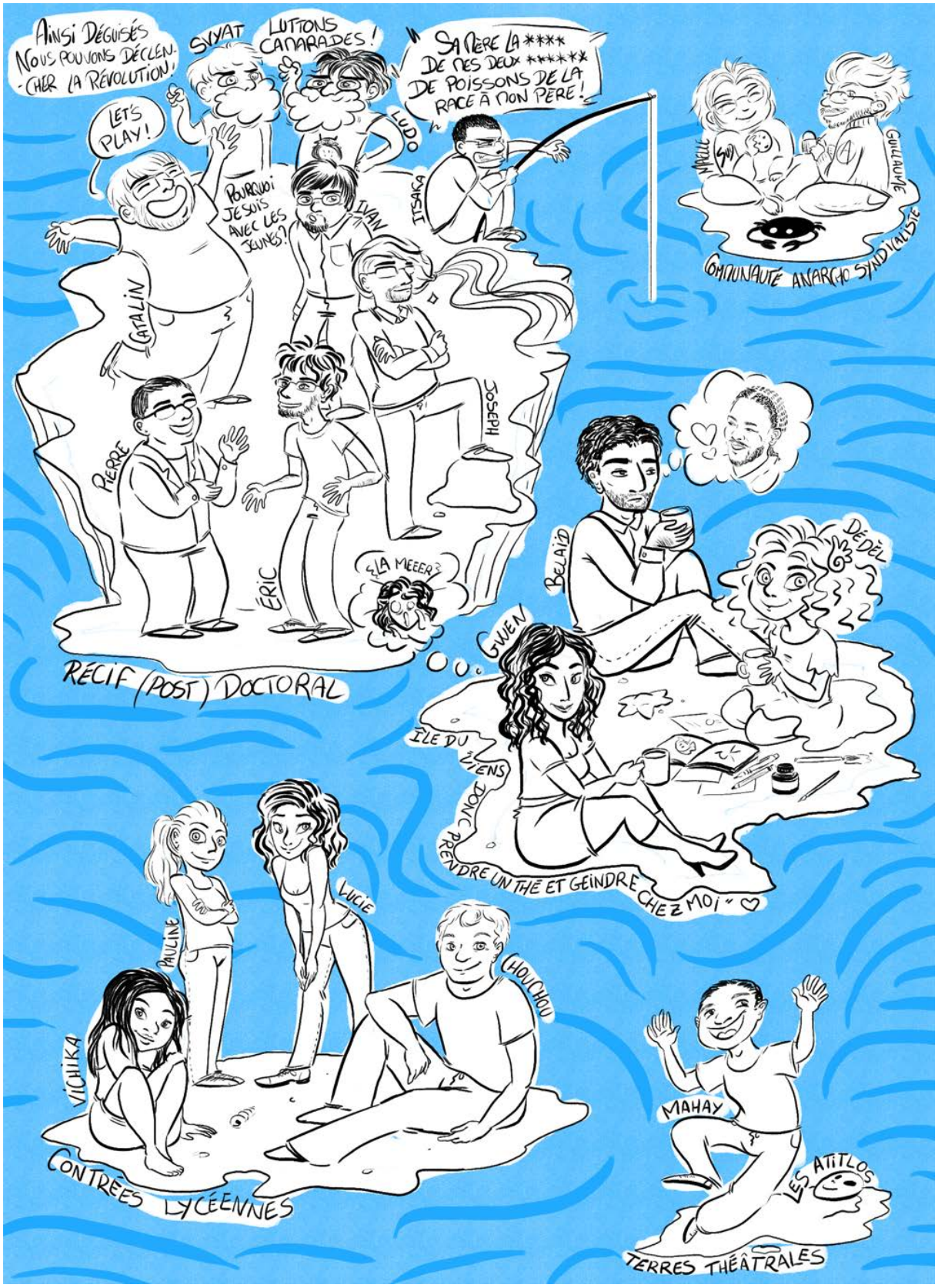
Alicia Filipiak

Composition du jury

Rapporteurs :	Valérie VIET TRIEM TONG Cas CREMERS
Examinateurs :	Véronique CORTIER Horatiu CIRSTEA Olivier PEREIRA Saïd GHAROUT
Invité :	Jacques TRAORE

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

Remerciements


AINSI DÉGUISÉS NOUS POUVONS DÉCLENCHER LA RÉVOLUTION!
SVYAT
LUTTONS CAMARADES!
SA PÈRE LA **** DE DES DEUX ****** DE POISSONS DE LA RACE À MON PÈRE!
LET'S PLAY!
LUDO
POURQUOI JE SUIS AVEC LES JEUNES?
IVAN
ITSARA
CATALIN
JOSEPH
PIERRE
ÉRIC
LA MEEER?
RÉCIF (POST) DOCTORAL
MARIE
GUILLAUME
COMMUNAUTÉ ANARCHO SYNDICALISTE
BELAID
DÉDÉL
GWEN
ÎLE DU LIENS
TONIC
PRENDRE UN THÉ ET GEINDRE CHEZ MOI" ♡
VICHIKA
PAULINE
LUCIE
CHOUCHOU
CONTREES LYCÉENNES
MAHAY
LES ATITLOS
TERRES THÉÂTRALES


MELISSA
JACQUES
PARIKO
MARIE-CRICRI
EXTENSION COLONIALE DE LA FAMILIA
BOUSKADOUX
KILLYA
AUX DEUX PATAPONS
PAPA
MAMAN
PIERRE
DIDIE
MAMA ALICIA
MAMIE
LA FAMILIA
JHIH-HUANG
BENOIT
MIGUEL
REFUGE DES ANCIENS BAGNARDS

Table of Contents

Introduction

The last decade has seen the massive democratization of smart devices such as phones, tablets, even watches. In the wealthiest societies of the world, not only do people have their personal computer at home, they now carry one in their pocket or around their wrist on a day to day basis. Those devices are no more used simply for communication through messaging or phone calls, they are now used to store and share personal photos or critical payment data, manage contacts and finances, connect to an e-mail box or a merchant website... We even use them for more complex tasks than just data management or secure connections. Estonia voting policy allows the use of smart ID cards and smartphones to participate to national elections [33]. In 2017, Transport for London launched the TfL Oyster app [60] to allow tube users to top up and manage their Oyster card from their smartphone.

As services grow with more complexity, so does the trust users and businesses put in them. We expect a mobile payment application to protect the client's critical payment data while being also largely scalable and guaranteeing that a merchant who validates a transaction will indeed be paid. We expect an electronic voting protocol to guarantee to each voter that their vote will be part of the final result and to ensure that the voter will be able to verify that through an accessible process that does not require great technical skills, while also ensuring there is no breach in the vote confidentiality.

Yet, this trust implies the trust of the whole service inside a complex ecosystem. We do not limit it to a simple trust between two entities anymore, such as in a client-server dynamic. When we trust a payment application, we trust that the messages exchanged between a payer, their payment card, the merchant and their payment terminal, the banks of both parties, the payment network over which transaction messages

are transmitted are thought to guarantee the security expectations we have about such an application. When we trust an electronic voting schemes, we trust that voters, servers, administrations and ballot box interact in a way that does not compromise the integrity of the result and the secrecy of the vote. This kind of trust goes beyond trusting a single software, for several entities taking part on a service might not even use the same application or operating system *we need to trust that entities exchange message in a secure way*.

1 Secure design of cryptographic protocols

Protocols define the exchanges between devices and entities, they are a humdrum of classical Internet surfing (TCP [73], UDP [98], HTTP [59] and HTTPS [99], DNS [90] and DNSSEC [7]...). *Cryptographic protocols* are specifically designed to ensure security guarantees, some of them quite classical: such as guaranteeing *authentication* of several parties to one another - like the Kerberos [94] protocol - or the secrecy of a specific value - as one of the optional functionality of IPsec [95]. Some others are a little more specialized, like the security claims of the Helios voting protocol that we use as a running example in this introduction.

1.1 An example of protocol: the Helios voting scheme

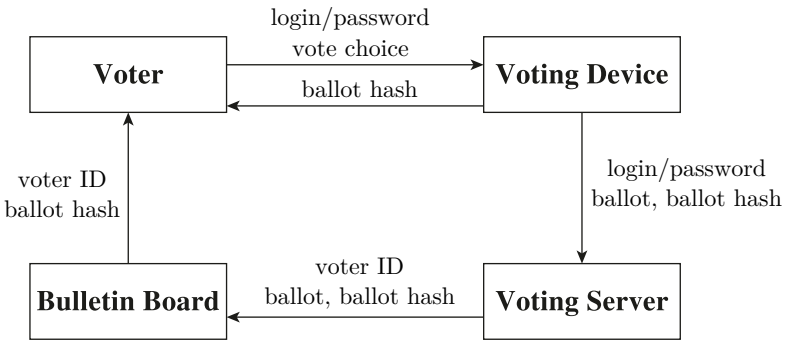


Figure 1: An overview of the Helios voting protocol

The Helios protocol [3, 4] defines a web-based voting scheme whose result can be audited by any third party thanks to the public election data. It was launched in 2008 and has been used for several elections such as the IACR board members elections since 2011.

In essence, during a Helios-based election, an *election administrator* sets the election up by defining a public list of eligible voters - or aliases - and a set of valid votes. Cryptographically speaking, the protocol relies on the El Gamal [48] asymmetric encryption scheme. A *tallying authority*, in charge of computing the election result, generates a pair of asymmetric election keys and publishes the election public key. Note that the tallying authority is usually a group of several entities that share the election decryption key through a threshold cryptosystem. A *voting server*, in charge of managing a publicly available ballot box, generates authentication credentials - typically a login/password pair - for each eligible voters. Those credentials are used by *voters* to connect to the voting server through their own *web browser*.

The voting phase steps are summarized in Fig.1.

1. The voter connects to the election dedicated web page through their browser. There, a *ballot preparation system* (BPS) runs as a service on the browser. The BPS allows the voter to choose their option among the valid votes set.

2. After recording the voter's choice, the BPS generates a fresh nonce and encrypts it together with the choice using the election public key. The BPS displays the hash of ballot.

3. The voter can process to an *optional* audit step of the ballot: the BPS discloses all information necessary to recompute the voter's ballot and check if it matches with the ballot that was displayed. The voter can use a third party server to perform this verification. This step can be repeated for as much as the voter wants. Once they are convinced that the ballot encryption is well performed, the BPS generates a new ballot randomized with another parameter.

4. The voter then *seals* the ballot. The BPS discards the nonce used to compute the ballot as well as the original vote of the user. The ballot is ready to be cast.

5. The voter connects to the voting server through their web browser with their authentication credential. The voting server records the ballot as the voter's choice and displays the voter identifier along the hash of the ballot on the public bulletin board.

6. The voter can process to an *optional* verification step. They look for their identifier on the bulletin board and check that the hash of their ballot matches the value displayed on the bulletin board.

Once the election is declared as over by the election administrator, ballots are collected by the administrator which performs the verification that each ballot was cast by an eligible voter - by checking the voting server's logs - and that all ballots are valid. If the verification of those is successful, thanks to the homomorphic properties of the El Gamal encryption scheme, ciphertexts are homomorphically combined to produce the encrypted tally on the bulletin board. The tally of the election is decrypted - along a proof of good computation that can be checked by anyone - by all entities composing the tallying authority and the result of the election is published.

Any third party can audit the election result by downloading the election public data - the list of eligible voters names or aliases, the public bulletin board that was tallied and the final tally. The election auditor can check that ballots were cast *with* eligible voters' name or aliases (and would have to trust the authentication performed by the server) and that they were all valid. The proof of good computation for the final result can also be checked.

Helios was proven to be *verifiable* [38]. Voters can individually check their vote is taken into account in the election and auditors can also check that all ballots in the bulletin board were part of the final tally.

The protocol has also been claimed as *preserving the vote confidentiality*. No one can allegedly guess a voter's vote, since all ballots were computed by using a different nonce and since the result is output after an homomorphic combination of all ballots, there is supposedly no way to link a ballot to a voter.

1.2 An error-prone task

The design of cryptographic protocols is known to be an error prone task. The classical example might be the Needham-Schroeder [93] authentication protocol that was shown to be vulnerable to a man-in-the-middle attack by Lowe [84] seventeen years after the creation of the protocol. A most recent example is the replay attack on the four-handshake of WPA2 [111], the protocol designed to secure wireless network connections that is widely used by our devices.

Example 1 (Helios vulnerabilities). The Helios protocol described in previous section 1.1 also presents some vulnerabilities on both privacy [41] and verifiability.

The attack on privacy is summarized in Fig.2. We suppose that an election is held for three voters: two honest (Blue and Green) and one dishonest (Red) - but still eligible for the election.

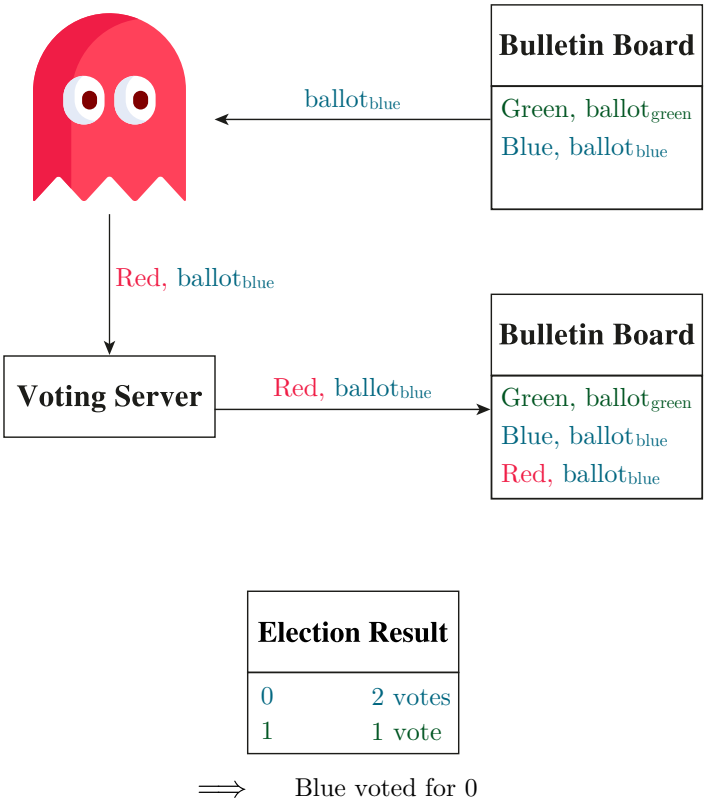


Figure 2: An attack on Helios' privacy

1. Since ballots are displayed along the voter's identifier on the public bulletin board, the attacker can retrieve one specific voter's ballot (Blue's one in Fig.2).

2. The attacker can then recast the ballot under the rogue voter's name (Red) to the voting server. The voting server accepts the rogue voter's ballot, for they are eligible and displays it on the public bulletin board.

3. Ballots are then tallied and the attacker can deduce its target's vote by analyzing the result (candidate "0" has two votes whereas candidate "1" one in the final tally, so Blue voted for "0").

If we had more eligible voters for the election, and if the attacker had a significant amount of voters under its control, the attacker could replay a voter's ballot enough times so that the occurrences of this particular ballot would magnify the honest voter's choice in the final result of the election. The vote confidentiality of this honest voter cannot be guaranteed.

Another vulnerability relies on the strong trust assumption we make on both the voting server and the personal voters devices used to cast ballots. Indeed, there is no control over the voting server in the design of Helios. A rogue voting server could stuff the bulletin board to influence the final election result by simply casting ballots for absentees, without being detected. This would compromise the integrity of the result and thus verifiability. If we also assume the voter's device is under an attacker control, which would be a reasonable assumption, the vote confidentiality would not hold, for the device would *see* the voter's choice.

In all of these examples, cryptography is not to blame, the design itself of the protocols allows an attacker to replay or intercept some values or exploit the cryptographic relations between exchanged

messages to break the protocol security. Indeed, an attacker is rarely a passive entity. At the end of it, it comes to two simple questions:

- *What* can a reasonable attacker do?
- *Who* is the attacker?

In other terms, *what is our attacker model*?

With this in mind, we can start asking ourselves what are the possibilities to prove the security of a protocol in the context of active attackers.

1.3 Protocol security proofs in the symbolic model

Years of research have given us several conceptual tools to mathematically model and prove the security of cryptographic protocols. Two worlds substantially coexist in this area: the computational model and the symbolic model. The first one focuses on finding flaws in the algorithms used by cryptographic primitives and is the model generally used by cryptographers. On the other hand, the symbolic model considers cryptographic primitives to be perfect and focuses on finding the logical flaws in the whole protocol. During this thesis, we proved our protocols using the symbolic model, thus we will not discuss here the differences between both kind of models as it has already been done by others [20].

Protocol modeling

Historically, we owe symbolic models to Needham, Schroeder, Dolev and Yao [93, 47]. In symbolic models, the cryptographic primitives are modeled as abstract function symbols and the operations they can perform - such as pairing, projecting, encrypting, signing, hashing - as rewriting rules or equalities modulo an equational theory.

Example 2 (asymmetric encryption). Considering the asymmetric encryption we need to model if we want to formally model the Helios voting scheme described in section 1.1 we represent the function generating a public key from a secret key by the function symbol pk, the asymmetric encryption by the function symbol aenc and the asymmetric decryption by the function symbol adec.

The decryption of a ciphertext (m) with the private key (k) can then be modeled by a rewriting rule of the form:

$$\mathsf{adec}(\mathsf{aenc}(m, \mathsf{pk}(k)), k) \rightarrow m$$

Or by the equation:

$$\mathsf{adec}(\mathsf{aenc}(m, \mathsf{pk}(k)), k) = m$$

Messages that are exchanged over the protocol are modeled as abstract terms built over a term algebra that includes the application of the function symbols modeling cryptographic primitives.

Attacker capabilities

As mentioned previously, we need to consider an active attacker when challenging the security of a protocol design. It shall have the full control of the public network, meaning it should be able to eavesdrop, send, receive and drop messages to participating entities and it should be able to compute the same cryptographic operations as the one used in the protocol. For instance, we could expect that an attacker is able to perform the encryption of a known plaintext, or the hash of a specific value.

This kind of behaviour is modeled by what is called a *Dolev-Yao adversary* [47] in the symbolic model. This kind of attacker is quite powerful, yet, it should be noted that the operating capabilities of

the attacker are limited to what kind of operations the protocol model authorizes. If the cryptographic signature scheme is not modeled in a specification, then the attacker would not be able to perform it. Still, the Dolev-Yao attacker is powerful, and modeling it this way allows to find an extensive amount of logical flaws in cryptographic protocols.

Modeling security properties

The symbolic model is expressive enough to express security properties - usually taking the form of logic lemmas - we can divide the security properties it can express into to distinctive families.

The first category, known as *trace properties*, allows to express the fact that an attacker should not be able to obtain a specific value - by deriving it - or the fact that if the protocol reaches a specific state with specific values, then another state with same values has also been reached. Those kind of properties are fit to model secrecy, mutual authentication or key agreement.

Example 3 (Informal cast as intended property). If we consider the Helios voting scheme, one property we could expect to be satisfied by the protocol commonly known as the *cast as intended* property can be intuitively expressed as follows:

"If the voting server registers the ballot b for the voter ID, then the voter ID has previously cast the ballot b encrypting the voter's choice v."

The other kind of properties are called *equivalence properties*. They state that an attacker should not be able to distinguish two processes. They are more commonly used to model anonymity or other privacy-type properties.

Example 4 (Informal vote confidentiality). Still considering the Helios voting scheme, we can expect it to preserve the vote confidentiality of those participating to the election. However, modeling the vote confidentiality is not trivial. We cannot satisfy ourselves with hiding simply the voter's identity - which is usually revealed during an authentication process to a voting server. Also, we do not want a voter's choice to be revealed but votes are - usually - publicly known values. So we want to *hide the link* between a voter and their vote. Hiding this link is classically [45] modeled by the following intuitive statement:

"An attacker should not be able to distinguish the case were Alice voted for candidate "0*" and Bob for candidate "*1*" from the case where the votes are reversed."*

Automation of security proofs

Symbolic models benefit from the great advantage of allowing the automation of security proofs. Two main families of model checkers for protocols exist.

There are tools who authorize only a *bounded number of sessions*, those are the ones that are efficient in finding vulnerabilities in protocols. The number of sessions handled by such tools are often very restricted. The AVANTSSAR platform (formerly known as AVISPA [15]) proposes three different tools (OFMC [16, 91], SATMC [8] and CL-AtSe [110]) that all bound the number of sessions, like Scyther [86]. Both tools deal with trace properties only. AKISS [10, 28], APTE [31], SAT-Equiv [34], TypeEQ [40] and SPEC [109], on the other hand, handle equivalence properties while Maude NPA [57] can manage both trace and equivalence properties.

The other kind of tools authorize an *unbounded number of sessions*. Those are tools mainly used to prove the security of protocols. In this category we have, from least popular to most used, TypeEQ [40], Maude NPA [57], Scyther [86], Tamarin [104] and ProVerif [19]. Table 1 provides an overview of what can automated tools handle. Note that even though Maude NPA checks all criteria in this table, its lack of popularity is justified by the fact that it often does not terminate.

This thesis is focused on designing and proving secure protocols in two use cases: electronic voting and mobile payment. To this purpose, we needed to rely on tools that allowed an unbounded number of

Tool name	Bounded number of sessions	Unbounded number of sessions	Trace properties	Equivalence properties
AVANTSSAR	✓	✗	✓	✗
AKISS	✓	✗	✗	✓
APTE	✓	✗	✗	✓
SAT-Equiv	✓	✗	✗	✓
SPEC	✓	✗	✗	✓
TypeEQ	✓	✓	✗	✓
Scyther	✓	✓	✓	✗
Maude NPA	✓	✓	✓	✓
ProVerif	✗	✓	✓	✓
Tamarin	✓	✓	✓	✓

Table 1: Model checkers for cryptographic protocols

sessions. Plus, our security analysis needed to verify both trace and equivalence properties. Out of the model checkers that allow an unbounded number of sessions for verification, only Maude NPA, Tamarin and ProVerif fit our needs. Maude NPA often does not terminate so instead, we chose to rely on ProVerif and Tamarin for our proofs.

2 Scope of this thesis

The tools mentioned in the previous section have been quite extensively used to study classical security protocols such as authentication ones. Usually those kinds of protocols are rather small. This is not necessarily the case for "industrial-level" protocols - such as payment and voting - which are heavier, rely on quite numerous data exchanged and aim at more complex security properties in most cases. During a card-based transaction for instance, how does one make sure the bank account information transmitted to a merchant terminal are kept secret and on what grounds do such guarantees rely on? Or how does an electronic voting scheme ensures the voters their vote is indeed going to be part of the final result tally and is going to be kept private?

We chose to focus this thesis on providing two *practical* protocols in two different use cases along their security proofs.

3 Electronic voting: the Belenios VS voting scheme

From the first proposal to use voting machines by the English Chartists during the XIXth century to nowadays Estonian electronic voting system for national elections, the automation of voting has been a vast topic of research and engineering for almost two centuries. Two families of voting designs coexist regarding this question: voting at a polling station assisted by voting machines for voting and/or tallying - like Diebold voting machines that are used in the USA - or voting over the Internet - like the Estonian elections.

Like traditional physical voting schemes, one of the main concern relies on the fact that a "good" election should satisfy two properties which, combined together, seem to form quite an antinomy: the vote confidentiality should be guaranteed for everyone while every voter should be able to verify that their vote is going to be part of the final tally.

There has been several security studies regarding those concerns over the last two decades, some of them have unveiled serious flaws in some schemes such as Diebold voting machine or the Estonian voting protocol [58, 107]. On another level, some protocols have been proven as secure, yet, they usually need a voter's device to be uncompromised, an assumption that we could prefer not to make.

3.1 Web-based voting

The first use case of this thesis is an e-voting protocol. Those kinds of protocols can be divided into two families:

In-site voting protocols

Those protocols usually rely on *paper ballots* and require the voter to go vote at a polling station. In this category, we have for instance:

- *Prêt à Voter* [101]: at the polling station, a voter randomly chooses a ballot marked with a unique cryptographic value. The ballot is divided into two parts: the list of candidates for the election and the voting sections. Candidates' order on the ballot is random. To vote, the user simply checks the section matching their candidate. They then separate both parts of the ballot and give them to the scanner that will put them on the ballot box. The cryptographic value on the ballot allow the computation of the candidates order during the tally to find out for which candidate the voter voted. A voter can verify that their vote was tallied by checking the list of tallied ballots numbers on a web page.

- *(Revised) Three Ballots* [100]: a voter gets three ballots, each one marked with a different identifier, those three ballots represent three columns. Rows are reserved for each candidate. To vote for a candidate, the voter marks two cross on one its row, the other candidates must receive one cross only. All three ballots are given at the polling station and the voter keeps one copy of one of the three ballots as a receipt. Ballots are published on a public bulletin board and the voter can check that their receipt is part of the final tally. This protocol actually does not require cryptography to be used on part of the voter.

- *Scantegrity II* [30]: this protocol aims at improving the verifiability of optical scan voting systems. Voters mark their ballot with invisible ink like traditional optical voting systems.

Remote voting protocols

On the other hand, these protocols are usually *purely electronic* voting protocols that only require an Internet connection to cast a ballot. This category counts quite a few protocols such as:

- *Civitas* [32]: this protocols claims to be the first remote electronic voting system that is coercion-resistant and verifiable. To resist to coercion, the voter can use fake credentials to cast fake ballots. Although its coercion-resistance represents its main appeal, Civitas is not a simple protocol to use for everyone.

- *Scytl* [39]: to cast a ballot, the voter relies on a voting card they previously received. Through a voting device, they input their password provided on this voting card. At several steps, during the voting phase, the user receives confirmation codes from their device and the server - that are displayed on the voting card - to confirm their ballot is the one matching their voting choice.
- *sElect* [80]: sElect is a lightweight web-based voting schemes whose main appeal resides in the fact that the verification process is fully automated. Accessibility of the scheme to voters is then greatly improved since no particular skill or task is required to verify that the vote is part of the tally. sElect was designed specifically for low-risk elections, like most of remote voting protocols.
- *Selene* [102]: the verifiability of this protocol is guaranteed by the use of an individual and private tracking number for ballots. Each ballot is displayed along its tracking number on the bulletin board.
- *Helios* [3, 4]: we described the Helios protocol along this introduction so we will not elaborate on the scheme here.
- *Belenios* [38]: Belenios is a variant of Helios that addresses the problem of having to trust the voting server. It adds an entity to the voting ecosystem, the registrar, that provide signature keys to voters in order to sign their ballots.
- *Belenios RF* [29]: Belenios Receipt-Free is a variant of Belenios that relies on the use of randomizable cryptographic schemes thus addressing the coercion problem. A voter cannot prove how they voted (event though they can still sell their credentials).

Table 2 propose an overview of the security claims of each one of this examples.

Building upon the Belenios voting schemes, we aimed at developing a protocol that guarantees both verifiability and vote confidentiality against a malicious server and a malicious device: Belenios VS. Our protocol is a *remote voting scheme* that relies on the use of a *voting sheet* to cast a ballot though.

3.2 Contributions

We list here the two main contribution that we made in the context of web-based voting schemes.

Proposing and proving an improvement of the Belenios RF protocol: Belenios VS along its security proof

We propose an improvement of the Belenios RF protocol called *Belenios Voting Sheet* (Belenios VS). Indeed, the Belenios protocols guarantee both verifiability and vote confidentiality, but it also assumes that the user's device is secure. We address this concern by relying on precomputed ballots. In Belenios VS, the voter does not compute directly their vote from their device. Instead, they receive an individual voting sheet where ballots are precomputed - this way a rogue voting device cannot "see" or modify the voter's choice - that the voter must scan to cast a ballot. A voter can audit their voting sheet - or delegate this operation to a person of trust - to check its conformity with the election. This allows us to cut even more the link between a voter and their vote, for the signature key used to sign the voter's ballot is not linked to a specific identity as in Belenios and Belenios RF. In fact, such keys are never displayed on the voting sheet and voting sheets are randomly distributed among all eligible voters.

Along the specification of the protocol, we provided an extensive security analysis in ProVerif that considers each plausible corruption cases combination (for instance by considering the cases where the voting sheet is leaked and the voting server is under the attacker control, or when the registrar is corrupted and the voter credentials are stolen) to state exactly the limits and strengths of our protocol.

Protocol	**Paper ballots**	**Verifiability**		**Vote confidentiality**	
		Rogue server	Rogue device	Rogue server	Rogue device
Prêt à Voter	YES	✓	-	✓	-
Three ballots	YES	✓	-	✓	-
Scantegrity	YES	✓	-	✓	-
Civitas	NO	✓	✗	✓	✗
Scytl	NO	✗	✓	✗[39]	✗
Select	YES	✓	✗	✓	✗
Selene	YES	✓	✗	✓	✗
Helios	NO	✓(1)	✓(2)	✗[41]	✗[41]
Belenios	NO	✓	✗	✓	✗
Belenios RF	NO	✓	✗	✓	✗
Belenios VS	YES	✓	✓	✓	✓

- : does not apply
✗ : compromised
✓ : satisfied
(1) : the voting serve may stuff the ballot box for absentees though
(2) : if audit of the computation with a different honest device

Table 2: A comparison of existing voting protocols and their security claims

In turns out that under some security assumptions, our protocol guarantees both verifiability and vote confidentiality even if a user's device is compromised.

Providing two theorems allowing the automation of verifiability proofs with ProVerif

A voting scheme can be qualified as *verifiable* on several levels. We want to make sure that each voter can be ensured that their vote is going to be part of the final tally (*individual verifiability*). We also want to make sure that the final result of an election is the one that matches all ballots that were cast in the ballot box *universal verifiability*. Finally, we also want to have some control over the fact that only authorized voters were allowed to participate to the election (*eligibility verifiability*), or at least that, if there are eligible corrupted voters, there are no more rogue ballots than the number of rogue voters.

This property has been formally expressed in [38]. However, such a property cannot be expressed with the ProVerif calculus, for it would need modeling a "counting" function, something that the tool does not handle. nor Tamarin - because the cryptographic primitives our protocol relies on can only be described with an equational theory that leads Tamarin into an infinite loop.

Some efforts have been made to automate the verifiability security proof [42, 39] by proposing a set of simple properties that are provable in ProVerif and imply verifiability. Yet, they prove a weaker verifiability than the one we evoked, for there is not eligibility verification in those proofs. That is why we provided two sets of trace properties that can be expressed in the ProVerif calculus. If one of those sets is satisfied by the protocol then the protocol is verifiable.

We applied both theorems to our security analysis of Belenios VS mentioned in the previous section.

4 Mobile payment: a token-based payment protocol for mobile devices

Mobile phones have been extensively used since their launching to perform or confirm transactions. We can use them to secure Internet transactions by providing a two factors authorization process as in 3D-Secure[97, 56] or even directly pay from them on merchants website, whether it is from a browser of with a "payment button" as in Apple Pay. In some countries they are even used to process to peer to peer payment and are reliable in continents where the vast majority of the population do not own a bank account, as in Africa. And last but not least, smartphones are now able to fully emulate payment cards to process to NFC-based payment and the competition is fierce between all new payment applications (Apple Pay, Android Pay, Samsung Pay, Orange Cash, Paylib...).

The last kind of applications is the one we focused on during this thesis.

4.1 Scalable mobile payment

As stated before, the competition between payment applications is fierce. Yet, security breaches on those applications are regularly found [88]. Partly due to the fact that there is no standard describing the exact requirements of mobile payment applications - merely frameworks or white papers - and the fact that each major actor in the market keep the specifications of their solutions away from the public eyes.

Nonetheless, one common ground between all of these applications is the fact that to be competitive, they must be as much scalable as possible. In the payment world, this means complying with existing standards for payments, usually card-based. The most widely deployed standard about payment is the EMV Chip and PIN standard, issued by the consortium EMVCo regrouping major card constructors such as Visa or MasterCard. For an application to be scalable without needing a massive update on merchant terminals, It is critical to be compatible with this specification.

Regarding the security management of mobile payment solutions, two main trends can be identified. The first one emulates the payment card inside a tamper-resistant hardware called a Secure Element, as in Apple Pay or Orange Cash. Development and maintenance costs are expensive for such solutions but it inherits the security of Chip and PIN cards. The second trends emulates the payment card directly from the smart device main OS, as in Android Pay. Such solutions are highly flexible and cheaper to maintain and develop but the security is more fragile than in Secure Element-centric solutions.

We proposed a payment protocol along its security proof that was proven secure even if the client device is infected.

4.2 Contributions

We describe here our the two main contributions we made regarding the context of mobile payment applications.

Proposing an open end-to-end specification of a mobile payment application

To the best of our knowledge, we proposed the first open end-to-end specification of a mobile payment protocol, for most mobile payment applications are proprietary solutions that are very opaque regarding the way they operate.

Our application relies on the use of ephemeral aliases that replace the original payment data that are called tokens. The EMVCo consortium issued a technical specification in 2014 [55] to explain the requirements for tokens regarding the existing payment ecosystem. Nonetheless, those requirements are

purely technical and there is no protocol describing the interactions between payer, merchant and banks to proceed to a token-based transaction. This is why we designed this protocol.

In our protocol, the user has a "tokens vault" on their device that they must regularly provision. Such tokens are stored directly encrypted on the mobile device and cannot be decrypted unless the Secure Element is involved. For each payment, the token is cryptographically bound to a transaction amount and a merchant identifier, thus restricting the stealing window for potential attackers, for an attacker could only use a token for the same merchant and the same amount.

We actually proved that our protocol guarantees as least as much security as the EMV authentication protocols. First, a user cannot get stolen, at least not easily, since they have to approve a transaction for a token to be decrypted. Second, the merchant benefits from an insurance that the payment owed by a client will be obtained. Those security claims are part of the EMV specification for Chip and PIN cards. We even considered not only the case of a corrupted device but of corrupted payment data and still managed to prove some security guarantees: even in the case of a stolen token, the user cannot be charged any money. In addition to those properties, we also proved that since our payment protocol relies on ephemeral payment data that can only be used once, a merchant could not track the client consumption habits, thus adding a privacy dimension to our protocol that nor EMV nor classical payment application such as Apple Pay can provide.

Formally proving the security of the mobile payment protocol with the Tamarin prover

EMV standard protocols have been formally studied in [44]. Yet, the analysis the payment data authentication protocols as defined by EMV assumes that the device holding the payment data - in this case, the Chip and PIN card - is honest. This is a reasonable assumptions in this scenario since a payment card provided by the issuer in not an active device connected to a public network, something that is no longer the case when considering a mobile payment application. Critical payment data are held in an environment that could be compromised by a malware in the mobile environment. Hence, we needed to prove the security of our mobile payment application in the context of a compromised device.

One of the main difficulties regarding the security proof of our protocol was embodied by the fact that we rely on the use of counters and counter verification in it. This is a feature that is not supported efficiently by ProVerif. Thus, we used the Tamarin prover tool to process our security proof, adding one more use case to the tool application.

5 Thesis outline

After an introductory chapter on how to model a protocol in the symbolic model, this thesis is divided into two independent parts corresponding to the two use cases we focused on: electronic voting and mobile payment.

Chapter 1 proposes a description of the theory behind the symbolic model ProVerif relies on. As a running example, we apply this theory to one of the standard protocol for card payment data authentication, EMV-DDA.

Our first use case, electronic voting, is the scope of part I of this thesis. We begin by presenting *Belenios Voting-Sheet* (Belenios VS), an improvement of the voting protocol *Belenios Receipt-Free* (Belenios RF) in chapter 2. In order to prove the security of this protocol regarding verifiability, we needed to find a way to automate this proof. Chapter 3 proposes two theorems that presents two different sets of properties easily expressible in ProVerif that imply verifiability when satisfied. Finally, chapter 4 presents the security analysis of Belenios VS regarding verifiability and vote confidentiality.

The second use case we worked on, mobile payment, is the main topic of part II. We begin by describing the landscape of mobile payment industry as it was by the time this thesis was written in chapter 5. Chapter 6 then describes the mobile payment protocol we devised and presented at the European Symposium on Security and Privacy in 2017 along its security formal analysis in chapter 7.

Grammar disclaimer

The reader may have noticed the extensive use of the singular “they”. This is a purely subjective choice to include a gender-neutral pronoun when considering clients, user, voters, attackers...

Chapter 1

Protocol Modeling

When it comes to analyzing the security of cryptographic protocols, symbolic models allow us to model them and formally prove their security - sometimes this proof can even be automated. Moreover, the adversary considered in the formal proofs is an active one. It has full control of the public network and can compute all cryptographic operations used along the protocols and is thus capable of eavesdropping on the public network and interact with actors of the protocol.

However, manually establishing a formal proof is a laborious task. Thankfully, years of research have brought us with a plethora of tools. So... Which one shall we use?

At an industrial level, an attacker would deal with numerous instanciations of the same protocol - e.g. when studying the security of a payment protocol, it is reasonable to hope the payment solution relying on it would be used by a large variety of users, otherwise such a solution is of no use for an aspiring payment service provider. So we need to prove that the security holds for all possible behaviours of a protocol and it has to do so in the presence of an active adversary. This means that our proofs need to guarantee the security for an *unbounded number of sessions*. At the time work on this thesis has been done, only a few of the automated tools available allowed it: ProVerif [19], Scyther [86], TA4SP [25], Maude NPA [57] and a most recent one Tamarin [104].

This thesis focuses on the study of two practical use cases: e-voting and e-payment. Both those type of protocols need to ensure agreement-type properties - e.g. are we sure all entities acting in the protocol are authenticated to one another? - and privacy-type properties - e.g. do all exchanged data provide any crucial information on the user (voting choice, consumption habits...)? Those kind of properties are respectively known as *trace properties* and *equivalence properties*. Out of the three tools formerly evoked, only ProVerif [22] and Tamarin [17] can prove the last kind of properties. Those are the tools we actually used, ProVerif for the e-voting study and Tamarin for the payment one. In this chapter, we will describe the underlying theory behind ProVerif. There are a few differences with Tamarin, however, in essence, a lot of definitions are quite similar and will be explained further on Part II.

ProVerif is an automatic cryptographic protocol verifier for protocols specified in the symbolic model. It is both very flexible regarding the possibilities of cryptographic primitives modeling and able to prove an extensive catalog of security properties (authentication, key agreement, confidentiality, secrecy...) while being quite efficient doing so. This chapter provides an overview of the formal model ProVerif is based on, which is partly inspired by the applied *pi-calculus* [89]. Like the pi-calculus, ProVerif's calculus allows the description of concurrent processes taking channels (and thus network) into account with a quite simple, yet very expressive, language. We begin by depicting the syntax ProVerif uses to describe protocol and then explain the syntax ruling the protocol execution. On the last section, we give some detail about how to express security properties.

In order to ease the reader's comprehension of this chapter, we will use one of the three standard payment card authentication protocols EMV-DDA (EMV Dynamic Data Authentication) [52] as a running example.

1.1 Syntax

A protocol is basically a specification of how two or more entities communicate together. In order to model them, we need a way to represent such entities - by *processes* - and exchanged information - by *terms*. The following section explains how this is done in ProVerif. We construct all messages upon the names, variables and functions used by actors of the protocol and model those ones by processes.

We refer to [21] from which we borrowed our notations and definitions.

1.1.1 Terms

Protocols design how two or more entities exchange information. Those information are sent as message often relying on cryptography. We model cryptographic messages and data by *terms* whose syntax is defined as follows.

We assume several sets. Atomic data - like nonces and cryptographic keys - are the elements of an infinite set of names, $\mathcal{N}$. $\mathcal{V}$ is an infinite set of variables that can be substituted by terms. Both names and variables are declared with their types, elements of the set $\mathcal{T}$. By default, $\mathcal{T}$ includes the types channel (channel names), bool (boolean values) and bitstring (bitstrings).

We also assume a signature, Σ (a finite set of function symbols), split into constructors $\mathcal{C}$ and destructors $\mathcal{D}$: $\Sigma = \mathcal{C} \sqcup \mathcal{D}$, with $\sqcup$ the union set of two disjoint sets. Constructors are used to build new terms. Destructors do not appear in terms, they however manipulate them - to retrieve some previous terms used to generate another one for instance, as stated in Examples 5 (decryption of a ciphertext) and 6 (verification of an electronic signature). All function symbols are declared with their arity and their types: $h(t_1, ..., t_k) : t$ means that the function h takes k arguments of respective types $t_1, ..., t_k$ and return a result of type t. Function symbols may be either public or private. In the first case, the attacker have access to them, in the other no.

Example 5 (asymmetric encryption). A lot of security protocols make use of asymmetric encryption. We can define the signature $\Sigma = \mathcal{C} \sqcup \mathcal{D}$ of asymmetric encryption as follows:

$$\mathcal{C} = \left\{ \begin{array}{rcl} \mathsf{pk}(skey) & : & pkey \\ \mathsf{aenc}(bitstring, pkey) & : & bitstring \end{array} \right\} \quad \mathcal{D} = \left\{ \ \mathsf{adec}(bitstring, skey) : \ bitstring \ \right\}$$

The function pk generates a public key of type $pkey$ from a secret key of type $skey$, this binds the asymmetric private key to the public key. aenc takes a message of type $bitstring$ and a public key of type $pkey$ as inputs and outputs a result of type $bitstring$ (which will be the encrypted message). adec is a destructor that takes a message of type $bitstring$ (the ciphertext) and a private key of type $skey$ as inputs and outputs a result of type $bitstring$ (the decryption of the cyphertext).

Example 6 (cryptographic signature). In order to generate certificate, security protocols heavily rely on cryptographic signature whose corresponding signature Σ can be defined as follows:

$$\mathcal{C} = \left\{ \begin{array}{rcl} \mathsf{spk}(sskey) & : & spkey \\ \mathsf{sign}(bitstring, sskey) & : & bitstring \end{array} \right\} \quad \mathcal{D} = \left\{ \ \mathsf{verify}(bitstring, bitstring, spkey) : \ boolean \ \right\}$$

Similarly to asymmetric encryption, spk generates a public verification key of type $spkey$ from a private signature key of type $sskey$. sign takes a message of type $bitstring$ and a private signature key of type $sskey$ as inputs and outputs a result of type $bitstring$ (which will be the signature of the message). verify is a destructor that takes two entries of type $bitstring$ (the message and its signature) and a verification key of type $spkey$ as inputs and outputs a result of type $boolean$ (true if the signature is valid and false otherwise).

Terms are built over names, variables and constructors. The set of terms built from $\mathcal{N}$ (atomic data) and Σ is denoted $\mathcal{T}(\Sigma, \mathcal{N})$. The grammar of terms is defined in Fig.1.1.

$M, C, ... :==$	terms	$P, Q, ... :==$	processes
x	$x \in \mathcal{V}$	0	nil
n	$n \in \mathcal{N}$	$\mathsf{out}(C, M); P$	output
$f(M_1, ..., M_k)$	$f \in \mathcal{C}$	$\mathsf{in}(C, x : t); P$	input
		$P \mid Q$	parallel composition
$D, ... :==$	expressions	$!P$	replication
M	term	$\mathsf{new}\ n : t\ ; P$	restriction, $t \in \mathcal{T}$
$h(D_1, ..., D_k)$	$h \in \mathcal{C} \sqcup \mathcal{D}$	$\mathsf{let}\ x : t\ = D\ \mathsf{in}\ P\ \mathsf{else}\ Q$	expression evaluation, $t \in \mathcal{T}$
fail	failure	$\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional
		$\mathsf{event}(M); P$	event
$\phi, ... :==$	formulas	$\mathsf{insert}\ T(M_1, ..., M_k); P$	insertion in the table T
$M = N$	equality	$\mathsf{get}\ T(M_1, ..., M_k)\ \mathsf{in}\ P\ \mathsf{else}\ Q$	lookup in the table T
$\phi_1 \wedge \phi_2$	and evaluation		
$\phi_1 \vee \phi_2$	or evaluation		
$\neg\ \phi$	negation		

Figure 1.1: ProVerif syntax and core language

1.1.2 Expression evaluation

Expressions model the data structure and cryptographic computation on terms that can be done by the entities part of the protocol. They are built over terms and both constructors and destructors as described

in Fig.1.1. As previously stated, destructors manipulate terms, we need some kind of rule specification to set how those manipulation are done. Such rules are called *rewriting rules*.

Before going further, we need to define *substitutions*. A substitution $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\Sigma, \mathcal{N})$ is a mapping replacing variables $(x_1, ..., x_k)$ by the corresponding terms $(M_1, ..., M_k)$, commonly denoted $\{M_1/x_1, ..., M_n/x_k\}$.

Expression evaluations are defined through rewriting rules. More precisely if h is a destructor, it is associated to a rewriting rule of the form $h(M_1, ..., M_k) \rightarrow M$ over terms. Note that there can actually be a finite ordered set of rewrite rules in ProVerif. However we only need one for this work in our models.

The expression is then recursively defined as follows:

- $h(D_1, ..., D_k)$ *evaluates* to M
  - if $h \in \mathcal{C}$: if $\forall\, i \in [1, k]$, D_i evaluates to M_i and $M = h(M_1, ..., M_k)$.
  - if $h \in \mathcal{D}$: if there exists a substitution σ such that $\forall\, i \in [1, k]$, $M_i\sigma = M'_i$ and $M = M'\sigma$ where $h(M'_1, ..., M'_k) \rightarrow M'$ is the rewriting rule associated to h.
- $h(D_1, ..., D_k)$ *evaluates* to fail otherwise.

Example 7 (adec rewriting rule). adec is the destructor defined in Example 5. It could be specified with the following rewriting rule:

$$\mathsf{adec}(\mathsf{aenc}(m, \mathsf{pk}(k)), k) \rightarrow m$$

Decrypting a ciphertext generated with the public key associated to the secret key k results in the original message m.

Example 8 (verify rewriting rule). We define the rewriting rule of verify, from Example 6:

$$\mathsf{verify}(m, \mathsf{sign}(m, s), \mathsf{spk}(s)) \rightarrow \mathsf{true}$$

Using the verification key to check the signature of a message signed with the matching signature key s will result in true.

1.1.3 Equations and formulas

ProVerif also allows the definition of cryptographic primitives through an *equational theory*. This comes in handy when a primitive can not be described by a rewrite rule. Actually, Tamarin relies on equational theories to specify the primitives on a way similar to ProVerif.

An equational theory $\mathcal{E}$ is a finite set of equations $M = N$ between two terms M and N of same type that do not contain names.

Example 9 (Equation for verify). The destructor verify, from Example 6 could be defined by the equation:

$$\forall\, m : bitstring,\ s : sskey;\ \mathsf{verify}(m, \mathsf{sign}(m, s), \mathsf{spk}(s)) = \mathsf{true}$$

With the use of equational theory, we can define the equality evaluation between terms as the equality *modulo* the equational theory: $=_{\mathcal{E}}$. It is obtained from the equations by reflexive, symmetric and transitive closure, closure under application of function symbols from Σ and closure under substitution of terms for variables.

With this, formulas can be defined as detailed in Fig.1.1 by extending the equality between terms as expected to $\wedge$, $\vee$ and $\neg$.

1.1.4 Processes

We can now very conveniently model entities playing a part in protocols with *processes*. Most constructs of such processes from Fig.1.1 come from the *pi-calculus* [89].

- The *nil process*, 0, does nothing.
- The *output process*, $\mathsf{out}(C, M); P$, outputs the message M on channel C then executes the process P.
- The *input process*, $\mathsf{in}(C, x : t); P$, on the other hand, inputs a message stored on variable x of type t on channel C then executes P with x bound in P. Input and output processes allow interaction between processes.
- The *parallel composition* of processes P and Q is denoted $P|Q$. It allows processes to run simultaneously.
- The *replication* of the process P for an arbitrary number of time is denoted $!P$. Intuitively, it represents $P|P|P|...|P$.
- The *restriction process*, $\mathsf{new}\ n : t; P$, generates a new name n of type t and then executes P. The name n is *bound* in the process $\mathsf{new}\ n : t; P$.
- The *expression evaluation process*, $\mathsf{let}\ x : t = D\ in\ P\ \mathsf{else}\ Q$, evaluates the expression D. If successful, it stores the evaluation in the term x of type t and executes P, if not, it will execute Q.
- The *conditional*, $\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q$ executes P if the formula ϕ is true, otherwise it executes Q.
- The *event*, $\mathsf{event}(M) : P$, is a special kind of process stating that the process reaches a state with some value M and executes P with no other incidence.
- The *table insertion process*, $\mathsf{insert}\ T(M_1, ..., M_k); P$ insert the record $(M_1, ..., M_k)$ in table T then executes P. The attacker has no writing access to tables.
- The *table lookup process*, $\mathsf{get}\ T(M_1, ..., M_k)\ \mathsf{in}\ P\ \mathsf{else}\ Q$ looks for the record $(M_1, ..., M_k)$ in table T then executes P, otherwise it executes Q. The attacker has no reading access to tables.

When the else branch only consists of 0, we may omit it.

We consider two processes to be equal modulo the renaming of their bound names and variables. $fn(P)$ (resp. $fv(P)$) is the set of *free names* (resp. *free variables*) of P (which are not bound). We say that P is a *closed process* if it does not contain free variables (although it may have free terms). All processes to be verified have to be closed.

Example 10 (EMV Dynamic Data Authentication processes). The EMV protocol suite [51, 52, 53, 54] designed by EMVCo in 1994 is the international security standard for smart payment cards and for payment terminals and automated teller machines . One of its goals is to ensure the card data authentication to payment terminals, to make sure data used for a transaction are valid ones provided by an issuer (the client's bank). This is achieved through three protocols: EMV Static Data Authentication (EMV-SDA), EMV Dynamic Data Authentication (EMV-DDA) and EMV Combined Data Authentication (EMV-CDA). As a running example for this chapter, we chose the EMV-DDA protocol, whose diagram is presented in Fig.1.2.

The payment card holds a pair of RSA private and public signature keys $(\mathsf{ssk}_C, \mathsf{spk}_C)$ and the cardholder information Data. Both the signature verification key spk_C and the cardholder information are

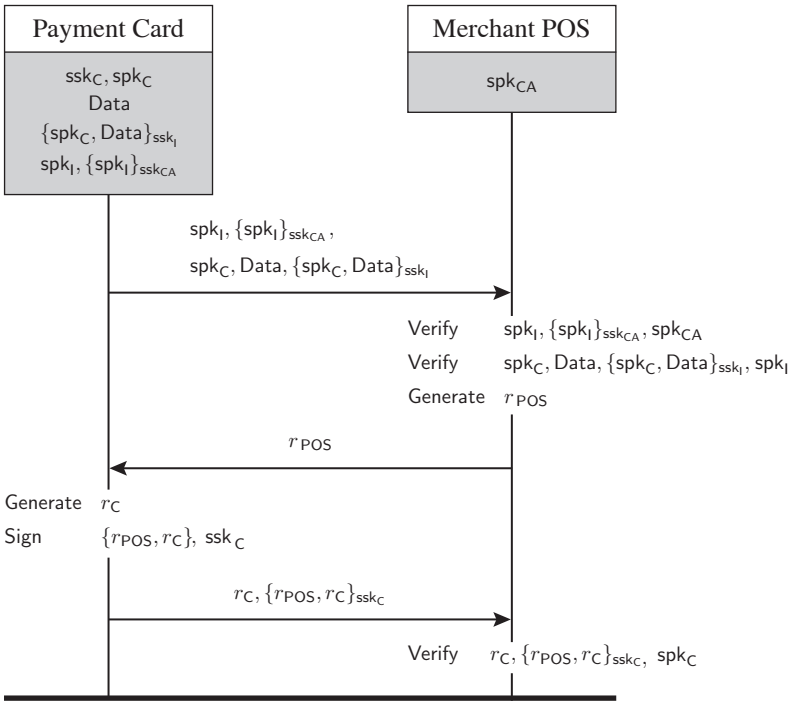


Figure 1.2: The EMV-DDA protocol

signed by the card issuer, with the private key ssk_I. In order to prove the validity of the information it holds, the payment card was also given the issuer verification key's certificate $(\mathsf{spk}_I, \{\mathsf{spk}_I\}_{\mathsf{ssk}_{CA}})$, signed by a certification authority whose verification key spk_{CA} is held by every merchant point of sale (POS).

To process the card authentication, the payment card first sends its public key and the cardholder data with their signature to the merchant terminal as well as the issuer verification key certificate. The merchant terminal can thus process the verification of the issuer verification key and the data and card verification key. It then generates a nonce r_{pos} and sends it to the payment card which will also generate a nonce r_{c} and sign it along with r_{pos}. It will then send r_{c} and the signed nonces to the merchant which will process the verification with spk_C.

The protocol describes in fact the behaviour of two entities: the user's card and the merchant terminal. Both of them can be modeled as processes.

We propose the following process, $\mathrm{ClientCard}$, to model the card:

$$
\begin{array}{l}
\text{let } \mathrm{ClientCard}(\mathsf{ssk_C} : \mathsf{sskey}, \mathsf{Data} : \mathsf{bitstring}, \mathsf{Cert}_{\mathsf{spk_C},\mathsf{Data}} : \mathsf{bitstring}) = \\
\quad \begin{array}{ll}
\text{event} & \mathsf{SEND\_DATA}(\mathsf{spk}(\mathsf{ssk_C}), \mathsf{Data}); \\
\text{out} & \big(\text{public},\ (\mathsf{spk_I}, \mathsf{Cert_I}, \mathsf{spk}(\mathsf{ssk_C}), \mathsf{Data}, \mathsf{Cert}_{\mathsf{spk_C},\mathsf{Data}});\big); \\
\text{in} & \big(\text{public},\ r_{\mathsf{pos}} : \text{rand}\big); \\
\text{new} & r_{\mathsf{c}} : \text{rand}; \\
\text{out} & \big(\text{public},\ (r_{\mathsf{c}}, \mathsf{sign}((r_{\mathsf{pos}}, r_{\mathsf{c}}), \mathsf{ssk_C}))\big).
\end{array}
\end{array}
$$

$\mathrm{ClientCard}$ takes the card's private signing key (ssk_C), the user's payment data (Data) and both values' certificate as inputs ($\mathsf{Cert}_{\mathsf{ssk}_C,\mathsf{Data}}$). The channel "public" is the channel through which the communication is happening. It is under the attacker control. The event $\mathsf{SEND\_DATA}(\mathsf{spk}(\mathsf{ssk}_C), \mathsf{Data})$ is triggered at the beginning of the authentication process to indicate the payment card holding $\mathsf{spk}(\mathsf{ssk}_C)$

and Data started the protocol by sending its information to a POS.

And we propose to model the merchant's payment terminal by the process POS:

$$
\begin{array}{l}
\text{let } \mathrm{POS}() = \\
\quad \text{in} \quad \big(\text{public},\ (\mathsf{spk_I} : \text{spkey}, \mathsf{Cert_I} : \text{bitstring}, \mathsf{spk_C} : \text{spkey}, \mathsf{Data} : \text{bitstring}, \mathsf{Cert}_{\mathsf{spk_C},\mathsf{Data}} : \text{bitstring})\big); \\
\quad \text{if} \quad \big(\mathsf{verify}(\mathsf{spk_I}, \mathsf{Cert_I}, \mathsf{spk_{CA}}) = \mathsf{true} \ \wedge\ \mathsf{verify}((\mathsf{spk_C}, \mathsf{Data}), \mathsf{Cert}_{\mathsf{spk_C},\mathsf{Data}}, \mathsf{spk_I}) = \mathsf{true}\big) \\
\quad \text{then} \quad \text{new } r_{\mathsf{pos}} : \text{rand}; \\
\qquad \text{out} \quad \big(\text{public},\ r_{\mathsf{pos}}\big); \\
\qquad \text{in} \quad \big(\text{public},\ (r_{\mathsf{c}} : \text{rand}, s : \text{bitstring})\big); \\
\qquad \text{if} \quad \mathsf{verify}((r_{\mathsf{pos}}, r_{\mathsf{c}}), s, \mathsf{spk_C}) = \mathsf{true} \\
\qquad \text{then} \quad \text{event } \mathsf{CARD\_AUTHENTICATED}(\mathsf{spk_C}, \mathsf{Data}).
\end{array}
$$

The process POS receives payment data and proceeds to verification. If successful, it starts the nonces exchanges to achieves the card authentication, triggering the event $\mathsf{CARD\_AUTHENTICATED}(\mathsf{spk}_C, \mathsf{Data})$ once the POS is convinced the payment card it interacts with is authentic.

Now that we have specified both processes involved in the protocol, we can model the whole EMV-DDA. We want to prove the security for an unbounded number of users so our main specification needs to generate ssk_C and Data on the fly - with the process new. Public values (such as the certificates) will also be made available on the public network - with out. Finally, we also want to prove the security for an unbounded number of sessions on part of the terminal - so as for the client part, we will need the replication process !. The main process is then:

$$
\begin{array}{ll}
& \text{out} \quad \big(\text{public},\ (\mathsf{spk}(\mathsf{ssk_{CA}}), \mathsf{spk}(\mathsf{ssk_I}), \mathsf{sign}(\mathsf{spk_I}, \mathsf{ssk_{CA}}))\big) \\
| & ! \quad \text{new } \mathsf{ssk_C} : \text{sskey};\ \text{new } \mathsf{Data} : \text{bitstring}; \\
& \qquad \text{out } \big(\text{public},\ (\mathsf{spk}(\mathsf{ssk_C}), \mathsf{Data}, \mathsf{sign}((\mathsf{spk_C}, \mathsf{Data}), \mathsf{ssk_I}))\big); \\
& \qquad \mathrm{ClientCard}(\mathsf{ssk_C}, \mathsf{Data}, \mathsf{sign}((\mathsf{spk_C}, \mathsf{Data}), \mathsf{ssk_I})) \\
| & ! \quad \mathrm{POS}()
\end{array}
$$

1.2 Semantics

Now that we have a way to specify processes and can describe a full protocol, we need to define how the protocol execution actually happens. This is the role of semantics which we will describe in this section.

1.2.1 Semantic configuration

We define a *semantic configuration* as a pair $(E, \mathcal{P})$. E is a pair of two finite sets $\{\mathcal{N}_{\text{pub}}, \mathcal{N}_{\text{priv}}\}$ representing respectively the set of public names (available to an attacker) and the set of private names used so far. We call E the *environment*. $\mathcal{P}$ is a finite multiset of closed processes (processes with no free variables) which contains the processes currently running.

1.2.2 Reduction

The *semantic of processes* is defined by the *reduction* of semantic configurations. To do so, we define reduction relations ($\rightarrow$). Fig.1.3 provides an overview of those reduction rules.

- The *nil reduction rule*, $E, \mathcal{P} \cup \{0\} \rightarrow E, \mathcal{P}$, removes the processes 0 from $\mathcal{P}$ for they do nothing.
- The *parallel composition reduction rule*, $E, \mathcal{P} \cup \{P|Q\} \rightarrow E, \mathcal{P} \cup \{P, Q\}$, adds processes P and Q to $\mathcal{P}$.

$$
\begin{array}{rcll}
E, \mathcal{P} \cup \{0\} & \rightarrow & E, \mathcal{P} & \\
E, \mathcal{P} \cup \{P|Q\} & \rightarrow & E, \mathcal{P} \cup \{P, Q\} & \\
E, \mathcal{P} \cup \{!P\} & \rightarrow & E, \mathcal{P} \cup \{P, !P\} & \\
\{\mathcal{N}_{\text{pub}}, \mathcal{N}_{\text{priv}}\}, \mathcal{P} \cup \{\mathsf{new}\ a : t; P\} & \rightarrow & \{\mathcal{N}_{\text{pub}}, \mathcal{N}_{\text{priv}} \cup \{a'\}\}, \mathcal{P} \cup \{P\{a'/a\}\} & \text{with } a' \notin \mathcal{N}_{\text{pub}} \cup \mathcal{N}_{\text{priv}} \\
E, \mathcal{P} \cup \{\mathsf{out}(C, M); Q,\ \mathsf{in}(C, x : t); P\} & \rightarrow & E, \mathcal{P} \cup \{Q, P\{M/x\}\} & \\
E, \mathcal{P} \cup \{\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q\} & \rightarrow & E, \mathcal{P} \cup \{P\{M/x\}\} & \text{if } D \text{ evaluates to } M \\
E, \mathcal{P} \cup \{\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q\} & \rightarrow & E, \mathcal{P} \cup \{Q\} & \text{if } D \text{ does not evaluates to } M \\
E, \mathcal{P} \cup \{\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\} & \rightarrow & E, \mathcal{P} \cup \{P\} & \text{if } \phi \text{ is true} \\
E, \mathcal{P} \cup \{\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\} & \rightarrow & E, \mathcal{P} \cup \{Q\} & \text{if } \phi \text{ is false} \\
E, \mathcal{P} \cup \{\mathsf{event}(M); P\} & \rightarrow & E, \mathcal{P} \cup \{P\} &
\end{array}
$$

Figure 1.3: Reduction rules in ProVerif

- The *replication reduction rule*, $E, \mathcal{P} \cup \{!P\} \rightarrow E, \mathcal{P} \cup \{P, !P\}$, adds an additional copy of process P. Since reduction rules can be applied again, this rule allows the creation of an unbounded number of copies of replicated processes.

- The *restriction reduction rule*, $E, \mathcal{P} \cup \{\mathsf{new}\ a : t; P\} \rightarrow \{\mathcal{N}_{\text{pub}}, \mathcal{N}_{\text{priv}} \cup \{a'\}\}, \mathcal{P} \cup \{P\{a'/a\}\}$, creates a fresh name a' which does not occur in E and adds it to the set of private names. It then substitutes all occurrences of a on the process P by a'. This rule can be used to generated fresh atomic data such as nonces and keys.

- The *I/O reduction rule*, $E, \mathcal{P} \cup \{\mathsf{out}(C, M); Q,\ \mathsf{in}(C, x : t); P\} \rightarrow E, \mathcal{P} \cup \{Q, P\{M/x\}\}$, allows processes to communicate provided the channels involved in the input and output processes are the same one. The process Q is added in $\mathcal{P}$ as well as the process P where all occurrences of the variable x are substituted by the message M. It basically models the fact that a process outputs the message M on channel C while the process at the other end of the channel receives it.

- The *evaluation reduction rules*, $E, \mathcal{P} \cup \{\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q\} \rightarrow E, \mathcal{P} \cup \{P\{M/x\}\}$ and $E, \mathcal{P} \cup \{\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q\} \rightarrow E, \mathcal{P} \cup \{Q\}$, define the semantics of expression evaluation. If the evaluation of D succeeds, the process evolves as P, otherwise, it evolves as Q.

- The *conditional reduction rules*, $E, \mathcal{P} \cup \{\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\} \rightarrow E, \mathcal{P} \cup \{P\{M/x\}\}$ and $E, \mathcal{P} \cup \{\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\} \rightarrow E, \mathcal{P} \cup \{Q\}$, define the semantics of conditionals. If the formula ϕ is true, P is added to $\mathcal{P}$, otherwise, Q is added to $\mathcal{P}$.

- The *event reduction rule*, $E, \mathcal{P} \cup \{\mathsf{event}(M); P\} \rightarrow E, \mathcal{P} \cup \{P\}$, corresponds to the execution of an event. Nothing is modified.

1.2.3 Trace

An execution of a modeled protocol is represented by a *trace*. It is a sequence of reductions between configurations $(E_0, \mathcal{P}_0) \rightarrow ... \rightarrow (E_n, \mathcal{P}_n)$. We say that a trace triggers the event $\mathsf{event}(M)$ if there

exists a configuration $(E_k, \mathcal{P}_k)$ $(k \in [0, n-1])$ and a process P such that the following reduction appears inside the trace:

$$(E_k, \mathcal{P}_k \cup \{\mathsf{event}(M); P\}) \rightarrow (E_{k+1}, \mathcal{P}_k \cup \{P\})$$

Example 11 (End-to-end execution of the EMV-DDA protocol). As an example of a trace, we provide the one matching a normal execution of the EMV-DDA protocol with the model from Example 10. Note that this trace triggers the event CARD_AUTHENTICATED, so formally, the last step of the trace shall be:

$$E, \mathcal{P} \cup \{\mathsf{event}(\mathsf{CARD\_AUTHENTICATED}(\mathsf{spk}_C, \mathsf{Data})); P\} \rightarrow E, \mathcal{P} \cup \{P\}$$

Fig.1.4 provides this trace.

1.3 Security Properties

We can model protocols using processes with the syntax defined in section 1.1 and their execution with traces defined by the semantics of section 1.2. We now nee a way to model what an attacker is and how to express security properties.

Formal definition of an adversary

In order to capture an extensive set of attacks, our attacker needs to be an active one. Thus, we assume protocols are executed in an untrustworthy network controlled by a Dolev-Yao adversary [47]. This kind of attacker can eavesdrop, intercept, compute and send all messages passing over the public network. In the ProVerif model, the adversary is represented by any process which has been given the set of free names $\mathcal{N}_{\text{pub}}$ as initial knowledge. Note that further on, the adversary can receive any term sent on the public channel.

Formally, we consider $fn(Q)$, the set of free names of a process Q, and the set $\mathcal{C}_{\text{pub}} \subset \mathcal{C}$ of public constructors, we say that the process Q is an $\mathcal{N}_{\text{pub}}$-*adversary* if and only if $fn(Q) \subset \mathcal{N}_{\text{pub}}$ and all function symbols appearing in Q are elements of $\mathcal{C}_{\text{pub}}$.

When running a protocol specification, ProVerif actually executes it in parallel with an $\mathcal{N}_{\text{pub}}$-adversary Q. Which is how the active adversary is part of the security proof.

1.3.1 Trace properties

Lots of properties - such as key agreement or mutual authentication - can be modeled as *correspondence properties* (or *trace properties*) between events which can be informally stated as "if a specific event has been triggered, then other events have been executed". For instance, if we consider what is expected of EMV-DDA, if the merchant POS believes the card data it received to be from an original card, then it is indeed because those data are coming from a real card.

Definition 1. A closed process P_0 satisfies the correspondence

$$\mathsf{event}(M) \rightsquigarrow \bigvee_{i=1}^{m} \bigwedge_{j=1}^{l_i} \mathsf{event}(M_{i,j})$$

against $\mathcal{N}_{\text{pub}}$-adversaries if and only if for some $\mathcal{N}_{\text{priv}}$ with $fn(P_0) \cup fn(M) \cup \bigcup_{i,j} fn(M_{i,j})$, for any $\mathcal{N}_{\text{pub}}$-adversary, for any substitution σ and for any trace tr:

$$tr \text{ triggers } \mathsf{event}(\sigma M) \;\Rightarrow\; \exists\, \sigma',\ \exists\, i \in [1, m] \;\middle|\; \begin{array}{l} \sigma' M = \sigma M \\ \forall\, j \in [1, l_i],\ tr \text{ triggers } \mathsf{event}(\sigma' M_{i,j}) \end{array}$$

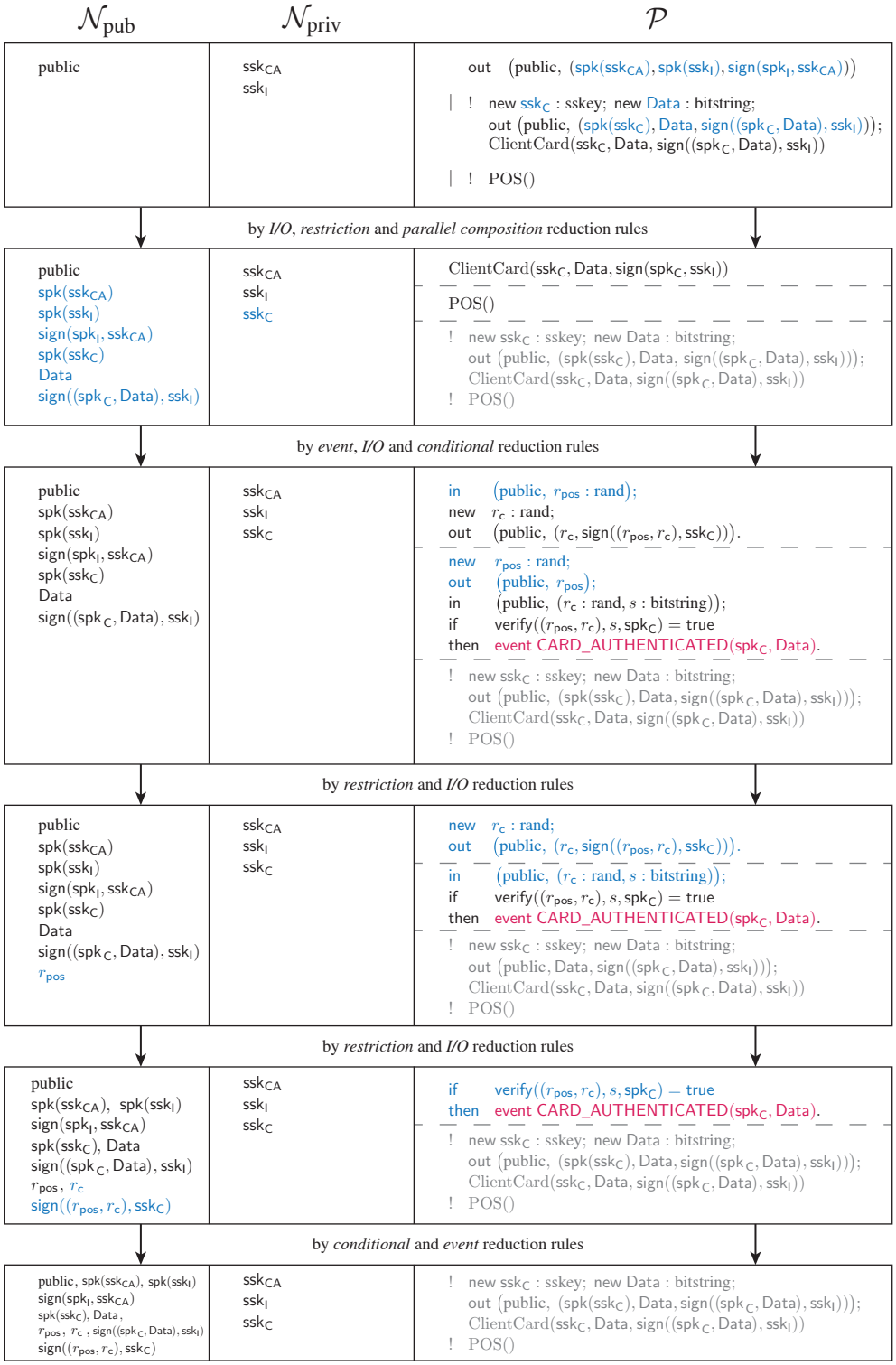


Figure 1.4: A reduction example with EMV-DDA protocol

Example 12 (Dynamic Data Authentication). If a merchant terminal is convinced the card it is talking to is the one holding spk_C and Data, it is indeed this card that shall have sent those data. This is called dynamic data authentication and we can model it with the following correspondence property:

$\mathsf{event}(\mathsf{CARD\_AUTHENTICATED}(\mathsf{spk}_C, \mathsf{Data})) \rightsquigarrow \mathsf{event}(\mathsf{SEND\_DATA}(\mathsf{spk}(\mathsf{ssk}_C), \mathsf{Data}))$

ProVerif actually proves that the EMV-DDA protocol modeled in Example 10 satisfies this property.

1.3.2 Equivalence Properties

Some properties like anonymity, unlinkability or vote privacy cannot be modeled by trace properties. In essence, those properties require that an attacker cannot distinguish two processes from one another. For instance, we could want a payment process to guarantee the users that their consumption habits cannot be tracked by a merchant with the data they use for payment. This would mean that, from a merchant point of view, it would be impossible to make the difference between two payments processed by the same user and two payment processed by different users.

Such properties are called *equivalence properties*. They are useful to model properties like the fact that an adversary can not distinguish two processes from one another. However, in most cases, proving equivalence properties for an unbounded number of sessions is an undecidable problem - in fact, it is the same for trace properties. Most automated tools either prove equivalence properties for a bounded number of sessions or they prove a stronger version of equivalence, the latter being ProVerif's approach - like Tamarin's. ProVerif can prove *diff-equivalence* properties which imply equivalence. Note that Tamarin essentially implements a variant of ProVerif's diff-equivalence using similar notation and specification constructions.

In a nutshell, along this thesis, all equivalence properties we want to express - voting secrecy and payment unlinkability - are equivalence between two instanciations of a same process that only differ by terms. This kind of equivalence can actually be expressed as diff-equivalence properties.

Biprocesses

To express diff-equivalence, ProVerif represents processes that differ only by their terms as *biprocesses*. The grammar for biprocesses is actually an extension of the grammar for processes defined in Fig.1.1 with the addition of two cases: $\mathsf{diff}[M, M']$ for terms and $\mathsf{diff}[D, D']$ for expressions.

A *biprocess* P defines two processes:

- $fst(P)$: occurrences of $\mathsf{diff}[M, M']$ (resp. $\mathsf{diff}[D, D']$) in P are replaced by M (resp. D).
- $snd(P)$: occurrences of $\mathsf{diff}[M, M']$ (resp. $\mathsf{diff}[D, D']$) in P are replaced by M' (resp. D').

$fst(T)$, $snd(T)$, $fst(F)$, $snd(F)$, with T term and F expression are similarly defined. We also extend this definition to multisets of processes and configurations ($fst(E, \mathcal{P}) = E, fst(\mathcal{P})$ and $snd(E, \mathcal{P}) = E, snd(\mathcal{P})$).

The semantics for biprocesses is quite similar to the semantics of processes described in Fig.1.3, but differs in some points described in Fig.1.5. In essence, a reduction rule is processed only if it can be processed for both components $fst(...)$ and $snd(...)$ of the biprocess. We then say that both components *reduce in the same way*.

- Regarding the *I/O reduction rule*, a communication cannot happen unless the channels for inputs and outputs are the same for both components of channels C and C'.
- The *expression evaluation* succeeds (resp. fails) if it succeeds (resp. fails) for both components.
- The *conditional reduction rules*, are omitted for they can be encoded as evaluation rules.

$$E, \mathcal{P} \cup \{\mathsf{out}(C, M); Q,\ \mathsf{in}(C', x : t); P\} \rightarrow E, \mathcal{P} \cup \{Q, P\{M/x\}\} \quad \text{if } fst(C) = fst(C') \text{ and if } snd(C) = snd(C')$$

$$E, \mathcal{P} \cup \{\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q\} \rightarrow E, \mathcal{P} \cup \{P\{\mathsf{diff}[M, M']/x\}\} \quad \text{if } fst(D) \text{ evaluates to } fst(M) \text{ and if } snd(D) \text{ evaluates to } snd(M)$$

$$E, \mathcal{P} \cup \{\mathsf{let}\ x = D\ \mathsf{in}\ P\ \mathsf{else}\ Q\} \rightarrow E, \mathcal{P} \cup \{Q\} \quad \text{if } fst(D)\text{'s evaluation fails and if } snd(D)\text{'s evaluation fails}$$

$$E, \mathcal{P} \cup \{\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\} \quad \text{omitted rule}$$

Figure 1.5: Changes in semantics for biprocesses in ProVerif

Diff-Equivalence

We can extend the notion of reducing in the same way to configurations: a configuration $\mathcal{C}$ reduces to $\mathcal{C}'$ ($\mathcal{C} \rightarrow \mathcal{C}'$) if $fst(\mathcal{C}) \rightarrow fst(\mathcal{C}')$ and $snd(\mathcal{C}) \rightarrow snd(\mathcal{C}')$. When the components $fst(\mathcal{C})$ and $snd(\mathcal{C})$ do not reduce in the same way, we say that the configuration $\mathcal{C}$ *diverges*.

Definition 2. A closed biprocess P_0 satisfies diff-equivalence with the set of private names $\mathcal{N}_{\text{priv}}$ if and only if for some $\mathcal{N}_{\text{pub}}$ disjoint from $\mathcal{N}_{\text{priv}}$ with $fn(P_0) \subset \mathcal{N}_{\text{pub}} \cup \mathcal{N}_{\text{priv}}$, for any $\mathcal{N}_{\text{pub}}$-adversary Q, there is no configuration $\mathcal{C}$ such that $(\mathcal{N}_{\text{pub}}, \mathcal{N}_{\text{priv}}), \{P_0, Q\}$ reduces to $\mathcal{C}$ and $\mathcal{C}$ diverges.

Example 13 (Payment unlinkability). We define the payment unlinkability as the incapacity for the attacker to distinguish two payment sessions done by the same client A from two payment sessions done by two different users A and B. It is usually expressed as the following equivalence property:

$$\mathrm{PaymentSession}(A) \mid \mathrm{PaymentSession}(A) \approx \mathrm{PaymentSession}(A) \mid \mathrm{PaymentSession}(B)$$

This property is indeed an equivalence property between two instanciations of the same process $\mathrm{PaymentSession}$, thus, it can be expressed as a diff-equivalence property with the biprocess:

$$\mathrm{PaymentSession}(A) \mid \mathrm{PaymentSession}(\mathsf{choice}[A, B])$$

If we use the processes defined in Example 10, the whole $\mathrm{PaymentSession}$ process would be:

$$\begin{array}{ll} & \mathsf{out}\ \ (\mathsf{public},\ (\mathsf{spk}(\mathsf{ssk}_{CA}), \mathsf{spk}(\mathsf{ssk}_I), \mathsf{sign}(\mathsf{spk}_I, \mathsf{ssk}_{CA}))) \\ \mid & \mathsf{out}\ \ (\mathsf{public},\ (\mathsf{spk}(\mathsf{ssk}_A), \mathsf{Data_A}, \mathsf{sign}((\mathsf{spk}_A, \mathsf{Data_A}), \mathsf{ssk}_I))) \\ \mid & \quad\ \ \mathrm{ClientCard}(\mathsf{ssk}_A, \mathsf{Data_A}, \mathsf{sign}((\mathsf{spk}_A, \mathsf{Data_A}), \mathsf{ssk}_I)) \\ \mid & !\ \ \mathrm{POS}() \end{array}$$

More precisely, the payment session instanciation depends on the subprocess $\mathrm{ClientCard}$ and its parameters - the card's RSA key, ssk_A, the user's data, $\mathsf{Data_A}$, and the payment data certificat, $\mathsf{sign}(...)$ - to which choice will be applied.

In the ProVerif calculus, the EMV-DDA biprocess expressing the unlinkability property would thus be the following one:

$$\begin{array}{ll} & \mathsf{out}\ \ (\mathsf{public},\ (\mathsf{spk}(\mathsf{ssk}_{CA}), \mathsf{spk}(\mathsf{ssk}_I), \mathsf{sign}(\mathsf{spk}_I, \mathsf{ssk}_{CA}))) \\ \mid & \mathsf{out}\ \ (\mathsf{public},\ (\mathsf{spk}(\mathsf{ssk}_A), \mathsf{Data_A}, \mathsf{sign}((\mathsf{spk}_A, \mathsf{Data_A}), \mathsf{ssk}_I))) \\ \mid & \mathsf{out}\ \ (\mathsf{public},\ (\mathsf{spk}(\mathsf{ssk}_B), \mathsf{Data_B}, \mathsf{sign}((\mathsf{spk}_B, \mathsf{Data_B}), \mathsf{ssk}_I))) \\ \mid & \quad\ \ \mathrm{ClientCard}(\mathsf{ssk}_A, \mathsf{Data_A}, \mathsf{sign}((\mathsf{spk}_A, \mathsf{Data_A}), \mathsf{ssk}_I)) \\ \mid & \quad\ \ \mathrm{ClientCard}(\mathsf{choice}\ [\ (\mathsf{ssk}_B, \mathsf{Data_B}, \mathsf{sign}((\mathsf{spk}_B, \mathsf{Data_B}), \mathsf{ssk}_I))\ ,\ (\mathsf{ssk}_A, \mathsf{Data_A}, \mathsf{sign}((\mathsf{spk}_A, \mathsf{Data_A}), \mathsf{ssk}_I))\ ]\ ) \\ \mid & !\ \ POS() \end{array}$$

The subprocess $\mathrm{ClientCard}$ outputs the certified payment data during the firs exchange of the protocol. Those values are publicly known and static, so an attacker can differentiate the payment sessions performed by two different users.So unsurprisingly, EMV-DDA does not guarantee payment unlinkability.

Conclusion

This introduction chapter provided an overview of protocol modeling and security proof in the ProVerif calculus. It allows to model a large variety of protocols in a rather simple language and efficiently express and prove security properties - whether they are *trace* properties or *equivalence*.

The first part of this thesis - voting protocols - will actually use ProVerif to prove the security of a voting scheme with different corruption scenarii.

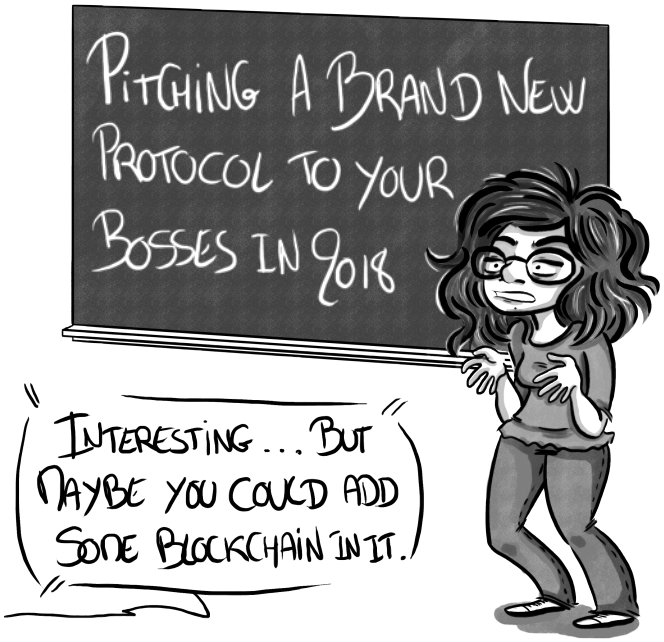
Although we used the Tamarin prover for the second part of this thesis, the calculus presented here will still be useful since the protocol modeling in the Tamarin calculus is quite close.

Part I

Voting Protocol

Chapter 2

Belenios VS



The last 20 years have been prolific for the democratization of electronic voting. Whether the technology was used for low key elections - unions, student government - or higher stakes elections - some US states allow the use of voting machine and Estonia relies on electronic voting for its presidential election - we see more and more voting schemes in our modern landscape. *What can we expect from an electronic voting scheme regarding its security?*

We could expect it to ensure *vote confidentiality*. It should be impossible to guess the voter's vote. In a classical election, this is guaranteed by the use of voting booth - where the elector can secretly choose their vote - and normalized ballots - all ballot sheets are the same for everyone as well as all ballot envelopes.

For high-profile elections, we could even expect a voting scheme to be *coercion resistant*: the vote confidentiality shall never be broken even if the voter *wants* to prove to someone else they voted for a specific candidate. In a physical election, such as the French ones, this is guaranteed by the fact that all ballots do not display any distinctive feature. Otherwise, the ballot is considered as invalid.

And what about the voters' *confidence* in the election they are participating to?

Indeed, a voter could expect that their vote is indeed going to be part of the final tally. If we pursue

with our analogy with the physical French election, this would be the equivalence of having a transparent ballot box that can be watched by *anyone*. A voter could for instance stay until the tally of the election to make sure that their ballot is going to be counted in the final result. This security guarantee is called the *individual verifiability* of an election.

We could even ask more from a voting scheme by requiring it to be *universally verifiable* and to ensure the *eligibility* of voters participating to an election: the final result shall only take eligible voters' votes into account and the result shall be verifiable by *anyone*.

Those two kind of properties that we will respectively call *privacy* and *verifiability* properties, appear as quite antagonistic considering an electronic voting scheme: we want the voter's vote to remain confidential, even if the voter wants to disclose it, no one shall learn how they voted. Yet we also want our protocol to allow users to know whether or not their own vote is going to be part of the final tally. A compromise between those two properties is then necessary, and modern electronic voting schemes reflect this.

Finally, a voting scheme shall remain *accessible and available*. The voter shall not need any particularly technical skill to cast their vote or to be convinced that their vote was taken into account.

The first part of this thesis focuses on specifying an electronic voting protocol that guarantees both verifiability and privacy even if it is performed using compromised devices from the user point of view or if the election administrators are rogue. It allows online Internet voting, we call it Belenios Voting Sheet (Belenios VS).

The protocol itself is a variant of an existing voting scheme, Belenios Receipt-Free (Belenios RF) [29], which is itself an extension of Belenios [38], which was built upon Helios [3, 4]. We first depict the evolution from Helios to Belenios RF by providing an overview of what these protocols have to offer and what are their limits regarding nowadays voting schemes landscape. We then provide the specification of our protocol and what trust assumptions we can or cannot make about it and we finish this chapter by stating our security claims for Belenios VS.

2.1 Combining verifiability and privacy in voting schemes: from Helios to Belenios RF

The first part of this thesis will focus on specifying an electronic voting protocol that guarantees both verifiability and privacy even if it is performed using compromised devices from the user point of view or if the election administrators are rogue. It allows online internet voting.

The protocol itself is a variant of an existing voting scheme, Belenios Receipt-Free (Belenios RF) [29], which is itself an extension of Belenios [38], which was built upon Helios [3, 4]. This chapter will depict the evolution from Helios to Belenios RF by providing an overview of what these protocols have to offer and what are their limits.

2.1.1 Helios: a web-based open-audit voting scheme

Helios [3, 4] is presented as "the first web-based open-audit voting system" and was launched in 2008. From a voter point of view, it can be run from a modern web browser. The voter computes their vote from their browser and cast it to a voting server after an authentication. The bulletin board is publicly accessible. At the end of the tally process, the election result can be audited by anyone by relying on publicly available data.

Helios was used several times to run real-world elections such as the election of the University of Louvain-la-Neuve president from 2010 to 2012 or the International Association for Cryptographic Research board members since 2011. This protocol has the advantage of its simplicity and was designed

to achieve both verifiability and privacy. The following section provides some insights about the way Helios works and its security features.

An overview of the Helios voting scheme

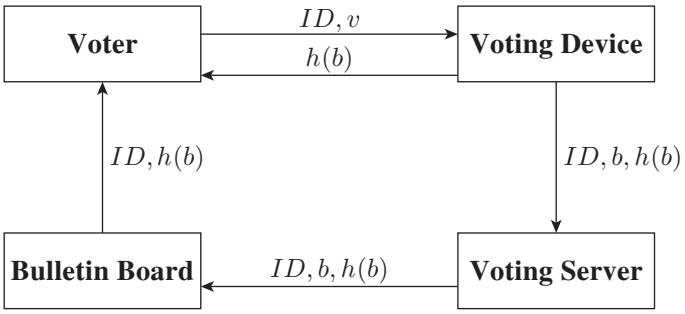
The first version of Helios technically relied on the combination of two cryptographic protocols: the ballot casting approach from Benaloh [18] and the Sako-Killian mixnet [103]. The first one proposes a rather simple way to obtain verifiable elections by following a specific scheme during ballot casting while the second one focuses on the tally part by providing a provable shuffling of the ballot box to ensure privacy and decrypt all ballots one by one.

The current version of Helios - Helios v.4 - does not rely on a mixnet for tally anymore. Instead, it relies on homomorphic encryption: the whole result is obtained from the global decryption of ballots rather than an individual decryption, an operation that is made possible thanks to an inherent property of the El Gamal asymmetric encryption scheme Helios relies on to encrypt the ballots. This kind of tally has the advantage of requiring only one server - instead of several for the good tally execution. Like Helios v.1, the tally is output with a proof of good computation.

We have two main phases during an election: the voting phase during which electors cast their ballots and the tallying phase which computes the election result.

The main entities interacting in the voting scheme are the *election administrator* who sets the election up, the *voter* and their *web browser*, the *voting server* which administrates the *public bulletin board* and the *tallying authority* in charge of tallying the election - this last entity could be a group of several tallying authorities, by sharing the election secret key among several entities using a threshold cryptosystem.

The voting scheme is summarized in Fig.2.1. $\mathsf{aenc}(m, k)$ is the asymmetric encryption of message m with private key k



$b := \mathsf{aenc}\big((v, r), \mathsf{pk_e}\big), \sigma_b$

Figure 2.1: Helios voting scheme

- **Setting up the election**

  The tallying authority generates the election asymmetric pair of keys ($\mathsf{sk_e}$ and $\mathsf{pk_e}$). As usual in the electronic voting world, we can used a threshold cryptosystem so that the tallying authority include several trustees among which the decryption key is split. For the sake of readability, we consider the tallying authority as one single entity.

  The public key is sent to the election administrator who will publish it. It also sets up and publishes the set of eligible voters (VL) and valid votes ($\mathcal{V}$). The election administrator provides the list of

eligible voters to the voting server who generates authentication credentials - typically a login and password pair - for each one of them.

The authentication credentials are transmitted to the voters and the election web page is publicly displayed as well as the bulletin board.

- **Casting and verifying a ballot**

  An eligible voter (ID) connects through their browser to the election page. There, a web service allows them to compute as many ballot as they want for the election. A ballot (b) is the encryption of a valid vote (v) and a randomly generated nonce (r) with the election public key ($\mathsf{pk_e}$). Along this ciphertext comes the *zero-knowledge proof*[1] (σ_b) that the decryption of b is a valid vote ($v \in \mathcal{V}$):

  $$b := \mathsf{aenc}\big((v, r), \mathsf{pk_e}\big), \sigma_b$$

  Each time a ballot is produced, the web service commits to the vote encryption by producing a hash ($h(b)$) of the ballot (b). There, the voter has two options:

  – Either they *cast* their ballot b.
  – Or they *audit* the ballot. If so, the web service displays the nonce r that was used to compute b so the voter can check if the encryption of their vote with r matches the ballot b with a third-party server. If the voter is satisfied, then a new ballot b' is computed with another nonce r' and the voter chooses again between casting or auditing the ballot.

  The voter is given their ballot and the hash of their ballot.

  Once the voter chooses to cast their ballot, they provide their authentication credentials to the voting server through the election web service. Both the nonce and the voter's voting choice are discarded by the web service. The voting server displays the ballot and its hash on the public bulletin board along the voter's name (or an identifier):

  $$\big(ID, h(b)\big)$$

  The verification process is then rather simple. A voter just looks for their name (or identifier) in the public bulletin board and checks their ballot is displayed.

- **Homomorphic tally of the bulletin board**

  Once the election administrator declares the voting phase is over, the tallying authority retrieves all original ballots from the bulletin board. All hashes and zero-knowledge proofs are individually checked for each ballot.

  The El Gamal encryption scheme allows the homomorphic combination of all ballots: instead of decrypting each ballot individually, they are combined altogether to produce the general encryption of the tally. The result of this computation is then decrypted partially by each one of the entities part of the tallying authorities, all of them producing a proof of good computation for their partial decryption. The result is obtained after the tallying authority has finished the decryption, along its proof of good computation.

  The tallying authority publishes the result along the proof of good computation. Any wannabe election auditor can download the election public data - eligible voters list, original ballots and tally outputs - to verify the election was correctly performed.

[1] A zero-knowledge proof is a method allowing a *prover* to prove to a *verifier* that a statement is true. In our case, the web service is the prover and convinces the verifier - the voting server and the public - that the decryption of b in a valid vote as defined by the election administrator.

Security strength of the protocol

The first and main appeal of Helios is the fact that it is a *simple* and *accessible* election scheme from the voter point of view. Indeed, no technical skill is required from the voter to cast a ballot: the web service on the browser is supposed to compute the ballot itself and the verification process can be done by simply processing a "search" command for the user's name and/or their ballot's hash on the public bulletin board.

This verification process allows for individual verifiability. Each voter has the proof that their ballot is displayed. Moreover, thanks to the proof of good computation on the election result, a voter - or an auditor - can check that a specific ballot was part of the final result. Helios' s verifiability was formally proven in [42].

The protocol is also private, thanks to the homomorphic tally.

The limits of Helios

Yet, some few limitations can be observed. As stated by [3], Helios should only be used for low-key elections, for it is not coercion resistant - a user can prove for who they voted, as long as they can figure out from the web service page, what nonce was used to encrypt their ballot, or use another tool to compute and cast their ballot.

But the main limitation of Helios comes from the *attacker model* that was used for the security analysis. Indeed, the voter's device and the voting server are both assumed to be honest.

If the voter's device is compromised, so is the voter's privacy, for the voter's device "sees" what candidate the user chose. Even more, if during the voting phase the voter audits their vote with the same - rogue - device, an attacker could even encrypt another voting option and make the user believe that they voted for their choice.

If the voting server - and thus the bulletin board it manages and displays - is rogue, Helios is also vulnerable regarding verifiability. Indeed, what would prevent a rogue voting server from stuffing the ballot box? Helios mitigates this from happening by relying on the fact that the voter's name - or alias - is displayed with their ballot and that the list of voters - or aliases - authorized to cast a vote is public. It allows an auditor to check whether all displayed ballots were cast in the name of a honest voter. However, if an eligible voter decided not to participate to the election, there is no way to effectively prevent a rogue voting server from registering a vote for them. Plus, in some countries, such as France, whether a voter participated or not to an election is a private information. It would be the voter's identifiers provided by the voting server that would be displayed on the bulletin board, thus preventing even more control over it.

Finally, since the voter's name is displayed along its ballot, when cryptography will be broken, in say, twenty years, anyone could learn the vote of each participant. *Everlasting privacy* is not guaranteed of the real name of voters is displayed on the bulletin board - or if aliases used to vote can be kinked to voters.

2.1.2 Belenios: strengthening Helios' verifiability with credentials

Belenios [38] is a voting scheme built upon Helios that was proposed by Véronique Cortier, David Galindo, Stéphane Glondu and Malika Izabachène. It addresses both the problem of potentially displaying the voter's identity along their ballot and the fact that there is no efficient control over the voting server by adding another entity to the whole Helios voting ecosystem: a *registrar*.

The registrar provides each voters with cryptographic signature credentials. It creates one credential per eligible voter and provides them to the voter. Each voter signs their ballot and cast them. The ballot are displayed along the public signing credential on the bulletin board.

We propose an overview of the Belenios voting protocol, explicitly stating the differences with Helios and discuss its security features in the following section.

Transforming a verifiable voting scheme into a verifiable voting scheme with weaker trust assumptions

The main entities participating to the Belenios voting scheme are almost the same as in Helios. The election administrator sets the election up, voters participates to the election through their web browser, the voting server provides each eligible voters authentication credential to cast a ballot and the tallying authority - or authorities, if a threshold cryptosystem is used - is in charge of computing the election result.

A new entity is introduced to this ecosystem: the registrar. In addition to their authentication credentials received from the voting server, voters receive *voting credentials* from the registrar - provided *via* a different channel than the one that was used to provide the authentication credentials. The voting credentials are a pair of asymmetric encryption key that are used by the voter through their device to sign their ballot.

[38] proposes a generic construction that transforms any voting scheme that is verifiable under the assumption that both the voting server and the registrar are honest - or, if there is no registrar, that the voting server is honest - into a voting protocol that is verifiable under the assumption that *at least one* of both entities is honest.

Belenios relies on El Gamal encryption to encrypt the votes [48], like Helios, and on Schnorr signatures to sign the ballots [105].

An overview the Belenios protocol

Belenios protocol is summarized in Fig.2.2.

- **Setting up the election**

  The election setup is exactly the same as in Helios with the addition of the voting credentials provisioning.

  For each eligible voter, the registrar generates a pair of signature keys: a signing private key (ssk) and a public signature verification key (spk). All credentials are *personal* meaning that they are all different from one another. They are provisioned to each eligible voter over a different channel than the one used by the voting server to provision the authentication credentials.

- **Casting and verifying a ballot**

  Similarly to Helios, an eligible voter connects through their browser to the election page. They compute their ballot *via* a web service. With the exception that this time, the ballot (b) computed with a nonce (r) and its comes with a signature (s_b) of the ballot along its zero-knowledge proof (σ_b):

  $$b := \mathsf{aenc}\big((v, r), \mathsf{pk_e}\big), \sigma_b$$
  $$s_b := \mathsf{sign}\big((b, \sigma_b), \mathsf{ssk}\big)$$

  As in Helios, the voter is given the hash of their ballot $h(b)$.

  The voter casts their ballot to the voting server after authenticating themself. The voting server will display the ballot as follows;

  $$\big(\mathsf{spk}, h(b), s_b\big)$$

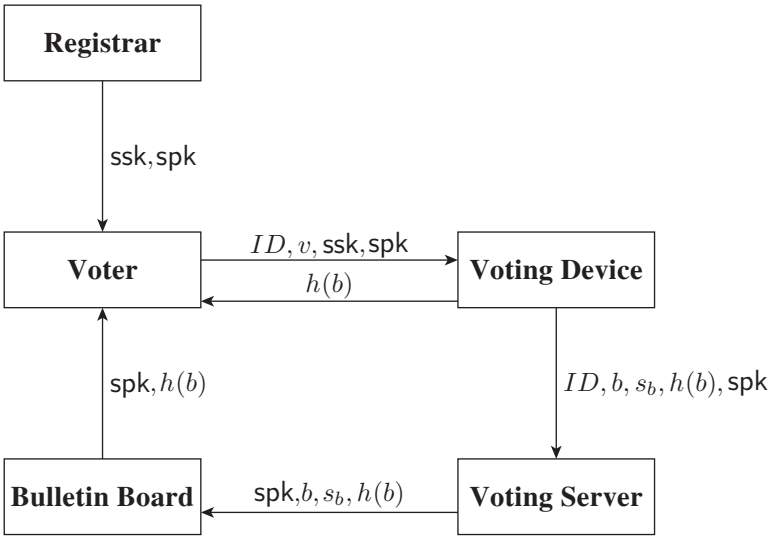


Figure 2.2: Belenios voting scheme

The voter verifies their vote by looking for the hash of their ballot $h(b)$.

- **Homomorphic tally of the bulletin board**

  The tally process is essentially the same as the one from Helios. The main difference comes from the fact that since each ballot is signed with a different signature key, there is no duplicate ballot and thus no weeding process needed.

  The tallying authority proceeds to a homomorphic tally and outputs the result as well as a proof of good computation.

  The election result can be audited with all public data available. An auditor has access to the number of eligible voters from the election administrator, the list of valid public credentials from the registrar and the public bulletin board displayed by the voting server. With those parameters,the auditor can check the consistency of the bulletin board - by making sure each ballot was cast with a valid public credential and that the total number of valid public credentials does not exceed the number of eligible voters.

A verifiable protocol vulnerable regarding privacy

Belenios was formally proven as verifiable under the assumption that *at most* either the voting server or the registrar is dishonest.

As Helios, the protocol is not coercion-resistant as the voter can prove how they voted. Plus, if the voter's device is compromised, so is the vote confidentiality, for the device sees what the voter votes for and computes the ballot. Plus, the voting audit process is not implemented by Belenios' original code. If the voting device is dishonest and if the voter does not have a way to audit their vote, as in Helios, the verifiability could be compromised, for the device could encrypt another vote than the voter's choice.

2.1.3 Belenios Receipt-Free: adding ballot randomization to achieve strong receipt-freeness

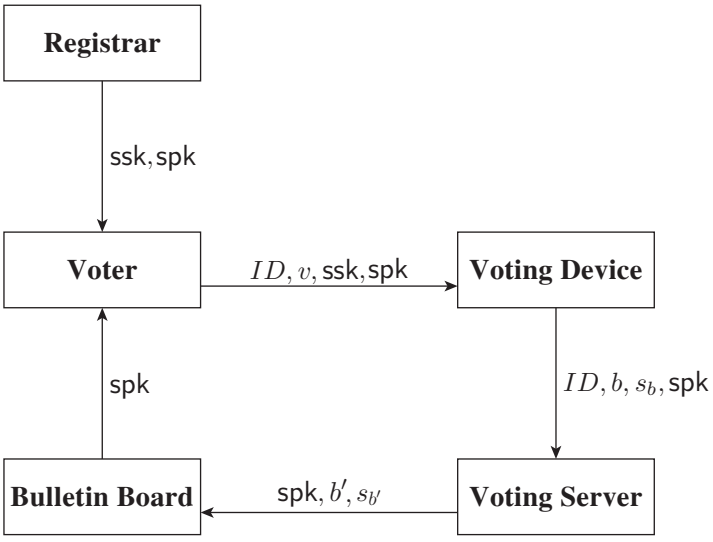
Belenios Receipt-Free (Belenios RF) [29] is a variant of Belenios addressing the non receipt-freeness of the protocol. It is structurally built as Belenios, with the same entities and quite the same messages, the main difference being the cryptographic primitives it is built on: randomizable ciphertexts and signatures [23].

Adding randomization for a receipt-free protocol

Belenios RF relies on cryptographic primitives that form an asymmetric encryption and a signature schemes that produce signed ciphertexts randomizable by anyone. Ballots are computed as the (randomizable) encryption of a vote along a zero-knowledge proof that the encrypted vote is a valid one. A (randomizable) signature of the ballot is also provided, as in Belenios. Yet, without knowing the election private key or the voter's private signing key, anyone can randomize the ballot by producing a new ciphertext and its zero knowledge proof and a new signature that is valid for the new ballot. Details about how this randomizable encryption scheme works are given in further section 2.2.2.

The possibility to randomize the ballots is used during the publication of the voter's ballot. Instead of displaying the original ballot that the user computed from their machine, the voting server randomizes the ballot cast by the voter and displays the new ballot.

As we can see from Fig.2.3, Belenios RF structurally relies on the same exchanges between all entities. The setup, voting and tally phases are the same and to verify their vote, the user still only needs to look for their public signature verification key.



$b := \mathsf{aenc}\big((v, r), \mathsf{pk_e}\big), \sigma_b$

$s_b := \mathsf{sign}\big(b, \mathsf{ssk}\big)$

$b', s_{b'}$: *randomization of* b *and* s_b

Figure 2.3: Belenios RF voting scheme

Because the voter has no control on the randomization happening on part of the server, they cannot

prove what was their voting choice or can lie about it. This is what guarantees the *receipt-freeness* of the protocol.

Limits of Belenios RF

Yet, the protocol is not exactly coercion resistant, for a voter could sell their voting credential to the highest bidder even if they cannot prove for whom they voted. Moreover, if the voting server is under the attacker control, it could display non randomized ballots thus bypassing the receipt-freeness policy.

2.1.4 Motivations to improve Belenios RF

In this section, we presented the evolutions between several voting schemes over time.

Helios is a web-based voting protocol that claims to be verifiable [42] and private. It has the strong advantage of being accessible from the user's point of view: a voter connects on the election web page, computes their ballot through their browser and cast it after authenticating themself to the voting server - usually with login/passwords. However, the verifiability of the protocol relies on the strong assumption that the voting server, the entity in charge of computing and displaying the bulletin board, is honest. Plus, Helios is not coercion resistant as the voter can prove what was their voting choice. Finally, if the device of the voter is compromised, so are verifiability and privacy.

Then came Belenios, a protocol built from Helios that requires the presence of a registrar in the voting ecosystem. The voter is given voting credentials - a pair of signature keys - to compute and sign their ballot. Belenios guarantees verifiability under the weaker trust assumption that at most either the registrar or the voting server is corrupted. Yet, Belenios still does not address the problem of coercion. Moreover, if the voter's device is corrupted the protocol's privacy is still vulnerable, and if the voter does not audit their ballot, so can be the verifiability.

Belenios RF answers partially to the coercion problem by proposing a receipt-free protocol. Thanks to the use of randomizable cryptographic primitives, a voter cannot prove what they voted for - while still having the possibility of selling their voting credentials. Although Belenios RF guarantees verifiability under the same assumptions as Belenios, it still need the voter's device to be secure to ensure privacy and verifiability. Regarding verifiability, it actually is more vulnerable than Belenios, for the voter cannot check the ballot they cast matches the one they computed since ballots are randomized before being displayed on the bulletin board. A rogue voter's device could cast the wrong ballot to the bulletin board and a voter would not be able to detect that. Thus, leaving the computation of ballots to a voter's device does not seem to be the best design choice. The fact that the voter's device is not compromised appeared to us as a strong security assumption. Building from Belenios RF, we wanted to propose a protocol variant that would have at least the same security guarantees *but* that would still be secure - private and verifiable - even if the voter's device is under an attacker control.

2.2 Presentation of the protocol

We devised a voting scheme stemming from Belenios [38] and Belenios Receipt-Free [29]. However, instead of letting the voter encrypt their vote from a device that could be corrupted - which could result in a loss of vote confidentiality - we rely on the use of *voting sheets*. The voter will scan their pre-encrypted vote from the voting sheet to cast their ballot. This allows us to ensure privacy even if the device of the user is corrupted - a very plausible scenario - and against a rogue voting server. Furthermore, the voter can easily verify that their ballot is actually going to be part of the final tally.

This section exposes the motivation for our voting scheme, which entities take part on it, what are the cryptographic primitives behind it and finally what is the protocol ruling our voting scheme.

2.2.1 Election ecosystem, entities and voting material

Several entities interact in our voting scheme and produce or use the voting material essential to the whole election process. Fig.2.4 summarizes the election ecosystem and will help the reader understanding the role of each one of the eight entities.

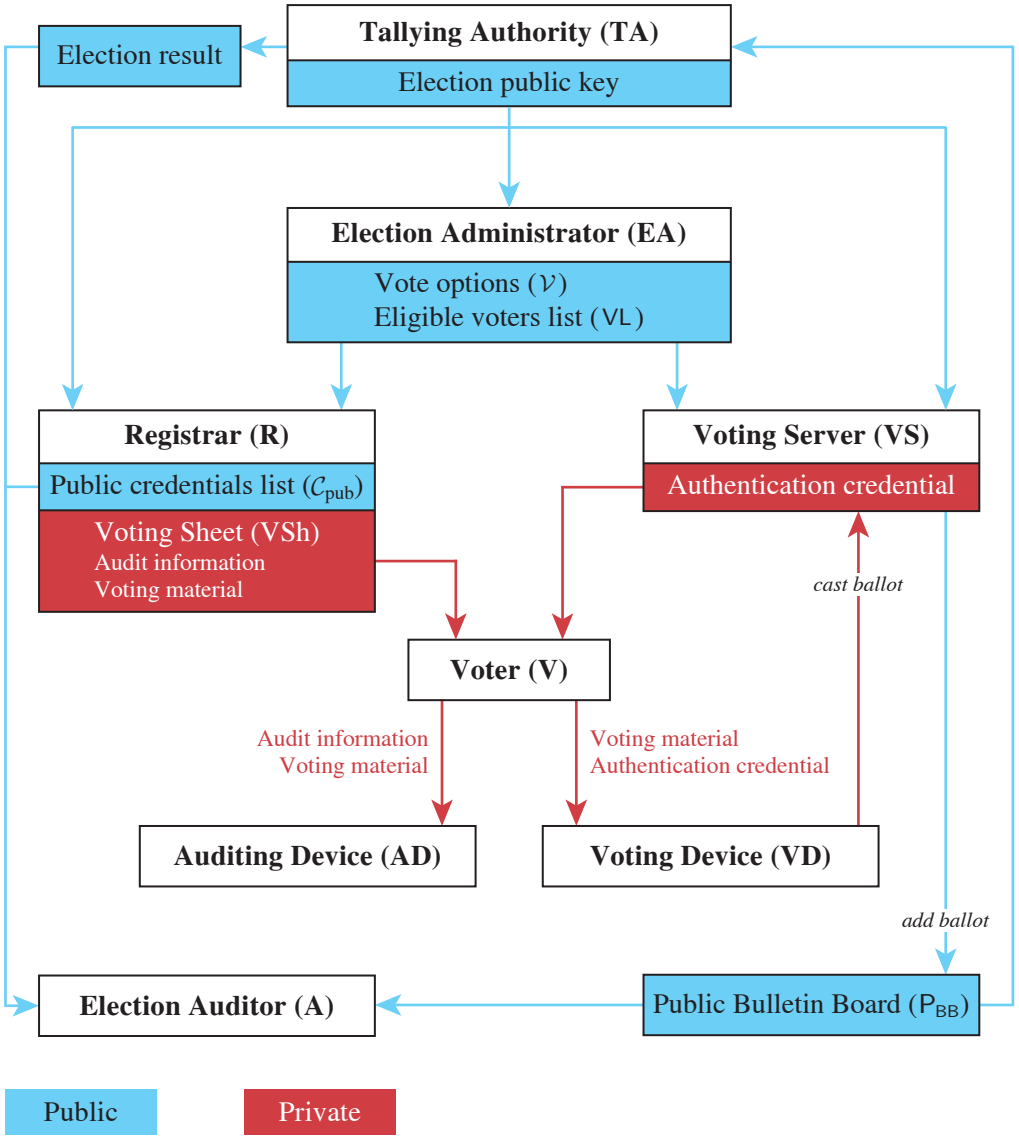


Figure 2.4: Our election ecosystem

Tallying Authority (TA)

The *tallying authority* is in charge of tallying the result of the election once the voting phase is over. To this purpose, it sets up the election asymmetric keys (sk_e and pk_e) and provides the public key to

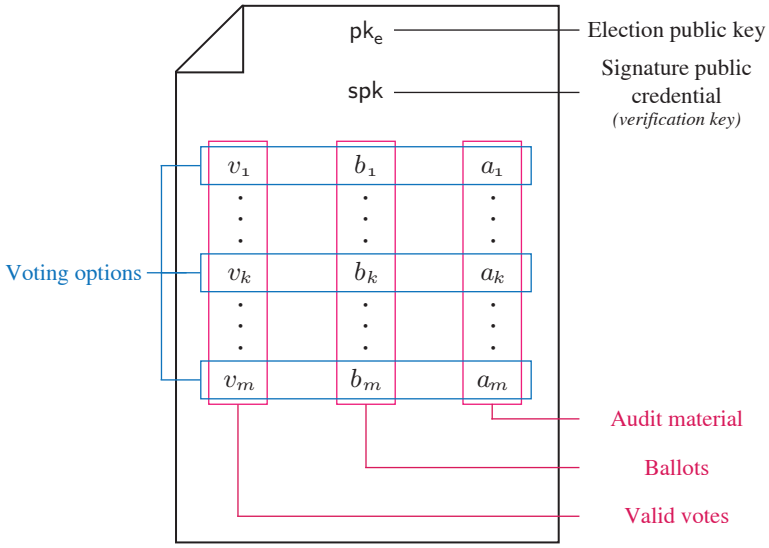


Figure 2.5: A voting sheet

the election administrator, the registrar and the voting server. The election private key could be shared among several trusted entities as in a threshold cryptosystem. To ease the reader's understanding, we assume in this section that the tallying authority is the only entity managing and storing the private key allowing the decryption of ballots.

At the end of the election, the tallying authority computes the result of the election as well as a publicly verifiable proof of validity of this result.

Election Administrator (EA)

The *election administrator* produces the list of eligible voters (VL) and the valid voting options list ($\mathcal{V}$) for the election. It then transmits them to the registrar and the voting server and ensures that they got the right public key (pk_e). It also makes sure that the public bulletin board, the voting options list and the number of eligible voters for this election are publicly available.

Registrar (R)

The *registrar* manages the voters credentials and material for the election. It generates a set of valid signing credentials ($\mathcal{C}$), one for each voter. Each credential is a pair of signature/verification keys (ssk and spk) and is used by the registrar to generate one **Voting Sheet (VSh)** for each voter.

The voting sheet provides the voter with every information needed to cast a ballot for the election. It displays the voter's signature public key, and for each valid voting option $v_k \in \mathcal{V}$ it displays three kind of information: the voting option (v_k), a precomputed ballot - the encrypted vote with its signature - and the audit material for this ballot so the user can check it does indeed encrypt their vote (see Fig.2.5). Those information could be displayed as flashable QR-codes for instance. It is important to note that a voting sheet is not bound to a specific voter.

The registrar ensures the list of valid public credential ($\mathcal{C}_{\text{pub}}$) is available to everyone, particularly to the voting server and elections auditors.

Voting Server (VS)

The *voting server* is in charge of receiving and publishing ballots on the **Public Bulletin Board**. To this purpose, it generates **authentication credentials** - typically a pair of login/password - for each voter on VL and sends it to the voters. These authentication credentials will be used by each voter during the voting phase to identify themself as an eligible voter.

Every time a ballot is cast to the voting server, it will process to the expected verification: is it cast by an eligible voter and is the ballot valid? If so, it will display the ballot on the public bulletin board after randomizing it. Every voter can verify their ballot appears by finding their signing public key on the bulletin board and anyone can audit the public bulletin board.

Voter (V)

Voters each hold credentials provided by the Voting Server to authenticate themself when casting their ballot during the voting phase as well as a voting sheet.

They can audit the voting sheet thanks to the audit material it displays or they can delegate this verification to a trusted person, not auditing a voting sheet will not prevent the voter from participating to the election.

They retrieve the well-formed ballot matching their choice of vote $v \in \mathcal{V}$ from the voting sheet and cast it through their voting device. This could technically be done by QR-code flashing for instance.

Finally, they can verify their vote appears on the bulletin board. This final step is also optional and can also be delegated to a trusted person.

Voting Device (VD)

The *voting device* is used by the voter for authentication to the voting server and to cast ballots retrieved from the voting sheet.

Auditing Device (AD)

The *auditing device*, checks if ballots displayed on the voting sheet truly encrypt all votes from $\mathcal{V}$. The voting sheet audit can be delegated to another person of trust for the voter. Note that this person - not the voting device - will then learn how the voter votes.

Election Auditor (A)

The *election auditor* is in charge of auditing both the public bulletin board and the election result (checking it is indeed consistent with the result produced by the tallying authority).

2.2.2 The cryptography behind our protocol

Our protocol is a variant of Belenios Receipt-Free [29][2] which is itself built on Belenios [38][3]. It relies on the cryptographic primitives defined in [23] based on *randomizable ciphertexts* [24].

Those primitives form an asymmetric encryption scheme and a signature scheme which are both randomizable by anyone. This means that given a ciphertext and its signature, anyone, without knowing the private decryption key, can generate a fresh ciphertext of the same plaintext - without knowing what

[2] See section 2.1.2.
[3] See section 2.1.3.

this plaintext is - *and* adapt the signature to the new ciphertext by producing a valid signature to it - without knowing the private signature key.

This section gives some insights about the capabilities of such cryptographic primitives.

Randomizable asymmetric encryption (pk, $\mathsf{aenc_p}$ and $\mathsf{adec_p}$)

During the setup phase, given a message space ($\mathcal{M}$), a pair of asymmetric keys is generated: a private key (k_a) and its associated public key ($\mathsf{pk}(k_a)$). The public key will be used to encrypt messages whereas the private key will decrypt ciphertexts.

The function $\mathsf{aenc_p}$ is used to encrypt a message m part of the message space $\mathcal{M}$. It takes four input parameters: the message, m, an asymmetric public key, pk, a signing public key, spk and a randomly generated nonce, r. It outputs a ciphertext c as well as a zero-knowledge proof that the plaintext of the ciphertext c is actually part of the message space (a proof that $m \in \mathcal{M}$) and that the message was encrypted with the verification key spk. The encryption is *randomized*, which implies that:

$$r_1 \neq r_2 \implies \mathsf{aenc_p}(m, \mathsf{pk}, \mathsf{spk}, r_1) \neq \mathsf{aenc_p}(m, \mathsf{pk}, \mathsf{spk}, r_2)$$

This way, the encryption of a message m does not necessarily produce the same ciphertext c.

The decryption of a ciphertext c is done with the function $\mathsf{adec_p}$ which takes the ciphertext c and the private key associated to the public key k_a that was used for the encryption. If the zero-knowledge proof is valid, it decrypts c with respect to the following equation:

$$\forall\, m, k_a, r, \mathsf{spk}.\ \mathsf{adec_p}(\mathsf{aenc_p}(m, \mathsf{pk}(k_a), \mathsf{spk}, r), k_a) = m \tag{2.1}$$

This encryption scheme will be used to encrypt and decrypt votes since it ensures several useful properties:

- It encrypts votes $v \in \mathcal{V}$ and ensures they are valid thanks to the zero-knowledge proof that v is an element of the message space $\mathcal{V}$.
- Thanks to the randomization done with the nonce - the encryption result of a vote v will not be unique. This property could help strengthen the vote confidentiality by using the randomization at several stages during our voting scheme.
- It links an encrypted vote v to a signing verification key spk.
- The decryption of a vote does not require any additional information apart from a ciphertext and the private decryption key - so there is no need to keep track of what was used to randomize the ciphertexts.

This asymmetric encryption scheme could easily be adapted to be a *threshold cryptosystem*: the private key would then be shared among several parties. Let n be the number of parties and t the threshold number. A ciphertext encrypted with the public key can be decrypted only if at least t of the n parties share their private key information, while less that $t-1$ parties have no useful information to decrypt anything.

Randomizable signature (spk, $\mathsf{sign_p}$ and $\mathsf{verify_p}$)

During the setup phase, a pair of signature keys is generated: a private signing key k_s and its associated public verification key $\mathsf{spk}(k_s)$. The signing key will be used to sign messages whereas the verification key will check the validity of a message signature.

The function $\mathsf{sign_p}$ signs a message m_p comprising a ciphertext and a zero-knowledge proof. It also takes four input parameters: the message, m_p, an asymmetric public key, pk, the signing private key, k_s, and a randomly generated nonce, r. If and only if the zero-knowledge proof is valid, $\mathsf{sign_p}$ outputs a signature of the message m_p.

Note that similarly to the function $\mathsf{aenc_p}$, the signature is *randomized* so we also have that the signature of the same message with the same signing key will not automatically produce the same signature since: $r_1 \neq r_2 \Rightarrow \mathsf{sign_p}(m_\mathsf{p}, \mathsf{pk}, \mathsf{spk}, r_1) \neq \mathsf{sign_p}(m_\mathsf{p}, \mathsf{pk}, \mathsf{spk}, r_2)$.

The verification of the signature s of a ciphertext m_p is performed by the function $\mathsf{verify_p}$ which takes the ciphertext, its signature, an asymmetric public key, pk, and the public verification key $\mathsf{spk}(k_s)$ as inputs. If and only if the zero-knowledge proof part of m_p is valid - that is, the plaintext encrypted by m_p is an element of $\mathcal{M}$ and the encryption was done with $\mathsf{spk}(k_s)$ as a parameter - the actual signature verification is computed. $\mathsf{verify_p}$ outputs true if the signature was indeed done with the private signing key k_s as described in the following equation:

$$\forall\, m, k, r, \mathsf{pk}.\ \mathsf{verify_p}(m_\mathsf{p}, \mathsf{sign_p}(m_\mathsf{p}, \mathsf{pk}, k_s, r), \mathsf{spk}(k_s)) = \mathsf{true} \tag{2.2}$$

We will use this signature scheme to sign ballots.

Randomization ($\mathsf{rand_p}$)

The encryption and signature schemes previously presented are *randomizable*. Which means that given a (randomized) encrypted message and its (randomized) signature, they can both be randomized by the function $\mathsf{rand_p}$ which will output a valid encrypted message along a valid signature.

The function rand takes six parameters as inputs: an asymmetric public key, pk - used for encryption - a signing public key, $\mathsf{spk}(k_s)$ - used for the signature - an encrypted message - ciphertext and zero-knowledge proof - its signature and two randomly generated nonces r_2 and s_2. It outputs a randomized encrypted message and its signature (also randomized) and consequently adapts the zero-knowledge proof according to the following equation:

$$\begin{aligned} & \mathsf{rand_p}(\mathsf{pk}, \mathsf{spk}(k_s), \mathsf{aenc_p}(v, \mathsf{pk}, \mathsf{spk}(k_s), r_1), \mathsf{sign_p}(\mathsf{aenc_p}(v, \mathsf{pk}, \mathsf{spk}(k_s), r_1), \mathsf{pk}, k_s, s_1), r_2, s_2) \\ = \ & (\mathsf{aenc_p}(v, \mathsf{pk}, \mathsf{spk}(k_s), r_1 + r_2), \mathsf{sign_p}(\mathsf{aenc_p}(v, \mathsf{pk}, \mathsf{spk}(k_s), r_1 + r_2), \mathsf{pk}, k_s, s_1 + s_2)) \end{aligned} \tag{2.3}$$

The randomization only needs public values - an encrypted message and its signature, public keys, nonces - so it can be processed by anything without needing private data. Thus, we will use this randomization at several stages in our voting protocol in order to ensure privacy.

The randomizable cryptography scheme applied to our voting protocol: computing ballots

Ballots will be computed with the randomizable cryptographic scheme defined in previous sections. Let $\mathsf{pk_e}$ be the asymmetric public key of the election and $(\mathsf{ssk}, \mathsf{spk})$ the signature keyset of a voter. Let also $\mathcal{V}$ be the set of valid votes.

We call a *valid encrypted vote* any output of the function $\mathsf{aenc_p}$ of the following form:

$$c_b := \mathsf{aenc_p}(v, \mathsf{pk_e}, \mathsf{spk}, r) \qquad (\text{with } v \in \mathcal{V},\ r:\ \text{nonce})$$

A *valid ballot signature* is an output of the function $\mathsf{sign_p}$ of the following form:

$$s_b := \mathsf{sign_p}(c, \mathsf{pk_e}, \mathsf{ssk}, s) \quad (\text{with } c:\ \text{valid encrypted vote},\ s:\ \text{nonce})$$

Finally, a *valid ballot* is a triplet of the form (spk, c_b, s_b) with c_b (resp. s_b) a valid encrypted vote (resp. valid ballot signature). A valid ballot is a *public* value that will be displayed on the public bulletin board.

The encryption of the vote as well as the randomization of the encryption and the signature induced by the use of $\mathsf{aenc_p}$ and $\mathsf{sign_p}$ will ensure the confidentiality of a vote. On another hand, the signature of a vote and the presence of the verification key spk on the ballot will provide a voter a way to easily verify their vote is on display on the public bulletin board as we will describe in the next section.

2.2.3 An overview of our protocol

Our protocol can be divided into three phases. First, organizers - tallying authority, election administrator, registrar and voting server - set the election up by providing each voter everything needed to cast a ballot. Then, once the voting ecosystem is set, voters process to the voting phase. Last, the public bulletin board is tallied and the election is audited.

Election setup

The first step of our protocol is summarized in Fig.2.6.

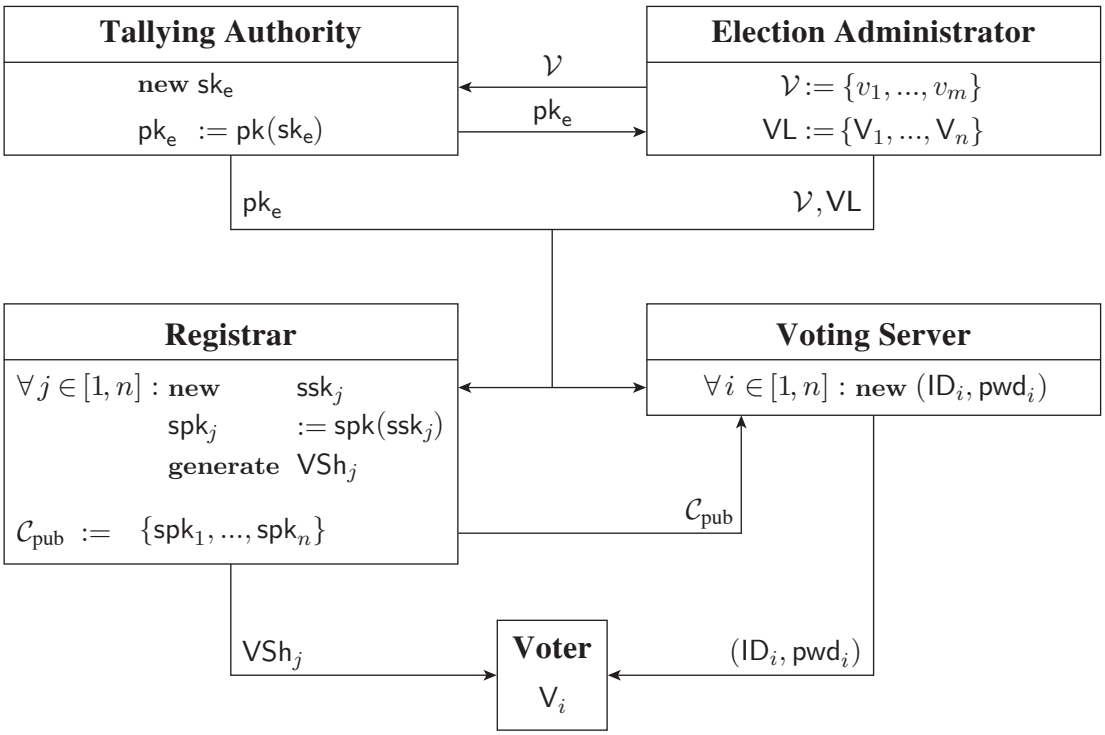


Figure 2.6: The election setup diagram

- **Election parameters:**

  First, the election administrator generates the valid voting options list ($\mathcal{V} = \{v_1, ..., v_m\}$) and the eligible voters list ($\mathsf{VL} = \{\mathsf{V}_1, ..., \mathsf{V}_n\}$). It sends $\mathcal{V}$ to the tallying authority which will use it as a

parameter to generate the election keys ($\mathsf{sk_e}$ and $\mathsf{pk_e}$). The election public key is then sent to the election administrator, the registrar and the voting server.

As stated before, we could use an asymmetric threshold cryptosystem and share the private key among several tallying authorities. This is a common security practice for voting schemes that we will not discuss in this thesis since it is out of our main scope. Nonetheless, for the sake of readability, we will consider the tallying authority to be a unique entity holding the election private key $\mathsf{sk_e}$ without any impact on the further security analysis of our protocol.

After receiving the election public key, the election administrator publishes the valid voting options list ($\mathcal{V}$) and the number of eligible voters (n). It sends $\mathcal{V}$ and VL to the registrar and the voting server.

- **Authentication credential generation:**

  For each voter $\mathsf{V}_{i,\ i\in[1,n]} \in \mathsf{VL}$, the voting server generates the voter's authentication credentials. In our scheme, we assume it to be a pair of login/password $(\mathsf{ID}_i, \mathsf{pwd}_i)$ although the reader shall keep in mind that other ways of authentication could be used.

  The authentication credentials are sent to each voter through a private channel - SMS, e-mail... Those credentials will be used by the voter to authenticate themself to the voting server during the voting phase.

- **Voting sheet generation:**

  The registrar generates as many credentials as the number of eligible voters (n): those are pairs of signing and verification keys $(\mathsf{ssk}_j, \mathsf{spk}_j)_{j\in[1,n]}$. The set of valid public credentials $\mathcal{C}_{\text{pub}} = \{\mathsf{spk}_1, ..., \mathsf{spk}_n\}$ is then publicly shared. It will be used as a reference by the voting server during the voting phase and by election auditors during the audit of the public bulletin board.

  For all $j \in [1, n]$, each pair of signature credentials is used to generate an associated voting sheet, VSh_j.

  As explained in section 2.2.1, with Fig.2.5, the voting sheet contains all information needed to process the vote, meaning that it displays ready-to-cast ballots for each valid vote options.

  First of all, it displays the election public key ($\mathsf{pk_e}$) and the voter's public signature key (spk_j).

  Given a pair of credentials $(\mathsf{ssk}_j, \mathsf{spk}_j)$, for all $k \in [1, m]$, the registrar generates two nonces ($r_{j,k}$ and $t_{j,k}$) and computes a valid encrypted vote ($c_{b_{j,k}}$) and its associated signature ($s_{b_{j,k}}$) as follows:

$$\begin{aligned} c_{b_{j,k}} &:= \mathsf{aenc_p}(v_k, \mathsf{pk_e}, \mathsf{spk}_j, r_{j,k}) \\ s_{b_{j,k}} &:= \mathsf{sign_p}(c_{b_{j,k}}, \mathsf{pk_e}, \mathsf{ssk}_j, t_{j,k}) \end{aligned}$$

  Those values are then used to generate the voting sheet, VSh_j.

  The voting sheet VSh_j displays m entries corresponding to each valid vote option from $\mathcal{V}$. For all $k \in [1, m]$, an entry features four information:

  - The vote option $v_k \in \mathcal{V}$.

  - The election public key with the signature verification key and the valid encrypted vote:

$$\mathsf{pk_e}, \mathsf{spk}_j, c_{b_{j,k}}$$

    It represents the first half of the information needed to cast a valid ballot.

- The signature associated to the encrypted vote, $s_{b_{j,k}}$. By flashing the previously mentioned values $(\mathsf{pk_e}, \mathsf{spk}_j, c_{b_{j,k}})$ with the associated signature, a voting device can compute the valid ballot:

$$b_{j,k} \quad := \quad (\mathsf{spk}_j, c_{b_{j,k}}, s_{b_{j,k}})$$

  Together, values $\mathsf{pk_e}$ and $b_{j,k}$ are called the *voting material*. It will be used by the voter during the voting phase to cast a ballot and does not contain any information about what v_k could be.

- The set of values comprising the vote option and the nonce used to generate the encrypted ballot $c_{b_{j,k}}$:

$$a_{j,k} \quad := \quad (v_k, r_{j,k})$$

  By flashing this set along the values $(\mathsf{pk_e}, \mathsf{spk}_j, c_{b_{j,k}})$, the voter has all information needed to audit the validity of the matching voting material. The voting sheet audit process will be detailed in the next section, but in essence, the nonce $r_{j,k}$ will be used by the auditing device to recompute the encrypted vote and see whether or not it matches with $c_{b_{j,k}}$.
  The values $(\mathsf{pk_e}, \mathsf{spk}_j, c_{b_{j,k}})$ along $a_{j,k}$ are called the *audit material* for the vote option v_k.

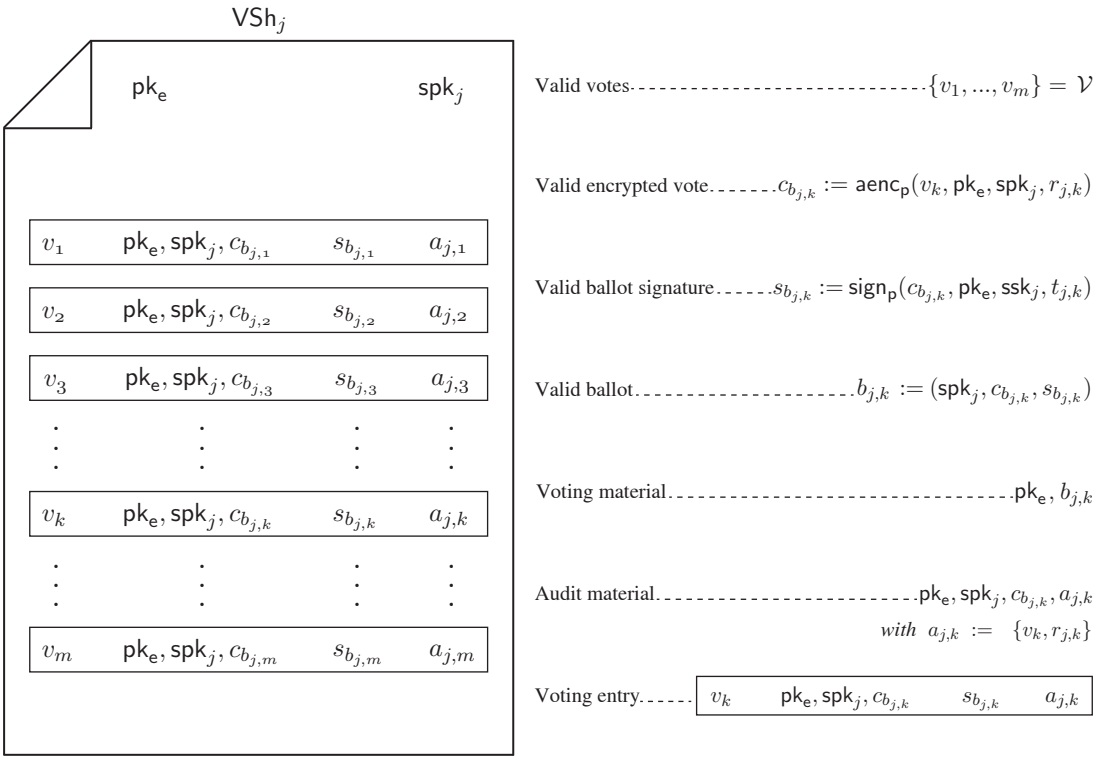


Figure 2.7: An election voting sheet

Fig.2.7 outlines what appears on the voting sheet. Each voting sheet is randomly distributed to each voter through a different channel than the one used by the voting server to send the authentication credentials. For instance, voters could claim their voting sheet directly at their polling station or receive them by mail, from a different shipper than the authentication credentials if those were also sent by mail. The important point being that a voter shall not be linked to a specific voting sheet.

Once the election setup is over, the voting process can actually begin.

Voting process:

The voting process can start once the voters have their voting sheet and their authentication credentials. It is recapped Fig.2.8.

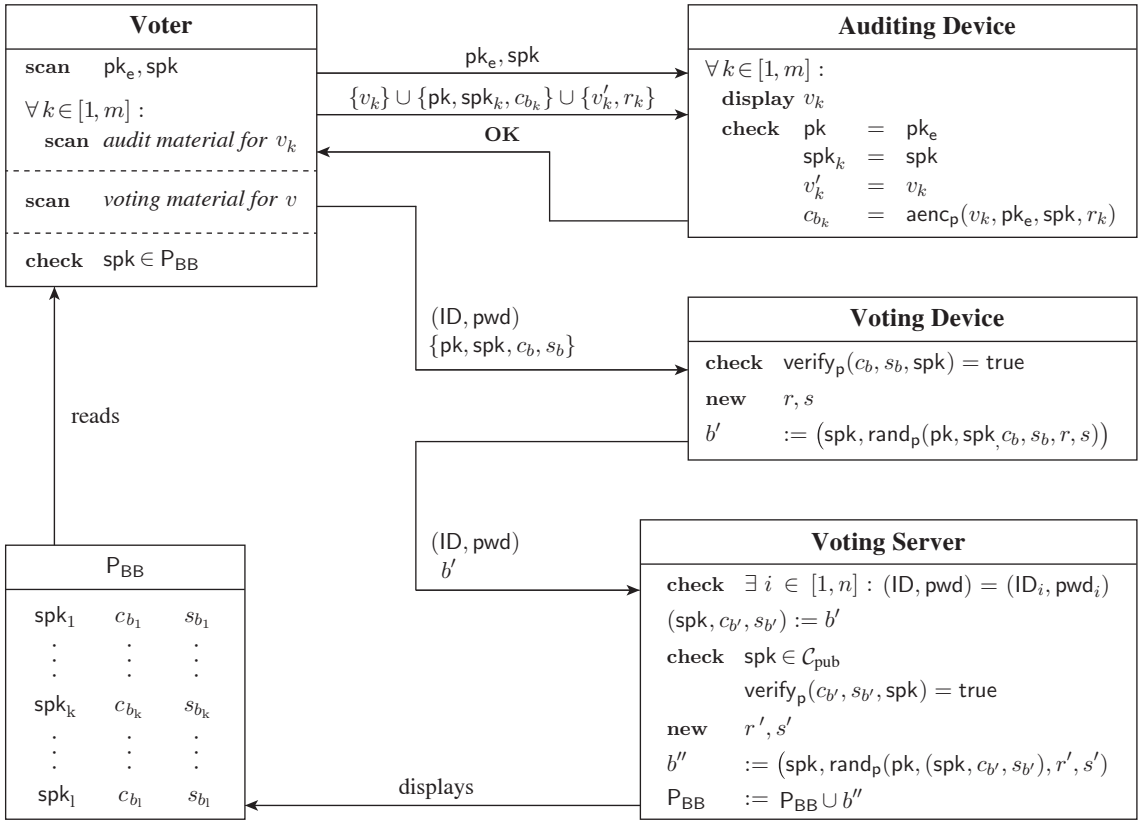


Figure 2.8: The voting process diagram

From the voter's point of view, the voting process can take up to three steps: auditing the voting sheet (optional), casting a ballot and verifying their vote has effectively been cast (optional). Four entities interact during this phase - the voter (V), the auditing device (AD), the voting device (VD) and the voting server (VS). We describe their exchanges in this section.

- **Voting sheet audit (*optional*):**

  The voter (V) can audit their voting sheet (VSh) by scanning the election public key ($\mathsf{pk_e}$) their signature verification key (spk) and *all* audit material along their matching vote option through their auditing device. All those information are displayed on the voting sheet as seen in Fig.2.7.

  1. The voter scans the election public key ($\mathsf{pk_e}$) and the signature verification key (spk) from the voting sheet (VSh). The auditing device will temporarily store them as a reference to audit the ballots.
  2. For all $k \in [1,m]$[4], the voter scans the vote option v_k and the matching audit material through the auditing device.

[4] m is the number of valid vote options from $\mathcal{V}$

3. For all $k \in [1, m]$, the auditing device processes the following verification:
   First, it parses the audit material and check it is of the right form:

$$(\mathsf{pk}, \mathsf{spk}_k, c_{b_k}, v'_k, r_k)$$

   It then displays v_k to the voter and checks the following equalities:

$$\begin{array}{lcl} \mathsf{pk} & = & \mathsf{pk_e} \\ \mathsf{spk}_k & = & \mathsf{spk} \\ v'_k & = & v_k \\ c_{b_k} & = & \mathsf{aenc_p}(v_k, \mathsf{pk_e}, \mathsf{spk}, r_k) \end{array}$$

   These equalities ensure the encrypted vote c_{b_k} is indeed valid regarding the election - it was encrypted with $\mathsf{pk_e}$ - and that it does encrypt the right vote v_k with the right verification key spk as a parameter. Note that this verification also includes the computation - and comparison - of the zero-knowledge proof part of c_{b_k} as stated in section 2.2.2.
4. If and only if all verification are valid, the auditing device validates the voting sheet to the voter.

Auditing the voting sheet is an optional step, not performing it does not prevent the user from casting a ballot. It could also be delegated to a trusted entity and/or person.

The main purpose of this audit is to allow the voter to check the voting sheet they are about to use to cast their ballot does actually encrypt their voting choice. For instance, an attacker could give the voter a voting sheet encrypting the same vote instead of all the options from $\mathcal{V}$ to ensure the voter votes for a given candidate. This could happen if for instance, the registrar was under the attacker control or if the voting sheet was intercepted during the dispatching process of the election setup.

Note that the auditing device does not proceed to the signature verification of the ballot - which will in fact be performed by the voting device. This prevents a rogue auditing device to get all information needed to cast a valid ballot on part of a honest voter.

The fact that the auditing device shall audit *the whole* voting sheet and not just an entry is actually important for privacy matters. If the voter only audited one entry and if the attacker controlled the auditing device, the attacker could actually assume this entry matches the voter's choice and guess their vote.

- **Casting a ballot:**

  To cast a ballot, the voter scans the voting material matching their choice through the voting device. They also have to provide their authentication credentials - in our case, the login/password pair sent by the voting server during the election setup (ID, pwd) - to the voting device to authenticate themself as an eligible voter to the voting server.

5. The voter scans the voting material b matching their voting choice through their voting device.
6. The voting device parses the message b and checks whether it is of the right form:

$$(\mathsf{pk}, \mathsf{spk}, c_b, s_b) := b$$

It then verifies the ballot signature is indeed valid:

$$\mathsf{verify_p}(c_b, s_b, \mathsf{spk}) = \mathsf{true}$$

Note that this verification includes a verification of the zero-knowledge proof part of c_b as stated in section 2.2.2. If the verification is successful, it generates two nonces r and s for *randomization*.

The new ballot is:

$$b' := \big(\mathsf{spk}, \mathsf{rand_p}(\mathsf{pk}, \mathsf{spk}, c_b, s_b, r, s)\big)$$

7. If the randomization is successful, the voter has to provide the authentication credentials $(\mathsf{ID}, \mathsf{pwd})$ to the voting device which will use them to connect to the voting server.
8. If the credentials sent to the voting server are valid - if there exists $i \in [1, n]$[5] such that $(\mathsf{ID}, \mathsf{pwd}) = (\mathsf{ID}_i, \mathsf{pwd}_i)$ - and if those credentials were not previously used to cast a ballot, the connection is approved and the ballot b is cast to the voting server.

As described in section 2.2.2, the randomization produces a valid ballot. It is used here for privacy matters: if the voting sheet was available to the attacker for instance, the ballot b matching the voting choice $v \in \mathcal{V}$ could not be linked to the randomized ballot b'.

- **Publication on the public bulletin board:**

Once a ballot b' is received by the voting server, it will process some verification before accepting it, randomize it and publishing it on the public bulletin board.

9. The voting server parses the ballot b' and checks it is of the right form:

$$(\mathsf{spk}, c_{b'}, s_{b'}) := b'$$

If so, it continues the ballot verification process.

10. The voting server then checks the signature verification key is an element of $\mathcal{C}_{\text{pub}}$, the valid public credential list set by the registrar during the election setup. It also checks this key was not previously used to cast a ballot.

If so, it processes to the signature verification of the ballot:

$$\mathsf{verify_p}(c_{b'}, s_{b'}, \mathsf{spk}) = \mathsf{true}$$

If this signature is valid, it generates two nonces r' and s' for randomization.

The new ballot is:

$$b'' := \big(\mathsf{spk}, \mathsf{rand_p}(\mathsf{pk}, (\mathsf{spk}, c_{b'}, s_{b'}), r', s')\big)$$

11. If the randomization is successful, the voting server adds b'' to the public bulletin board ($\mathsf{P_{BB}}$) which is the public list of *valid ballots*[6] cast by voters.
12. Once the publication is done, the voting server will end the session with the voting device and mark ID as a voter that has already voted and spk as an already used credential - meaning spk appears on $\mathsf{P_{BB}}$.

[5] n is the number of eligible voters in VL

[6] Triplets comprising a signature verification key, a valid encrypted vote and a valid ballot signature as stated in section 2.2.2.

Similarly to the previous step, the ballot randomization is done to ensure more privacy.

Note that our voting scheme does not accept revoting. This will be discussed later as a prerequisite for our protocol to be verifiable.

- **Bulletin board verification (*optional*):**

  After successfully casting a ballot, a voter can check it was taken into account by checking if their signature verification key (spk) appears on the public bulletin board ($\mathsf{P_{BB}}$)

  13. The user retrieves the public bulletin board $\mathsf{P_{BB}}$ which is publicly displayed by the voting server.
  14. If there is an entry on $\mathsf{P_{BB}}$ where spk appears, the user verified their vote, meaning that they are ensured their vote will be part of the final tally.

  Like the voting sheet audit, this step is optional and can be delegated to a trusted entity or person by providing the signing public key (spk) appearing on the voting sheet. This does not compromise the vote confidentiality since the third party does not have any information on the voter's choice nor can he deduce it.

  Our voting verification can seem quite simple to actually guarantee a voter their vote will be taken into account, however, we expect having some compelling arguments. Each voter is supposedly given a unique voting credential spk which is part of a public list $\mathcal{C}_{\text{pub}}$. If an entry on the bulletin board contains spk, the voter can expect it matches their ballot. Moreover, thanks to the public display of the bulletin board and the possibility of auditing it with all the information it holds - more detail on that on the next section - the voter can be reassured that their ballot displayed on the bulletin board is a valid one that was indeed signed with spk. So the ballot appearing on the bulletin board shall be encrypting the voter's choice.

Once the voting phase is over the tallying process can begin.

Public bulletin board audit and tallying process:

The tallying process can start as soon as the voting phase over. However, to make sure the votes which are about to be tallied are valid ones, there should be a public bulletin board audit process which we will also detail in this section.

- **Auditing the public bulletin board:**

  The public bulletin board audit detailed in Fig.2.9 can be performed by anyone, even people outside of the election organization and at anytime during the voting and tally phase.

  Let's assume an election auditor who only has access to the public election parameters - the election public key ($\mathsf{pk_e}$), the number of eligible voters (n), the valid voting options ($\mathcal{V}$) and the valid public credentials list ($\mathcal{C}_{\text{pub}}$) - and the public bulletin board ($\mathsf{P_{BB}}$). Let's also assume that at the moment the election auditor is retrieving the public bulletin board, it displays l entries $(b_i)_{i\in[1,l]}$.

  1. The election auditor first checks that the number of entries is less than the number of eligible voters: $l \leqslant n$. If not, the bulletin board is not valid.
     If so, the election auditor can parse all ballots displayed on the bulletin board:

     $$(\mathsf{spk}_i, c_{b_i}, s_{b_i}) := b_i$$

     Then, for all $i \in [1, l]$, the election auditor will process to the following verification.

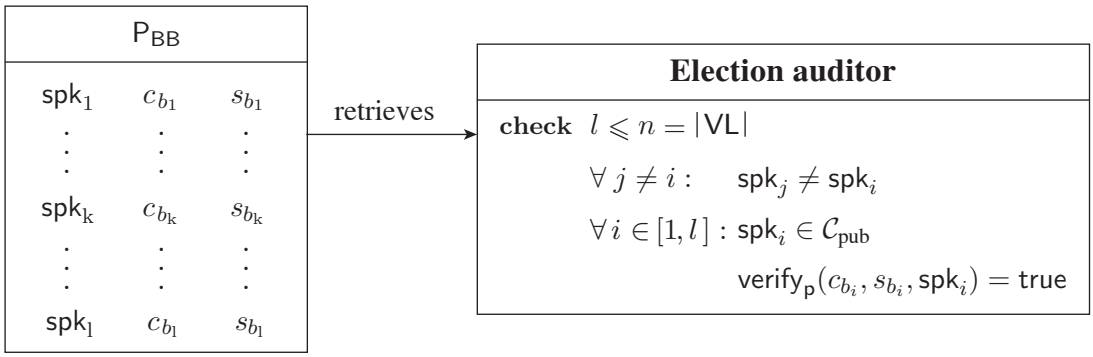


Figure 2.9: The public bulletin board audit

2. The signature verification key is displayed in only one entry.

$$\forall\, j \neq i.\ \mathsf{spk}_j \neq \mathsf{spk}_i$$

3. The signature verification key was generated by the registrar.

$$\forall\, i \in [1, l].\ \mathsf{spk}_i \in \mathcal{C}_{\text{pub}}$$

4. The ballot is valid as well as its signature.

$$\forall\, i \in [1, l].\ \mathsf{verify_p}(c_{b_i}, s_{b_i}, \mathsf{spk}_i) = \mathsf{true}$$

If all verification are successful, the bulletin board is considered as *valid*.

- **Election tally:**

  The tallying process is actually the same one detailed in [29].

  First, the tallying authority proceeds to the public bulletin board audit as previously described.

  If the audit is successful, all ballots' ciphertexts ($c_{b_i},\ i \in [1, l]$) are sent over a mix-network to be shuffled before decryption. After this operation, each ballot is individually decrypted with the election secret key $\mathsf{sk_e}$. The tallying authority outputs the tally result of the election (R) together with a proof of correct tabulation (Π).

  Note that in practice, for instance as in Helios or Belenios protocols, the actual asymmetric cryptosystem is a threshold one, as previously stated in section 2.2.1. Typically, more than three authorities share the decryption key and at least two of them have to be involved to actually compute the tally. This ensures a greatest robustness of the voting scheme which will not be described here, we refer the reader to [37] for more information on the topic.

- **Auditing the election tally:**

  The election result can be audited. It takes the public bulletin board that was tallied ($\mathsf{P_{BB}}$), the result (R) and the proof (Π) as inputs.

  First, the bulletin board is audited with the audit process previously described.

  Then, the election auditor can check whether Π is a valid proof of correct tallying for the result R.

If all verification are successful, the election result is declared as valid.

2.3 Threat model

During the electronic voting process, several security breaches could happen. The voter's devices could be infected by a malware, the voting server or the registrar could be the target of an attack... Since eight entities intervene in our scheme and communicate over several different channels, we need to consider each plausible corruption scenario for the security analysis of the protocol.

The purpose of this section is to discuss our threat model and security assumptions we made for our security proofs.

- Which entities are corruptible, which are not?
- What information could be leaked?
- Which networks are under the - complete or partial - control of the attacker?

We will end this sections by summarizing which are the corruption scenarii we consider for our security analysis.

2.3.1 Threats

We describe here which are the possible threats to our voting ecosystem. We distinguish two different kinds of threats: the attacker controls a specific entity or some errors have been done by one of them.

Honest entities

Because of their role in our scheme, we assume three of the acting entities of this voting scheme are always honest.

Since the **election administrator** only has an administrative role in our voting scheme - it ensures the registrar and voting server do have the right election public key and produces the list of valid votes as well as the one of eligible voters - we assume it is a honest entity. The list of valid votes is public thus pointless to corrupt whereas producing a fake eligible voters list could be assimilated to producing voting credentials for unregistered voters, something that could be done by the registrar and the voting server. Moreover, even if we do not require it to be a public value - although it does not provide any security breach in our scheme if it were - the eligible voters list can actually be publicly available.

The asymmetric encryption scheme our protocol is based on is actually a threshold cryptosystem. This is a security measure required by most voting schemes. Given a security parameter k and a number of entities n, it means that strictly less than k entities cannot decrypt the ballots. There has to be at least k out of n independent entities involved in the process. This is abstracted in our models, the **tallying authority** is considered as a unique entity. Thus we also consider it to be always honest, for the corruption of the majority of tallying authorities would be unlikely and that the result output by the tallying authority is compliant with the bulletin board content.

The election audit might be performed by anyone outside of the election organizers. Indeed, it only requires access to public values: the public bulletin board that was tallied, the public credentials list, the number of eligible voters and the election result along its proof of correct computation. This means that the election result could be audited by several independent entities and that we could hope for one of them to be honest. Thus we assume the **election auditor** to also be a honest entity and that the election audit is always correctly performed.

Corruptible entities

Regarding **the other entities** - registrar, voting server, voters, auditing device and voting device - they **can either be honest or rogue**. A rogue entity is entirely under the attacker control. For instance, a rogue voter could provide their credentials to the attacker or a rogue voting server could push rogue ballots on P_{BB}. More specifically, a rogue entity will give all its secret values to the attacker.

Corrupted voters are allowed to participate to the election. They are under the control of the attacker thus giving it all credentials they hold.

Possible human errors

Without considering an entity to be entirely under the attacker control, we can consider three more corruption scenarii that could be related to human errors.

Both the **authentication credentials** and the **voting sheet** could be available to the attacker without an entity being completely under its control. The voting sheet could be lost or seen by the attacker. Same goes for the authentication credentials: the attacker could get them directly from the voter - by a phishing attack for example. We considered the loss of the voting sheet or voting credentials as two additional corruption cases.

Moreover, since the **voting sheet audit** is an optional step, a voter could skip it. This also another corruption case.

Output of the election secret key

If the **election secret key** is given to the attacker, it can decrypt all ballots. In this case, vote confidentiality could not be guaranteed which makes it a useless case to analyze regarding privacy. However, we still consider this corruption case in our study of verifiability to check the resistance of our protocol to it.

2.3.2 Communication model

Our protocol relies on the use of thirteen different channels summarized in Table 2.1. We can distinguish two kinds of channels: channels between entities and channels to retrieve public values.

Regarding public values - the valid public credentials list ($\mathcal{C}_{\text{pub}}$) and the number of eligible voters (n), the public bulletin board P_{BB} and the election result - we assume they are *accessible to everyone* (they could be accessed from the Internet). However, since it would be technically too difficult to corrupt *each and every* device used to consult them, or a DNS, we also presume they are *the same for everyone*. That is, if anyone - a voter or an election auditor for instance - retrieves those values, they will be the actual ones output by the entity responsible for them.

Since we consider the election administrator and the tallying authority to be honest, The channel between them is always considered as *secure*.

We consider the channel between the voting device and the voting server to be *authenticated*. This implies that an attacker can eavesdrop, intercept and/or drop messages going over this channel, however it cannot affect their integrity.

As for all other channels, they are considered as secure. However, since those channels are between corruptible entities - registrar, voting server, voter, voting device, auditing device - we consider them to be under the attacker's control whenever an entity at one end is corrupted.

Entities or public values	Channel type	Corruptible
$\mathcal{C}_{\text{pub}}$ and n - All	Public	No
P_{BB} - All	Public	No
Election result - All	Public	No
Tallying Authority - Election Administrator	Secure	No
Tallying Authority - Registrar	Secure	Yes
Tallying Authority - Voting Server	Secure	Yes
Election Administrator - Registrar	Secure	Yes
Election Administrator - Voting Server	Secure	Yes
Registrar - Voter	Secure	Yes
Voting Server - Voter	Secure	Yes
Voter - (Voting Sheet) - Auditing Device	Secure	Yes
Voter - (Voting Sheet) - Voting Device	Secure	Yes
Voting Device - Voting Server	Authenticated	Yes

Table 2.1: Channels of our voting scheme

2.3.3 Corruption scenarii

We summarize our communication and threat model considered for our security analysis in Fig.2.10. So in total, we have nine corruption cases that can be combined all together:

- Five entities are corruptible:
  - the registrar (R),
  - the voting server (VS),
  - the voter (V_k),
  - the voting device (VD),
  - the auditing device (AD).
- Two objects can be leaked to the attacker:
  - the authentication credentials ($\mathsf{ID}_k, \mathsf{pwd}_k$),
  - the voting sheet (VSh_i).
- The voter can forget to audit their voting sheet.
- The election private key can be output.

We analyze the security of our protocol with regard to verifiability and privacy for all possible combinations of those corruption cases.

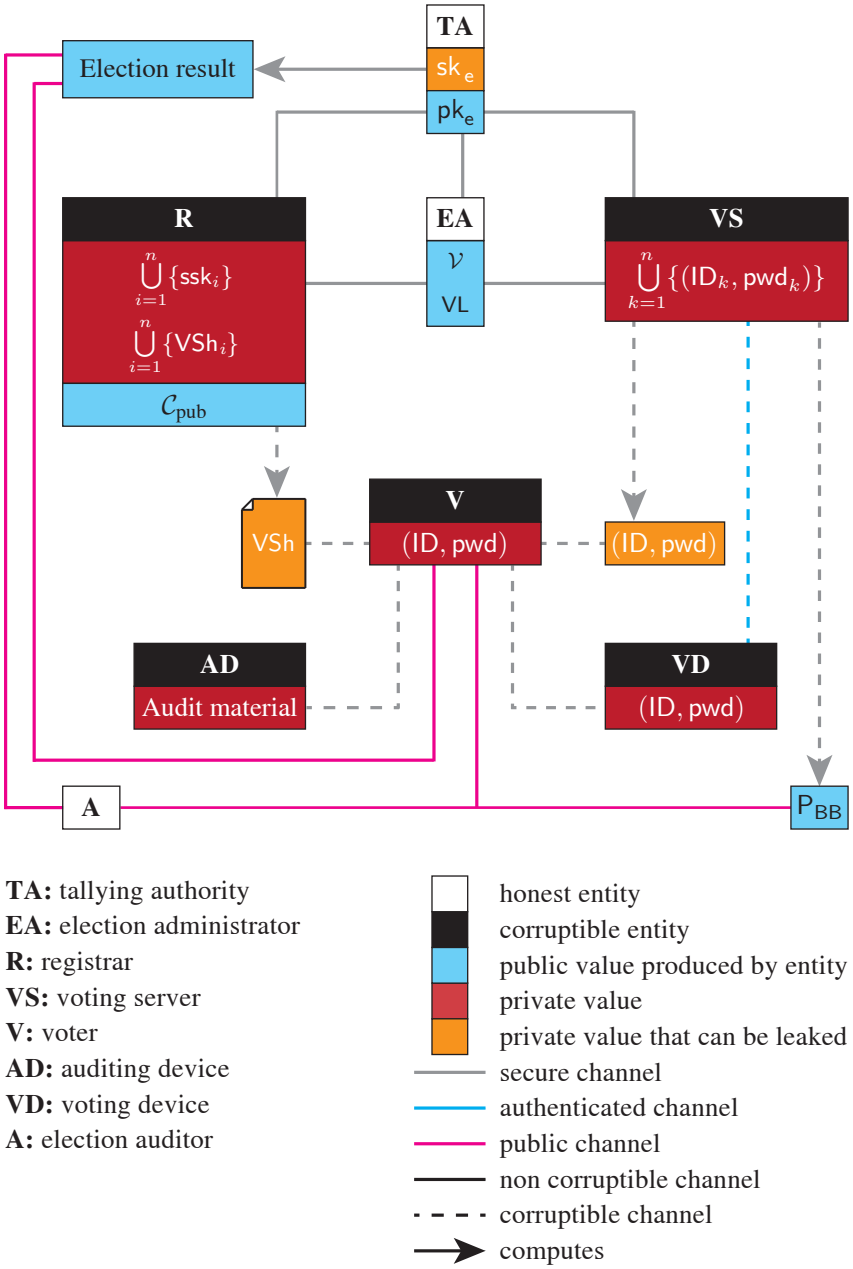


Figure 2.10: Communication and threat model of our voting scheme

2.4 Security claims

We analyze our protocol security regarding two properties: verifiability and privacy. However, since there are several cases of corruption, we need to explicitely state for which corruption scenarii our protocol is secure.

2.4.1 Verifiability

We rely on the definition of strong verifiability proposed in [38]. In essence, we can say that a voting protocol is verifiable if it guarantees both individual and universal verifiability.

Intuitively, we say that a voting scheme is verifiable if the election result matches the votes from:

- All honest voters who verified that their vote was correctly cast. We call this the *Tallied as Cast* property.

- A subset of the votes from honest voters who did not verify their vote was correctly cast, which is plausible in practice. This means that an attacker can at most erase the vote of a honest voter (and not replace it for instance).

- If we have a subset of n corrupted voters, at most n other ballots will be tallied as valid votes. This means that an attacker may only use corrupted voters to cast valid votes.

A remark on our revote restriction policy

As stated during the description of our protocol, we prohibit revote in our protocol. So honest voters will only cast their vote once.

The main reason behind this is the fact that we would loose verifiability if revoting was authorized. Indeed, because of the randomization happening at several steps of the voting process, there is no way to make a connection between the vote displayed on the bulletin board and the voter's original ballot.

Figure 2.11: Attack on verifiability when authorizing revote in our scheme.

Let's assume revoting is authorized. Then, verifiability is compromised as illustrated by the attack described in Fig.2.11 when the same voter casts two ballots for two different candidates. An attacker controlling the voter's device or the voting server could intercept and drop the second ballot, randomize the first one, and cast it - or register it - as the voter's choice. The voter could still verify their ballot was cast by checking the presence of the signature verification key on the bulletin board but it would not be a guarantee that the plaintext of the ballot would actually be their last choice.

Because of that, and because we consider the server as a corruptible entity (which would make this attack even easier), we chose to forbid revoting.

2.4.2 Privacy

Privacy basically means that an attacker shall not learn any information about how the voter voted. We discuss on how to formalize that in further section 4.2.

A remark on our protocol's receipt-freeness and resistance to coercion

Because ballots are randomized at several steps during our protocol - by the voting device before casting the ballot and by the voting server before publishing it - once a voter voted for a candidate, they cannot prove for whom they voted. In this sense, our protocol inherits the receipt-freeness from Belenios RF.

Nonetheless, like Belenios RF, Belenios VS is *not* coercion resistant, for there is nothing preventing a voter from selling their voting credentials and voting sheet to someone else.

2.4.3 Our protocol security against several corruption scenarii

Considering the informal definitions of verifiability and privacy we stated, we make the following security claims:

Belenios VS is *verifiable* if either one of these conditions is satisfied:

- The registrar is honest *and* the voting sheet has not been leaked.
- The voting server *and* the voting device are honest *and* the authentication credentials have not been leaked *and* either the registrar is honest *or* the voter audits their voting sheet with a honest auditing device.

In all cases, whether the election secret key is leaked or not has *no* impact on verifiability.

Belenios VS is *private* if either one of these conditions is satisfied:

- The election secret key is not leaked *and* the registrar is honest *and* the voting sheet is not leaked *and* the voter does not audit the voting sheet.
- The election secret key is not leaked *and* the registrar is honest *and* the voting sheet is not leaked *and* either the voter audits their voting sheet with a honest auditing device *or* the voting device is honest.
- The election secret key is not leaked *and* the authentication credentials are not leaked *and* the voting server *and* the voting device are honest *and* either the the voter audits their voting sheet with a honest auditing device *or* the registrar is honest.

Conclusion and discussion

We proposed a variant of Belenios RF that guarantees vote confidentiality and vote verifiability even against a rogue user's device under some security assumptions.

The use of a voting sheet is the main difference with Belenios RF: the voter does not compute their ballot directly from their device - which could result in a leak of their voting choice if the device is corrupted. Instead, ballots are pre-computed and displayed on a voting sheet. This way, if the device used to cast a ballot is corrupted, an attacker could still not guess what the user actually voted. During the voting phase, from the user's point of view, easy tasks are required from them to cast a ballot. They can audit their voting sheet by flashing it, they also cast their ballot by flashing the voting sheet and finally they simply need to find an entry (their signature verification key) to verify their vote was correctly cast. To add more simplicity, the user could delegate the voting sheet audit and the vote verification to another person of trust without a loss on security.

In order to be precise with the security claims of our protocol and because of the many entities involved in it and the size of its specification, we needed to define carefully our trust assumptions and attacker model for our security proofs. We considered classical corrupted voters as a possibility, however we stated that all entities - except the tallying authority, the election administrator and the election auditor - were corruptible. Although the election administrator's honesty is quite understandable since it only has an administrative role, there is still some legit discussion regarding the honesty of the two other entities.

The tallying authority is considered as a honest entity as a result of an approximation: this voting protocol relies on a threshold cryptographic scheme. More particularly, the election decryption key used for tallying is shared among n tallying authorities - typically three - and a threshold number k - typically two - is defined. The tally cannot be processed unless at least k among the n entities process it. $k - 1$

entities do not possess the information to decrypt ballots. Setting up and choosing the total number of tallying authorities n and the threshold number k is defined after assessing the risk that at most $k-1$ tallying authorities among n could be corrupted. We considered this risk assessment to be correctly done thus implying the tallying authority to always be honest however we do not define what n and k should be.

We also assumed that the election auditor is always honest. Our main argument was that this task could be performed by anyone of independent interest because it only requires public values to be correctly done. If the election is big enough, we could indeed hope the interest around it to be quite ample and for at least one election auditor to be honest while processing this audit, given the challenge it represents. However, in some countries, like France, electronic voting is disregarded as a valid method for high profile elections - this criticism is indeed legit but we will not discuss it in this thesis. Thus, the election relying on electronic voting raise usually less interest than a national one involving every citizen of a whole country - for instance, the election of union staff representatives would probably only concern the employees of a specific firm or branch. In practice, the election audit could then be performed by only one auditor. In this case, an error or a corruption of the auditor is not unlikely so we could ask ourselves if regarding the social context of an election, the impact on security could not be higher than we expect it to be.

We did not discuss the necessity of separating the audit and voting device. Regarding verifiability, in some cases, the fact that both devices are corrupted does not impede the protocol's verifiability. Yet, the importance of separating both devices appears when privacy is at stake. If both devices are corrupted, the attacker has all information needed to learn the voter's vote. To diminish the risk, the voter shall use two separate devices to audit the voting sheet and cast a ballot.

The role of the registrar appears as really strong in this scheme, whereas the predominance of the voting server is more important in other voting schemes. Indeed, the registrar is always required to be honest, except in the case where all other entities are honest *and* the voters all audit their voting sheet. This last scenario seems highly tedious to be guaranteed in practice. However, we could argue that it is still easier to protect one registrar and the voting sheet distribution than to make sure every device of every voter is not infected.

Finally and quite surprisingly for us, we found out that our protocol's privacy was not independent from its verifiability. In fact, the security analysis we performed raised some attacks we did not foresee whenever the verifiability was lost, we present and discuss such attacks in further chapter 4. Analyzing those attacks on verifiability, we figured out that the privacy of our protocol in a specific corruption scenario implied the verifiability of our protocol in the same corruption scenario. Additionally, the attack we found on privacy whenever the verifiability could not be guaranteed could be adapted to other voting schemes for it substantially the same in every case. The next chapter will provide some insights about the possible implications of such attacks on electronic voting protocols.

In our next chapter, we will present the work done to formally prove our voting protocol security under several corruption scenarii. Two main obstacles had to be overcome: first, the verifiability is not a security property that an be expressed with nowadays automatic verification tools. We will discuss why and how to adapt our model to still automatically guarantee the verifiability of a scheme. Second, the high number of corruption scenarii required us to optimize the generation of our models. The whole plausible corruption cases represent more that 70 corruption combinations to be considered. We will also discuss this point in the next chapter.

Chapter 3

Achieving verifiability with ProVerif provable properties

The protocol we presented in the previous chapter was the subject of an extensive security study regarding its verifiability and privacy against all plausible corruption scenarii it could fall under.

But before proceeding directly to analyze whether or not our protocol is verifiable and against which combination of corruptions cases, we need to make a little *detour*, for the informal definition for verifiability we gave in section 2.4.1 cannot be expressed as a security property in automatic verification tools.

In fact, this problem has been raised in several previous works [42, 39]. [42] provides a formal proof of Helios' verifiability using the F* tool. It proposes a type-based set of security properties - *individual* and *universal* verifiability - and prove that they imply *end-to-end* verifiability if satisfied by a protocol under some security assumptions. End-to-end verifiability though is a weaker definition than the one we use since it does not require any control over the number of rogue ballot as [38] does. On the other hand, [39] proposes an analysis of the Neuchâtel electronic voting protocol which is not *academically* verifiable, for the content of the bulletin board is not publicly available. Instead, the Neuchâtel protocol

claims - and was proven - to satisfy *Cast-as-Intended* and *Recorded-as-Cast* properties assuming the voting server is honest Intuitively, those properties are defined as follows:

- *Cast-as-Intended (Neuchâtel protocol)*: if the voting server registers a ballot for a specific voter, then the ballot contains the vote intended by the voter.
- *Recorded-as-Cast (Neuchâtel protocol)*: if a voter completes the voting process, then they are assured their ballot has been registered by the voting server.

Although those properties are not enough to imply verifiability if a voting server is compromised, it appeared that we could build from them and the ones from [42] to provide a set of properties that imply verifiability as defined by [38].

This is the scope of this chapter, we begin by formalizing verifiability and explain why we cannot verify it with automated verification tools, we then propose two theorems along their proof: each one of them proposes a set of trace properties expressible in the ProVerif calculus that, when satisfied,

3.1 Formalizing verifiability

We remind here the reader the definition of verifiability stated in [38]: a voting protocol is verifiable if the election result matches the votes from:

- All honest voters who verified that their vote was correctly cast.
- A subset of the votes from honest voters who did not verify their vote was correctly cast.
- If we have a subset of n corrupted voters, at most n other ballots will be tallied as valid votes.

The first goal of this section is to provide a protocol-independent formal definition of verifiability. Unfortunately, this definition of verifiability is intrinsically linked to the notion of *counting*, which is a big issue regarding the fact that we would like to be able to prove the security of voting protocols with the help of existing automatic verification tools. ProVerif does not handle counting. As for Tamarin, although it can handle some counting security properties, it does not handle the equational theory that models the randomizable cryptographic scheme used in our protocol.

If a voting scheme relies on a registrar - which provides the voting credentials to cast a ballot - and a voting server - which provides credentials for the voters authentication during the voting process, it seems that we cannot expect the protocol to be verifiable if both entities are corrupted, for the attacker could then easily stuff the ballot box. Yet, if at least one of those entities is honest, we can hope for a voting protocol to still be verifiable.

Extrapolating the security claims from the Neuchâtel protocol, it appears that we have the premises of some *ground properties* that can be expressed as trace properties and that, if satisfied under some hypothesis we need to define, imply verifiability:

- *Cast-as-Intended:* if a ballot registered from a honest voter is about to be tallied, then this ballot was cast by a honest voter and contains its intended vote.
- *Tallied-as-Cast:* if a voter verified their vote - with the verification process defined by the voting scheme - then there is a ballot in the tally that contains their voting choice.

We voluntarily separate those informal definition from the voting server's action, unlike [39], for we consider corruption cases where the voting server could be corrupted. Yet, we can note that the notion of "ballot registered from a honest voter" slightly differs regarding the corruption status of a registrar or a

voting server. If the voting server is considered as a honest entity, we can hope the authentication of voters to the voting server is correctly performed. Thus a ballot registered for a honest voter would be a ballot registered using the authentication credentials of a honest voter. But if we have no guarantee about the security of the voting server, then we cannot trust this authentication process to be correctly performed. However, in that case, and if the voting scheme we consider relies on such an entity, we consider the registrar is honest and that it provided the voter some information - typically voting credentials - to cast ballots. So a ballot registered for a honest voter would be a ballot cast with the voting credentials of a honest voter.

This intuition lead us to define two contexts: one where the authentication credentials were not corrupted, the other where, if they exist in the voting scheme, the voting credentials are not compromised. We had to clarify what those contexts where by formalizing them as hypothesis. Then, regarding those contexts, we could define several *trace properties*[7] that could easily be expressed in ProVerif and which, if satisfied, imply the verifiability of a voting scheme.

Voting schemes on our scope here answer to some requirements:

- The ballot box must be displayed on a *public bulletin board* available to everyone.
- The *election audit* is presumably correctly - and honestly - performed. We discussed this assumption in our previous chapter and refer the reader to section 2.3.1 where the discussion is located.
- We assume a *voting server* is in charge of managing and sending individual *authentication credentials* to - eligible, if this entity is honest - voters. Such credentials are used during the voting process by voters to identify themselves to the voting server.
- *If* the voting scheme relies on an additional entity that act as a *registrar*, we assume this entity manages and send each voters individual *voting credentials* that allow the computation of well-formed ballots. In our case for instance, those are the election public key (for voting encryption) and the user's signing key (for ballot signature). Note that we do require them to be all different from one another for each voter. Yet, we do not require the registrar and the voting server have to be separated entities, at least for verifiability. Also note that the registrar's existence is not mandatory if the authentication process is correctly performed.

We provide two sets of trace properties that can be expressed with ProVerif. The first set of properties should reasonably be satisfied by a voting protocol assuming that the authentication process during the voting phase is correctly performed. This includes the hypothesis that the voting server acts as a honest entity. The second set of properties, on the other hand, is considered as satisfied in the context where individual voting credentials are used to cast a ballot and where those credentials were correctly distributed to voters. This captures the hypothesis that the voting protocol relies on a honest registrar. Sections 3.2 and 3.3 propose two theorems showing that our trace properties imply verifiability respectively to the security contexts evoked. Note that these theorems only consider protocols where revoting is prohibited. This sections proposes the formal definition of verifiability and the general security assumptions, expressed as hypothesis, we make regarding protocols in our scope. Towards this end, we begin by describing some generic elements (sets, functions, events...) one can expect from a voting protocol and then we provide the formal definition of verifiability.

From now on, we consider ourselves in the symbolic world with processes[8] that model protocols and events[9] triggered by such processes when reaching some states that we will define. We remind the reader that we call *trace* the execution of a protocol. Let $\mathcal{P}$ be a process that models a voting protocol.

[7] See section 1.3.1.
[8] See section 1.1.4 for the definition of processes.
[9] See section 1.1.4 for the definition of events.

3.1.1 Sets and multisets

We assume the following sets can be defined from an election scheme and thus for its model, $\mathcal{P}$:

- $\mathcal{V}$: the *valid votes* set. This set represents the valid voting options of a given election.
- $\mathcal{B}$: the *valid ballots* set. Valid ballots are well-formed ballots regarding a specific voting scheme.
- $\mathcal{C}$: the *valid credentials* set. This set can be empty. It represents the sets of valid credentials (generated by a registrar for instance) used by voters to cast ballots during the election.
- $\mathcal{N}$: the *valid identifiers* set. This set contains the names (or identifiers) of all voters as decided by the election administrator.
- $\mathcal{R}$: the *elections results* space. It holds all valid election results.

Those sets are generic enough to be applied to a wide range of voting protocols.

Before going further, we also need to define *multisets*. Multisets generalize the concept of *sets* by allowing multiple instances of their elements. For instance: $\{\{a, a, b\}\}$ and $\{\{a, b\}\}$ define different multisets (they nonetheless define the same set). Moreover, the *order* of multisets elements does not matter: $\{\{a, a, b\}\}$ and $\{\{a, b, a\}\}$ define the same multiset. If E is a set, we note $\mathbb{P}_{\mathsf{m}}(E)$ the power set of E-multisets (multisets with elements from E). We also note $\subset_{\mathsf{m}}$ the multiset inclusion. With this definition in mind, we can define the following sets and multisets:

- The set of *honest voters*: $\mathsf{HV} \subset \mathcal{N}$.
- The set of *honest voters who verified their vote* (if there is a verification specified in $\mathcal{P}$): $\mathsf{HV}_{\mathsf{CH}} \subset \mathsf{HV}$.
- The set of *corrupted voters*: $\mathsf{C} \subset \mathcal{N}$. Note that this set is the complement of HV. So $\mathsf{HV} \sqcup \mathsf{C} = \mathcal{N}$
- The multiset of *votes cast by honest voters*: $\mathcal{V}_{\mathsf{HV}} \in \mathbb{P}_{\mathsf{m}}(\mathcal{V})$. It is a $\mathcal{V}$-multiset containing all votes cast by honest voters.
- The multiset of *votes cast by honest voters who verified their vote*: $\mathcal{V}_{\mathsf{HV}_{\mathsf{CH}}} \subset_{\mathsf{m}} \mathcal{V}_{\mathsf{HV}}$.

3.1.2 Election related functions

We also assume the voting scheme modeled by $\mathcal{P}$ require the use of several (natural) functions related to the election.

- A *counting* function:

  $$\rho : \mathbb{P}_{\mathsf{m}}(\mathcal{V}) \to \mathcal{R}$$

  The counting function tallies votes and outputs a result. Typically, it can be a function tallying the number of votes received by each candidate in an election.

- A *ballot unwrapping* function:

  $$unwrap : \mathcal{B} \to \mathcal{V}$$

  An unwrapping function takes a valid ballot and outputs the vote it holds. For instance, in our scheme, ballots are unwrapped by being decrypted. By extension, we define the unwrapping function on a subset of ballots: $unwrap : \mathbb{P}_{\mathsf{m}}(\mathcal{B}) \to \mathbb{P}_{\mathsf{m}}(\mathcal{V})$, which takes a (multi)set of ballots and outputs the multiset of votes matching the individual unwrapping of each ballot of the original set.

- A *credential reading* function:

$$extract : \mathcal{B} \rightarrow \mathcal{C}$$

  This function is *optionally* defined, only if the voting scheme relies on the use of *voting credentials*. It allows reading or computing the voting credential from a valid ballot. For instance, in our voting scheme, the public bulletin board displays valid ballots containing the voter's verification key as the first element of the ballot. Our $extract$ function would be retrieving the first entry of a ballot.

Like the sets previously defined, those functions should - for the counting function - or could - for the others - be generic enough to be part of a broad spectrum of voting protocols.

3.1.3 Events and events-defined multisets

As we said, $\mathcal{P}$ models an election scheme. We assume that the model contains the following events, and that those events are correctly placed in the model, according to the protocol:

- $\mathsf{VOTER}(ID, cred, H)$: the voter $ID \in \mathcal{N}$ holds $cred \in \mathcal{C}$ and is honest.
- $\mathsf{VOTER}(ID, cred, C)$: the voter $ID \in \mathcal{N}$ holds $cred \in \mathcal{C}$ and is corrupted.
- $\mathsf{VOTE}(ID, v)$: the voter $ID \in \mathcal{N}$ votes for $v \in \mathcal{V}$. This event shall only be triggered by the process modeling a honest voter.
- $\mathsf{VERIFIED}(ID, v)$: the voter $ID \in \mathcal{N}$ verified their vote $v \in \mathcal{V}$ was taken into account.
- $\mathsf{GOING\_TO\_TALLY}(ID, cred, b)$: the ballot $b \in \mathcal{B}$ registered for $ID \in \mathcal{N}$ with $cred \in \mathcal{C}$ will be tallied. It has at least parameter b defined.

Note that if those events are not correctly placed in the model, that does not formally change the implications we are going to state in the following sections (and thus the theorems). Those will still be true. Yet, if the *model* is false, one could prove a protocol as secure when it facts it is not. We emphasize the fact that those events are abstract tools used to help prove the verifiability of a protocol but that we do make the assumption they appear where they are supposed to appear regarding our definitions in the protocol model.

Given those events, then for all traces $\mathcal{T}$ of protocol $\mathcal{P}$, we can give a formal definition of the multisets defined in section 3.1.1.

Voters

They are defined as a subset of $\mathcal{N}$. We have two kind of voters: the honest ones and rogue ones. Their status is defined by the event $\mathsf{VOTER}(*, *, *)$.

The protocol specification should be done so that only honest voters processes trigger events of the form $\mathsf{VOTER}(*, *, H)$. We can then define the *honest voter set* as follows:

$$\mathsf{HV} \quad := \quad \left\{ \; ID \in \mathcal{N} \mid \mathsf{VOTER}(ID, *, H) \in \mathcal{T} \; \right\}$$

Among the honest voters, some can verify their vote appears on the bulletin board. We define this subset as the *honest voters who verified their vote* set.

$$\mathsf{HV_{CH}} \quad := \quad \left\{ \; ID \in \mathcal{N} \;\middle|\; \exists\, v \in \mathcal{V} \;.\; \begin{pmatrix} \mathsf{VOTER}(ID, *, H) & \in \mathcal{T} \\ \mathsf{VERIFIED}(ID, v) & \in \mathcal{T} \end{pmatrix} \right\}$$

Finally, corrupted voters who are under the attacker control are represented by a process triggering events of the form $\mathsf{VOTER}(*, *, C)$

$$\mathcal{C} \quad := \quad \{\ ID \in \mathcal{N} \mid \mathsf{VOTER}(ID, *, C) \in \mathcal{T}\ \}$$

Multisets linking honest voters and their votes

We need to be able to link honest voters to their votes.

The pairs (ID,v) of *honest voters with their votes* are element of the following multiset:

$$\mathcal{V}_{\mathsf{HV}}^{\mathsf{ID}} \quad := \quad \left\{\!\left\{ \begin{array}{l|l} (ID, v) \in \mathcal{N} \times \mathcal{V} & ID \in \mathsf{HV} \\ & \mathsf{VOTE}(ID, v) \in \mathcal{T} \end{array} \right\}\!\right\}$$

We subsequently define the multiset $\mathcal{V}_{\mathsf{HV_{CH}}}^{\mathsf{ID}} \subseteq_{\mathsf{m}} \mathcal{V}_{\mathsf{HV}}^{\mathsf{ID}}$ of *voters who verified the bulletin board and their votes*:

$$\mathcal{V}_{\mathsf{HV_{CH}}}^{\mathsf{ID}} \quad := \quad \left\{\!\left\{ \begin{array}{l|l} (ID, v) \in \mathcal{N} \times \mathcal{V} & ID \in \mathsf{HV} \\ & \mathsf{VOTE}(ID, v) \in \mathcal{T} \end{array} \right\}\!\right\}$$

Votes from honest voters

They can be seen as a projection of the previous multisets.

Votes from honest voters are then defined as:

$$\mathcal{V}_{\mathsf{HV}} \quad := \quad \{\!\{\ v \in \mathcal{V} \mid \exists\, ID \in \mathsf{HV}.\ \mathsf{VOTE}(ID, v) \in \mathcal{T}\ \}\!\} \qquad =_{\mathsf{m}} snd(\mathcal{V}_{\mathsf{HV}}^{\mathsf{ID}})$$

And *votes from honest voters who verified them* as:

$$\mathcal{V}_{\mathsf{HV_{CH}}} \quad := \quad \{\!\{\ v \in \mathcal{V} \mid \exists\, ID \in \mathsf{HV_{CH}}.\ \mathsf{VOTE}(ID, v) \in \mathcal{T}\ \}\!\} \quad =_{\mathsf{m}} snd(\mathcal{V}_{\mathsf{HV_{CH}}}^{\mathsf{ID}})$$

Public Bulletin Board

Finally, we define the public bulletin board as the multiset of ballots about to be tallied:

$$\mathsf{P_{BB}} \quad := \quad \{\!\{\ b \in \mathcal{B} \mid \mathsf{GOING\_TO\_TALLY}(*, *, b) \in \mathcal{T}\ \}\!\}$$

3.1.4 Security assumptions and hypothesis on a voting protocol

Some generic security assumptions can be made regarding a voting protocol. In our study, we only considered voting protocols where revoting is forbidden, hence we must make some assumptions about the voter's behaviour. We also considered voting scheme with a publicly displayed ballot box, thus we can also make some assumptions on the ballots contained in it. Finally, the counting function of the election is supposed to authorize partial tallying of votes (meaning we can count separately count subsets of the whole election votes without influencing the election final result).

We will here expose those assumptions we make on voting protocols. We believe they are generally satisfied by most e-voting schemes where revote is forbidden. These hypothesis are not provable by ProVerif but we will explain why we assume them to be trivial to infer.

Notation. Given a multiset $\mathcal{M}$ and an element $e \in \mathcal{M}$, we note $mult_{\mathcal{M}}(e)$ the number of occurrences of e in $\mathcal{M}$ (*i.e.* its multiplicity).

The following properties shall be satisfied for all traces $\mathcal{T} \in Trace(\mathcal{P})$.

Honest voter behaviour

We suppose here that a honest voter ID votes at most once and that if they verify their vote appears on the public bulletin board, they indeed previously voted.

$$\begin{aligned} &\forall\, ID \in \mathsf{HV}.\ mult_{\mathcal{T}}(\mathsf{VOTE}(ID, *)) \leqslant 1 && (1)\\ &\forall\, ID \in \mathsf{HV},\ \forall v \in \mathcal{V}.\ \mathsf{VERIFIED}(ID, v) \implies \mathsf{VOTE}(ID, v) && (2) \end{aligned} \tag{HV}$$

This assumption is quite reasonable as long as the protocol does not authorize revoting (1). (2) is actually more of a sanity check on the protocol specification $\mathcal{P}$ since it implies a honest voter verifies a vote they actually previously voted for. Any process modeling a honest voter behaviour shall reflect that.

Uniqueness of a ballot on the public bulletin board

We suppose that the public bulletin board ($\mathsf{P_{BB}}$) visible by everyone is the same for all. This hypothesis is commonly made when considering verifiable protocols, since it would require all displaying devices - or a DNS - to be under the attacker control to corrupt the public bulletin board display for each voter and election auditor.

It has been proven [81] that if ballots displayed on a public bulletin board are not all different from one another, then a protocol's verifiability is vulnerable to clash attacks: voters can obtain the assurance that their vote was taken into account by obtaining exactly the same proof as other voters. For instance, two different voters could have the same receipt and consider the same unique ballot as theirs. We suppose the protocol $\mathcal{P}$ to have some prevention against clash attacks. Therefore we assume that each visible ballot on the bulletin board is different from all others.

$$\forall\, b \in \mathcal{B}.\ mult_{\mathcal{T}}(\mathsf{GOING\_TO\_TALLY}(*, *, b)) \leqslant 1 \tag{UoB-PBB}$$

This property shall be guaranteed as long as at least one election auditor is honest. We already explained in section 2.3.1 why we think this hypothesis is reasonable, at least for our voting scheme - but the same argument could be used for other voting protocols.

Partial tallying of the election counting function

The counting function ρ of the election must ensure partial tallying as defined in [38]. We assume a ballot box that is arbitrarily partitioned. Each part is tallied and the final tally is computed by the combination of all "sub-tally". Then we assume that the result is the same for all arbitrary partition of our ballot box.

$$\begin{aligned} &\forall\, S_1,\ S_2 \in \mathbb{P}_{\mathsf{m}}(\mathcal{V}).\ \rho(S_1 \cup S_2) = \rho(S_1) \circ \rho(S_2) && (1)\\ &\qquad\qquad \text{with } \circ : \mathsf{R} \times \mathsf{R} \to \mathsf{R} \text{ (commutative)} \\ &\text{This implies that:} \\ &\forall\, \mathsf{P_{BB1}},\ \mathsf{P_{BB2}} :\ \text{Public Bulletin Boards. } \mathsf{P_{BB1}} \cap \mathsf{P_{BB2}} = \varnothing \\ &\implies \rho(unwrap(\mathsf{P_{BB1}} \cup \mathsf{P_{BB2}})) = \rho(unwrap(\mathsf{P_{BB1}})) \circ \rho(unwrap(\mathsf{P_{BB2}})) && (2) \end{aligned} \tag{PT}$$

Such a property is usually satisfied by the counting function of most voting protocols. For instance, the trivial function that counts the number of votes per candidates satisfies the partial tallying property.

However, this hypothesis is not satisfied by the Condorcet vote or Australia's Single Transferable Vote for example.

From now on, we assume the voting protocols under consideration satisfy all of these hypothesis.

3.1.5 Verifiability

Regarding the informal definition of verifiability stated in 2.4.1 that we borrowed from [38], we can now formally define the verifiability property.

Definition 3 (Verifiability)**.** Let $\mathcal{P}$ be a process modeling a voting protocol with ρ be the tallying function of this protocol. We say that $\mathcal{P}$ is *verifiable* if for all $\mathcal{T} \in Trace(\mathcal{P})$:

$$\exists\, \mathcal{V}_{\mathsf{count}} \subset_{\mathsf{m}} \mathcal{V}_{\mathsf{HV}} \backslash \mathcal{V}_{\mathsf{HV_{CH}}},\ \exists\, \mathcal{A} \in \mathbb{P}_{\mathsf{m}}(\mathcal{V}). \quad \begin{array}{ll} & \rho(unwrap(\mathsf{P_{BB}})) = \rho(\mathcal{V}_{\mathsf{HV_{CH}}}) \circ \rho(\mathcal{V}_{\mathsf{count}}) \circ \rho(\mathcal{A}) \\ \wedge & |\mathcal{A}| \leqslant |\mathsf{C}| \end{array} \tag{V}$$

Indeed, we previously defined $\mathsf{P_{BB}}$ as the multiset of ballots about to be tallied and $unwrap$ as the function that unwraps the election ballots. Thus $\rho(unwrap(\mathsf{P_{BB}}))$ is our election result. $\mathcal{V}_{\mathsf{HV_{CH}}}$ is the multiset of votes cast by the honest voters who verified their vote and $\mathcal{V}_{\mathsf{HV}} \backslash \mathcal{V}_{\mathsf{HV_{CH}}}$ the multiset of votes cast by honest voters who did not verify their vote. Our formal definition states that the result of the election is equal to the tally of:

- All votes cast by honest voters who verified their vote: $\mathcal{V}_{\mathsf{HV_{CH}}}$.
- A subset of votes from honest voters who did not verify their vote: $\mathcal{V}_{\mathsf{count}}$. It is only a subset and not the whole set because ballots from honest voters who did not verify their vote could be deleted.
- Another $\mathcal{V}$-multiset: $\mathcal{A}$, the number of elements of $\mathcal{A}$ being less than the number of corrupted voters (elements of C). This captures the fact that the protocol guarantees that the number of corrupted ballots cannot exceed the number of corrupted voters.

Unfortunately, this property is impossible to express in the ProVerif calculus, for the tool cannot "count" terms appearing on a trace. The other tool that could be used to "count" elements, the Tamarin prover, cannot handle the equational theory describing our randomizable encryption scheme, so there was no direct way to automatically prove the verifiability of our protocol.

However, as stated in our introduction to this chapter, in some automated analysis of voting protocols [39, 42], we can observe that the proof of a protocol verifiability is achieved indirectly with the proof of some other properties provable with ProVerif - cast as intended or tallied as cast properties for instance. Yet, [39] does not prove the verifiability of the protocol it studies and [42] does prove an implication of verifiability but for a weaker definition than the one we aim at (the third condition does not appear). We worked on defining sets of ground properties, protocol-independent and easily provable in Proverif, which imply the verifiability in order to automatize the security proof of a voting scheme. The first set is satisfied when the authentication credential are not compromised, the second one when the voting credential are not corrupted.

3.2 Verifiability based on correct authentication

If a voting server is considered as honest, we could expect the ballot casting process to be done with respect to the election rules. More specifically, if a ballot is registered for a specific voter under the authentication credentials provided by the voting server, we shall expect it to indeed come from this voter. This situation translates as several hypothesis and properties, specific to a honest voting server behaviour, which we state in this section. We then show that these imply verifiability.

3.2.1 Trace properties satisfied in the context of a correct authentication

We formalize here several properties that should be satisfied when the authentication of a voter to a - honest - voting server is correctly performed. First, we express the fact that, if a voting server is honest, then all ballots displayed on the public bulletin board come from different voters. Then we formally express what the behaviour of a honest voting server should be. finally we formalize the Cast as Intended and Tallied as Cast properties in the context of a correct authentication from the voter to the voting server.

Proper voter list

Since we do not consider revote, a honest voting server should guarantee that ballots displayed on the public bulletin board can be expected to all come from different voters.

As stated in section 3.1, we define the bulletin board with the event $\mathsf{GOING\_TO\_TALLY}(*,*,*)$, so we formalize this hypothesis as follows:

$$\forall\, ID_i,\ cred_i,\ b_i\ (i=1,2).$$
$$\left(\begin{array}{ll} \mathsf{GOING\_TO\_TALLY}(ID_1, cred_1, b_1) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(ID_2, cred_2, b_2) & \in \mathcal{T} \end{array}\right) \Longrightarrow \left(\begin{array}{lcl} ID_1 & \neq & ID_2 \\ \vee\ cred_1 & \neq & cred_2 \\ \vee\ b_1 & = & b_2 \end{array}\right) \qquad \text{(PVL)}$$

With this property, we state that ballots come from different voter - whether we base this differentiation on the authentication credential ID or the voting credential $cred$ - or are the same ones.

Honest voting server behaviour

A honest voting server shall guarantee that each voter cast their ballot once using one voting credential and that neither the voter's identifier nor the voting credential have previously been used to cast a ballot.

We formalize this as follows:

$$\forall\, ID_i,\ cred_i\ (i=1,2).$$
$$\left(\begin{array}{ll} \mathsf{GOING\_TO\_TALLY}(ID_1, cred_1, *) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(ID_2, cred_2, *) & \in \mathcal{T} \end{array}\right) \Longrightarrow \begin{array}{l} \left(\begin{array}{lcl} ID_1 & = & ID_2 \\ cred_1 & = & cred_2 \end{array}\right) \\ \vee\ \left(\begin{array}{lcl} ID_1 & \neq & ID_2 \\ cred_1 & \neq & cred_2 \end{array}\right) \end{array} \qquad \text{(HS)}$$

Note that even if those two trace properties can be expressed in ProVerif, it does not mean that the tool can prove them. In our security analysis for instance, the proof did not terminate. Nonetheless, the honest voting server model shall easily show that these are guaranteed

Valid credential (ID)

If a ballot is displayed on the public bulletin board, it shall come from a valid voter - honest or rogue - who was authenticated with their identifier ID.

$$\forall\, b \in \mathsf{P_{BB}}.\ \exists\, ID \in \mathcal{N}.\ \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T} \qquad (1)$$

$$\forall\, b \in \mathcal{B},\ \forall\, ID \in \mathcal{N}.\quad \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T} \Longrightarrow \mathsf{VOTER}(ID, *, *) \in \mathcal{T} \qquad (2) \qquad \text{(VC-ID)}$$

This property could be seen as a sanity check for the protocol specification for it states that if he event $\mathsf{GOING\_TO\_TALLY}(*, , b)$ is triggered, it shall have been with a parameter ID (1) and that this ID is indeed assigned to an eligible voter (2).

Cast as intended (ID)

We expect that whenever a ballot registered for a honest voter ID is going to be tallied, then indeed the voter ID voted for what the ballot wraps.

$$\forall\, b \in \mathcal{B},\ \forall\, ID \in \mathcal{N} :$$

$$\left(\begin{array}{ll} \mathsf{GOING\_TO\_TALLY}(ID, *, b) & \in \mathcal{T} \\ \mathsf{VOTER}(ID, *, H) & \in \mathcal{T} \end{array}\right) \implies \exists\, v \in \mathcal{V}.\ \left(\begin{array}{l} \mathsf{VOTE}(ID, v) \in \mathcal{T} \\ unwrap(b) = v \end{array}\right) \tag{CAI-ID}$$

Tallied as cast (ID)

The following property states that whenever a honest voter identified by ID verifies their vote, then there is indeed a ballot registered for ID on the public bulletin board wrapping ID's vote which is going to be tallied in the final result.

$$\forall\, v \in \mathcal{V},\ \forall\, ID \in \mathcal{N} :$$

$$\mathsf{VERIFIED}(ID, v) \in \mathcal{T} \implies \exists\, b \in \mathcal{B}.\ \left(\begin{array}{l} \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T} \\ unwrap(b) = v \end{array}\right) \tag{TAC-ID}$$

The satisfaction of those five properties implies the verifiability of a protocol if the voting server is honest - and if the authentication process is not compromised - as we explain it in our next section.

3.2.2 A theorem for verifiability based on a correct authentication

We can expect a voting scheme to be verifiable as long as the authentication credentials have not been compromised and the voting server is honest. We took this context into account by formalizing it with hypothesis and we provided five trace properties that, if satisfied, are enough to imply verifiability.

Theorem 1 (Verifiability based on ID)**.** *We assume $\mathcal{P}$ satisfies the hypothesis PT, HV and UoB-PBB.*

If $\mathcal{P}$ satisfies the proper voter list (PVL), honest server behaviour (HS), valid credential (VC-ID), cast as intended (CAI-ID) and tallied as cast (TAC-ID) properties, then $\mathcal{P}$ achieves verifiability (V).

3.2.3 Proof

The proof of the theorem is actually quite intuitive. We begin by defining subsets of the public bulletin board: ballots from honest voters and ballots from rogue voters. We then define a function mapping the public bulletin board to voters and their votes. We will prove its injectivity on the set of honest voters and their votes. This ensures the condition that a subset of votes from honest voters who did not verify their vote will be tallied. We also prove that this function defines a bijection between the ballots registered for honest voters who verified their votes and the votes from said voters. Finally, thanks to the fact that

we are in the context of a honest server, we explain why the number of rogue votes does not exceed the number of corrupted voters. Last but not least, we will prove that our subsets defined at the beginning of our proof define a partition of the public bulletin board which allows us to conclude on the proof of the theorem.

Subsets of the public bulletin board:

Our formal definition of verifiability (V) relies on the definition of the sets of honest voters - who did or did not verify their votes - corrupted voters and the multisets of votes cast by all different kind of voters. We already defined such sets and multisets in section 3.1.3 with the events we assume to be present in the specification of our protocol model.

Following this models, we define three sub(multi)sets of our bulletin board. As a reminder, the *public bulletin board* was defined in section 3.1 as the multiset of ballots appearing on the events $\mathsf{GOING\_TO\_TALLY}(*, *, b)$. Since the voting server is honest, it is relevant to divide our bulletin board into ballots that were registered for honest voters and the ones that were registered for corrupted voters.

The sub-bulletin board matching the *ballots registered for honest voters* is then defined as:

$$\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB}-\mathsf{HV}} \quad := \quad \{\{\ b \in \mathsf{P}_{\mathsf{BB}} \mid \exists\, ID \in \mathsf{HV}.\ \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T}\ \}\}$$

We can also define the sub-bulletin board of *ballots registered for honest voters who verified their votes*:

$$\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB}-\mathsf{HV}_{\mathsf{CH}}} \quad := \quad \{\{\ b \in \mathsf{P}_{\mathsf{BB}} \mid \exists\, ID \in \mathsf{HV}_{\mathsf{CH}}.\ \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T}\ \}\}$$

And finally, we have *ballots registered for corrupted voters*:

$$\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB}-\mathsf{C}} \quad := \quad \{\{\ b \in \mathsf{P}_{\mathsf{BB}} \mid \exists\, ID \in \mathcal{C}.\ \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T}\ \}\}$$

The public bulletin board is a set

We will start our proof by explaining why we can say that $\underline{\mathsf{P}_{\mathsf{BB}} \text{ is a set}}$:

Proof. By UoB-PBB, we have that $mult_{\mathcal{T}}(\mathsf{GOING\_TO\_TALLY}(*, *, b)) \leqslant 1$ for any $b \in \mathcal{B}$. Thus we have that $\forall\, b \in \mathcal{B}, mult_{\mathsf{P}_{\mathsf{BB}}}(b) = 1$, so P_{BB} is a set.

By extension, since $\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB}-\mathsf{HV}}$, $\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB}-\mathsf{HV}_{\mathsf{CH}}}$ ans $\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB}-\mathsf{C}}$ are sub-multisets of P_{BB}, we can also state that they are sets. □

Each vote displayed on the bulletin board was cast by a different voter

We also prove that $\underline{\forall\, ID \in \mathcal{N}.\ mult_{\mathcal{T}}(\mathsf{GOING\_TO\_TALLY}(ID, *, *)) \leqslant 1}$:

Proof. Let's suppose that there exists $ID \in \mathcal{N}$ such that $mult_{\mathcal{T}}(\mathsf{GOING\_TO\_TALLY}(ID, *, *)) \geqslant 2$. Then, by definition of the event $\mathsf{GOING\_TO\_TALLY}$:

$$\exists\, b_1,\ b_2 \in \mathcal{B}.\ \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(ID, *, b_1) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(ID, *, b_2) & \in \mathcal{T} \end{pmatrix}$$

By UoB-PBB, we can state that $b_1 \neq b_2$ and by PVL, we can also state that:

$$\exists\, cred_1,\ cred_2 \in \mathcal{B}.\ \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(ID, cred_1, b_1) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(ID, cred_2, b_2) & \in \mathcal{T} \\ b_1 \neq b_2 & \\ cred_1 \neq cred_2 & \end{pmatrix}$$

Which is a contradiction according to HS. □

A bijection between $\mathsf{P_{BB}}$ and $\mathcal{N}_{\mathsf{GTT}}$

We now define $\mathcal{N}_{\mathsf{GTT}} := \{\{\ ID \in \mathcal{N} \mid \mathsf{GOING\_TO\_TALLY}(ID, *, *) \in \mathcal{T}\ \}\}$ Let's prove that we have a bijection between $\mathsf{P_{BB}}$ and $\mathcal{N}_{\mathsf{GTT}}$.

Proof. First, according to the previous result, we can state that $\mathcal{N}_{\mathsf{GTT}}$ is a set since all of its elements are of multiplicity 1.

By VC-ID, we have that for all $b \in \mathsf{P_{BB}}$, there is an $ID \in \mathcal{N}$ such that $\mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T}$ and since UoB-PBB is verified, this ID is uniquely defined. Moreover, this ID is in $\mathcal{N}_{\mathsf{GTT}}$.

Now for all $ID \in \mathcal{N}_{\mathsf{GTT}}$, by definition of the event $\mathsf{GOING\_TO\_TALLY}$, there exists a $b \in \mathcal{B}$ such that $\mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T}$, so $b \in \mathsf{P_{BB}}$. And since we proved that it is impossible to have $b' \neq b$ such that $\mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T}$ and $\mathsf{GOING\_TO\_TALLY}(ID, *, b') \in \mathcal{T}$, we can also state that this b is uniquely defined. □

Defining a map between $\mathsf{P^{ID}_{BB-HV}}$ and $\mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}}$

We define φ as follows:

$$\begin{array}{rccl} \varphi: & \mathsf{P^{ID}_{BB-HV}} & \to & \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}} \\ & b & \mapsto & (ID, v).\ \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(ID, *, b) & \in \mathcal{T} \\ \mathsf{VOTE}(ID, v) & \in \mathcal{T} \\ unwrap(b) = v & \end{pmatrix} \end{array}$$

We begin by proving that φ is well defined.

Proof. By definition of $\mathsf{P^{ID}_{BB-HV}}$ and HV, we have the following implications:

$$\begin{array}{rcl} b \in \mathsf{P^{ID}_{BB-HV}} & \Rightarrow & \exists\, ID \in \mathsf{HV}.\ \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T} \\ & \Rightarrow & \exists\, ID \in \mathsf{HV}.\ \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(ID, *, b) & \in \mathcal{T} \\ \mathsf{VOTER}(ID, *, H) & \in \mathcal{T} \end{pmatrix} \end{array}$$

Since CAI-ID is verified:

$$\begin{array}{rcl} b \in \mathsf{P^{ID}_{BB-HV}} & \Rightarrow & \exists\, ID \in \mathsf{HV}.\ \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T} \\ & \Rightarrow & \exists\, v \in \mathcal{V}.\ \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(ID, *, b) & \in \mathcal{T} \\ \mathsf{VOTE}(ID, v) & \in \mathcal{T} \\ ID \in \mathsf{HV} & \\ unwrap(b) = v & \end{pmatrix} \end{array}$$

Also, thanks to the demonstration of the bijection between $\mathsf{P_{BB}}$ and $\mathcal{N}_{\mathsf{GTT}}$, we have that $\forall\, b \in \mathsf{P_{BB}}.\ \exists!\, ID \in \mathcal{N}_{\mathsf{GTT}}$ such that $\mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T}$. Hence the $ID \in \mathcal{N}$ such that $\mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T}$ is uniquely defined for each given $b \in \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}}$. And since HV is verified, we can also state that for a given $ID \in \mathsf{HV}$, there is a unique $v \in \mathcal{V}$ such that $\mathsf{VOTE}(ID, v) \in \mathcal{T}$. So, the pair $(ID, v) \in \mathcal{N} \times \mathcal{V}$ is uniquely defined and is indeed an element of $\mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}}$. □

φ is injective

Let's see why we can state that φ is injective.

Proof. Let's suppose that we have b_1 and b_2, elements of $\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}}$ such that $\varphi(b_1) = \varphi(b_2) = (ID, v)$. By definition of φ:

$$\varphi(b_1) = \varphi(b_2) = (ID, v) \Rightarrow \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(ID, *, b_1) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(ID, *, b_2) & \in \mathcal{T} \end{pmatrix}$$

But, as previously proven: $mult_{\mathcal{T}}(\mathsf{GOING\_TO\_TALLY}(ID, *, *)) \leqslant 1$, so $b_1 = b_2$.
Hence:

$$\varphi(b_1) = \varphi(b_2) \Rightarrow b_1 = b_2$$

□

Finding a subset of $\mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}} \backslash \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}}$ that was tallied

Let's also prove that $\varphi(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}) \subset \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}} \backslash \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}}$.

Proof. Let $b \in \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}$. By definition, we have that:

$$\exists\, ID \in \mathsf{HV}.\ \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(ID, *, b) & \in \mathcal{T} \\ \mathsf{VERIFIED}(ID, *) & \notin \mathcal{T} \end{pmatrix}$$

$$\Rightarrow \quad \begin{pmatrix} ID \in \mathsf{HV} \backslash \mathsf{HV_{CH}} \\ \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T} \end{pmatrix}$$

$$\Rightarrow \quad \varphi(b) \in \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}} \backslash \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}}$$

□

A bijection between $\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}$ and $\mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}}$

Let's now prove that $\varphi_{|\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}}$ is a bijection between $\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}$ and $\mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}}$.

Proof. The injectivity of $\varphi_{|\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}}$ is a consequence of φ's injectivity.

By definition: Let's consider $v \in \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}}$. By definition, and since HV is verified:

$$(ID, v) \in \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}} \quad \Rightarrow \quad \begin{pmatrix} ID \in \mathsf{HV_{CH}} \\ \mathsf{VOTE}(ID, v) \in \mathcal{T} \end{pmatrix}$$

$$\Rightarrow \quad \begin{pmatrix} ID \in \mathsf{HV} \\ \mathsf{VOTE}(ID, v) & \in \mathcal{T} \\ \mathsf{VERIFIED}(ID, v) & \in \mathcal{T} \end{pmatrix}$$

So by TAC-ID, we can state that:

$$
\begin{aligned}
(ID, v) \in \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}} \quad &\Rightarrow \quad \exists b \in \mathcal{B}. \left(\begin{array}{ll} ID \in \mathsf{HV_{CH}} & \\ \mathsf{VOTE}(ID, v) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(ID, *, b) & \in \mathcal{T} \\ unwrap(b) = v & \end{array} \right) \\
&\Rightarrow \quad \exists b \in \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}. \, \varphi(b) = (ID, v) \\
&\Rightarrow \quad \varphi(b) \text{ is surjective}
\end{aligned}
$$

□

So, $\varphi : \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \to \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}}$ is an injection such that:

$$
\begin{array}{l}
snd \circ \varphi = unwrap \\
\varphi_{|\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}} : \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}} \to \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}} \text{ is a bijection.}
\end{array}
$$

Controlling that $|\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}| \leqslant |\mathcal{C}|$

We now prove that $\underline{|\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}| \leqslant |\mathcal{C}|}$:

Proof. By definition of $\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}$:

$$
|\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}| = \left| \; \left\{ \; b \in \mathcal{B} \;\middle|\; \exists \, ID \in \mathcal{C} \mid \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T} \; \right\} \; \right|
$$

Thanks to VC-ID, we have that:

$$
|\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}| = \left| \left\{ \; b \in \mathcal{B} \;\middle|\; \exists \, ID \in \mathcal{C} \;\middle|\; \begin{array}{ll} \mathsf{GOING\_TO\_TALLY}(ID, *, b) & \in \mathcal{T} \\ \mathsf{VOTER}(ID, *, C) & \in \mathcal{T} \end{array} \right\} \right|
$$

Now, as stated previously, we know that there is a bijection between $\mathsf{P_{BB}}$ and $\mathcal{N}_{\mathsf{GTT}}$. More precisely:

$$
\begin{array}{ll}
\forall \, ID \in \mathcal{C}. & \mathsf{GOING\_TO\_TALLY}(ID, *, *) \in \mathcal{T} \\
& \Rightarrow \exists! \, b \in \mathcal{B}. \, \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T} \\
\forall \, b \in \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}. & \exists! \, ID \in \mathcal{C}. \, \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T}
\end{array}
$$

This implies that:

$$
\begin{aligned}
|\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}| &= \left| \underbrace{\left\{ \; ID \in \mathcal{C} \;\middle|\; \exists \, b \in \mathcal{B} \mid \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T} \; \right\}}_{\subset \mathcal{C}} \right| \\
&\leqslant |\mathcal{C}|
\end{aligned}
$$

□

A partition of the public bulletin board

Now let's see why $\underline{\mathsf{P}_{\mathsf{BB}} = \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}} \sqcup (\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}) \sqcup \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}}$:

Proof. By definition of P_{BB}, and since it it a set as previously stated:

$$\mathsf{P}_{\mathsf{BB}} = \left\{ \; b \in \mathcal{B} \;\middle|\; \exists\, ID \in \mathcal{N}.\, \mathsf{GOING\_TO\_TALLY}(*, *, b) \in \mathcal{T} \; \right\}$$

Since we have VC-ID:

$$\mathsf{P}_{\mathsf{BB}} = \left\{ \; b \in \mathcal{B} \;\middle|\; \exists\, ID \in \mathcal{N}. \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(ID, *, b) & \in \mathcal{T} \\ \mathsf{VOTER}(ID, *, *) & \in \mathcal{T} \end{pmatrix} \right\}$$

$$= \left\{ \; b \in \mathcal{B} \;\middle|\; \exists\, ID \in \mathcal{N}. \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(ID, *, b) & \in \mathcal{T} \\ \mathsf{VOTER}(ID, *, H) & \in \mathcal{T} \end{pmatrix} \right\}$$
$$\sqcup \left\{ \; b \in \mathcal{B} \;\middle|\; \exists\, ID \in \mathcal{N}. \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(ID, *, b) & \in \mathcal{T} \\ \mathsf{VOTER}(ID, *, C) & \in \mathcal{T} \end{pmatrix} \right\}$$

$$= \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \sqcup \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}$$

$$= \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}} \sqcup (\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}) \sqcup \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}$$

□

Concluding the proof

Finally, since the counting function ρ ensures PT, we obtain V as follows ; let r be the election result:

Proof.

$$\begin{aligned} r &= \rho(unwrap(\mathsf{P}_{\mathsf{BB}})) \\ &= \rho(unwrap(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}} \sqcup (\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}) \sqcup \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}})) \\ &= \rho(unwrap(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}})) \circ \rho(unwrap(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}})) \circ \rho(unwrap(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}})) \\ &= \rho(snd(\varphi(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}))) \circ \rho(snd(\varphi(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}))) \circ \rho(snd(unwrap(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}))) \\ &= \rho(\mathcal{V}_{\mathsf{HV_{CH}}}) \circ \rho(snd(\varphi(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}))) \circ \rho(snd(unwrap(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}))) \end{aligned}$$

With $snd(\varphi(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}})) \subset_{\mathsf{m}} \mathcal{V}_{\mathsf{HV}} \backslash \mathcal{V}_{\mathsf{HV_{CH}}}$

and $|unwrap(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}})|_{\mathsf{m}} = |\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-C}}| \leqslant |C|$

□

3.3 Verifiability assuming a correct use of voting credentials

As stated in section 3.1, in order to extensively analyze the security of a voting scheme, we need to consider the case where no security assumption can be made on the voting server. In other terms, a voting protocol should have some measures that guarantees security even if the voting server is compromised. This is an important matter because if there is a massive corruption of authentication credentials - either because of a leak or if the voting server is under the attacker's control - and if there is no other control other than the voter's identification to the voting server, then a ballot box could be stuffed quite easily by casting and registering ballots instead of honest voters.

Some protocols (like Three Ballots or the Belenios suite) rely on the use of individual voting credentials provided by a registrar. Such credentials are used by the voter to compute a valid ballot and can be used by the voter to check their ballot was taken into account in the bulletin board. In the Three Ballots protocol, this is materialized by the voting sheet containing the three different ballots the voter has to use. The verification value is the one out of three random numbers on the voting sheet the voter chooses to keep and the verification process is performed by the voter by checking that this same number appears in one of the ballot on the bulletin board. In Belenios, the voting credential is the individual pair of signature keys used by the voter to compute their ballot and the verification value is the hash of their ballot. In Belenios RF - as in Belenios VS - the verification value is the signature verification key.

We already discussed the fact that if both the authentication credentials *and* the voting credentials are compromised, we cannot expect a protocol to be verifiable[10]. Indeed, an attacker has all the information needed to cast a ballot instead of a honest voter. Now that we have a set of security properties that guarantees verifiability when satisfied when *at least* the authentication credentials and their use are not compromised, we want to provide a similar set that also implies verifiability as long as the voting credentials are safe.

This section's purpose is to provide such a set. We put ourselves in the context of a honest use of voting credentials. We assume them to be *personal* to each voter and *well-distributed* (meaning that they are not under an attacker's control). The following set of trace properties are, as we believe, satisfied when the registrar honestly behaves and are quite symmetric to the properties exposed in the previous section 3.2.

3.3.1 An additional hypothesis on a honest registrar's behaviour

We assume the hypothesis mentioned in 3.1.4 (PT, HV, and UoB-PBB), are satisfied by the protocol $\mathcal{P}$. We add an hypothesis to our context, that cannot be expressed in the ProVerif calculus, but that appeared to us as a reasonable property that captures the behaviour of a honest registrar.

Honest registrar behaviour

If a registrar is honest, we also can assume that there are no more credentials than the number of eligible voter. Moreover, all credential are different from one another. We formalize this with the hypothesis:

$$\begin{array}{ll} \forall\, ID \in \mathcal{N}.\ mult_{\mathcal{T}}(\mathsf{VOTER}(ID, *, *)) \leqslant 1 & (1) \\ \forall\, cred \in \mathcal{C}.\ mult_{\mathcal{T}}(\mathsf{VOTER}(*, cred, *)) \leqslant 1 & (2) \\ \forall\, cred \in \mathcal{C}.\ \mathsf{VOTER}(*, cred, *) \Rightarrow \exists\, ID \in \mathcal{N}.\ \mathsf{VOTER}(ID, cred, *) & (3) \end{array} \quad \text{(HR)}$$

With these hypothesis, we ensure that linked to each credential, there is a voter (3) and that each voter receives at most one credential (1) that will be different from all other credentials (2).

[10] See section 3.1.

3.3.2 ProVerif provable properties

The four properties stated here can be expressed and proven in the ProVerif calculus. They are actually quite similar to the properties defined in section 3.2 although instead of relying on the variable ID, they rely on $cred$.

Public bulletin board verification

The voting credentials $cred$ used to cast a ballot *and* for voting verification shall be publicly visible from the bulletin board, and all different from one another. We state the following property on such credentials:

$$\begin{array}{l} \forall\, b \in \mathcal{B},\ \forall\, cred \in \mathcal{C}. \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) \implies extract(b) = cred \quad (1) \end{array}$$

(PBB-Verif)

$$\begin{array}{l} \forall\, b_1,\ b_2 \in \mathsf{P_{BB}}. \\ extract(b_1) = extract(b_2) \implies b_1 = b_2 \qquad (2) \end{array}$$

(1) ensures the voting credential used to cast a ballot can be read from the ballot on the bulletin board (with $extract$). (2) is verified as long as whenever a ballot is on the public bulletin board, there cannot be another ballot cast with the same credential. For instance, in Belenios VS, the voting credential used to compute *and* verify the ballot is the public signature verification key of the voter. If a voter checks their vote on the bulletin board, they have to look for their public key that appears as the first entry of their ballot. In our case, our $extract$ function is "reading the first entry of ballot b displayed on the bulletin board" (1). If we have two ballots on the bulletin board that present the same signature verification key, then it means that those ballots are in fact the same one because the signature keys are individually distributed to each voter and that revoting is forbidden.

Valid credential (cred)

A ballot displayed on the bulletin board should have been cast by a valid voter - honest or rogue - using a voting credential.

$$\forall\, b \in \mathsf{P_{BB}}.\ \exists\, cred \in \mathcal{C}.\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) \in \mathcal{T} \qquad (1)$$

$$\begin{array}{l} \forall\, b \in \mathcal{B},\ \forall\, cred \in \mathcal{C}. \quad \mathsf{GOING\_TO\_TALLY}(*, cred, b) \in \mathcal{T} \\ \qquad\qquad \implies \mathsf{VOTER}(*, cred, *) \in \mathcal{T} \qquad (2) \end{array}$$

(VC-cred)

Satisfying this property ensures that each ballot displayed on the public bulletin board was cast with a voting credential (1) and that this credential is owned by an eligible voter (2).

Cast as intended (cred)

The cast as intended property in a context where there in no guarantee that the voting server is honest changes a little. We say here that every ballot registered with a credential $cred$ held by a honest voter voter ID indeed wraps the voter's choice.

$$\forall\, b \in \mathcal{B},\ \forall\, cred \in \mathcal{C},\ \forall\, ID \in \mathcal{N}.$$

$$\left(\begin{array}{ll} \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \\ \mathsf{VOTER}(ID, cred, H) & \in \mathcal{T} \end{array}\right) \implies \exists\, v \in \mathcal{V}. \left(\begin{array}{l} \mathsf{VOTE}(ID, v) \in \mathcal{T} \\ unwrap(b) = v \end{array}\right)$$

(CAI-cred)

Tallied as cast (cred)

Like both previous properties, we also mirror the tallied as cast property to adapt it in the context of a honest registrar. Whenever a honest voter identified by ID verifies their vote, then there is indeed a ballot registered for the credential held by ID that is published on the public bulletin board which wraps ID's vote. Such a ballot is guaranteed to be a part of the final result.

$$\forall\, v \in \mathcal{V},\ \forall\, ID \in \mathcal{N}.$$
$$\mathsf{VERIFIED}(ID, v) \in \mathcal{T} \implies \exists\, b \in \mathcal{B},\ \exists\, cred \in \mathcal{C}.\ \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred, *) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \\ unwrap(b) = v & \end{array}\right) \quad \text{(TAC-cred)}$$

Satisfying those four properties implies the verifiability of a scheme assuming the voting credentials are honestly used and that they are publicly readable from the bulletin board.

3.3.3 A theorem for verifiability based on the correct use of voting credentials

In section 3.2, we assumed the authentication credential and their use were not compromised. It made sense in the context of a honest voting server. However, we also need to consider corruption scenarii compromising this assumption. We can still hope a voting scheme to hold as long as the voting credentials are not compromised. The following theorem states that in the context of a honest registrar, verifiability shall be achieved if three properties are satisfied.

Theorem 2 (Verifiability based on credential)**.** *We assume $\mathcal{P}$ satisfies the hypothesis PT, HV, UoB-PBB as well as HR.*

If $\mathcal{P}$ satisfies PBB-Verif, VC-cred, CAI-cred, TAC-cred, then $\mathcal{P}$ achieves Verifiability (V).

3.3.4 Proof

The proof of this theorem is quite similar to the previous one, for the properties and assumptions take voting credentials into account instead of authentication ones. Thus, its structure is more or less the same.

Subsets of the public bulletin board

Like the proof of our first theorem, we begin by defining a bunch of subsets of our bulletin board - that will be proven to be a partition of it. The public bulletin board was defined in section 3.1 as the multiset of ballots appearing on the events $\mathsf{GOING\_TO\_TALLY}(*, *, b)$. Since we consider the use of voting credentials as not compromised, our bulletin board partition will rely on those credentials and we will distinguish them whether they are held by honest or rogue voters.

The sub-bulletin board matching the *ballots registered for honest voters credentials* is then defined as:

$$\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}} := \left\{\!\!\left\{ b \in \mathsf{P}_{\mathsf{BB}} \;\middle|\; \begin{array}{l} \exists\, ID \in \mathsf{HV} \\ \exists\, cred \in \mathcal{C} \end{array} \cdot \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred, H) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \end{array}\right) \right\}\!\!\right\}$$

We can also define the sub-bulletin board of *ballots registered for honest voters who verified their votes*:

$$\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}} := \left\{\!\left\{ b \in \mathsf{P_{BB}} \;\middle|\; \begin{array}{l} \exists\, ID \in \mathsf{HV_{CH}} \\ \exists\, cred \in \mathcal{C} \end{array} \cdot \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred, H) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \end{array} \right) \right\}\!\right\}$$

And finally, we have *ballots registered for corrupted voters credentials*:

$$\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-C}} := \left\{\!\left\{ b \in \mathsf{P_{BB}} \;\middle|\; \begin{array}{l} \exists\, ID \in \mathsf{HV_{CH}} \\ \exists\, cred \in \mathcal{C} \end{array} \cdot \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred, H) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \end{array} \right) \right\}\!\right\}$$

The public bulletin board is a set

Just like in the previous proof, $\mathsf{P_{BB}}$ is a set. By extension, $\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}}$, $\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}}$ and $\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-C}}$ are also sets.

Each vote displayed on the bulletin board was cast using a different credential

Let's also prove that $\forall\, cred \in \mathcal{C}.\, mult_{\mathcal{T}}(\mathsf{GOING\_TO\_TALLY}(*, cred, *)) \leqslant 1$:

Proof. Let's suppose that there exists $cred \in \mathcal{C}$ such that $mult_{\mathcal{T}}(\mathsf{GOING\_TO\_TALLY}(*, cred, *)) \geqslant 2$. Then, by definition of the event GOING_TO_TALLY:

$$\exists\, b_1,\ b_2 \in \mathcal{B}. \left(\begin{array}{ll} \mathsf{GOING\_TO\_TALLY}(*, cred, b_1) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b_2) & \in \mathcal{T} \end{array} \right)$$

By UoB-PBB, we can state that $b_1 \neq b_2$. However, by PBB-Verif, we have that:

$$extract(b_1) = extract(b_2) \Rightarrow b_1 = b_2$$

Which is a contradiction. □

A bijection between $\mathsf{P_{BB}}$ and $\mathcal{N}_{\mathsf{GTT}}$

We now define the following multiset:

$$\mathcal{N}_{\mathsf{GTT}} := \left\{\!\left\{ ID \in \mathcal{N} \;\middle|\; \exists cred \in \mathcal{C}. \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred, *) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, *) & \in \mathcal{T} \end{array} \right) \right\}\!\right\}$$

Let's prove that we have a bijection between $\mathsf{P_{BB}}$ and $\mathcal{N}_{\mathsf{GTT}}$.

Proof. We need to prove first that $\mathcal{N}_{\mathsf{GTT}}$ is a set. Let's suppose there exists an $ID \in \mathcal{N}_{\mathsf{GTT}}$ such that $mult_{\mathcal{N}_{\mathsf{GTT}}}(ID) \geqslant 2$. By definition of $\mathcal{N}_{\mathsf{GTT}}$:

$$\exists\, cred_1,\ cred_2 \in \mathcal{C}. \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred_1, *) & \in \mathcal{T} \\ \mathsf{VOTER}(ID, cred_2, *) & \in \mathcal{T} \end{array} \right)$$

Which implies that $mult_{\mathcal{T}}(\mathsf{VOTER}(ID, *, *)) \geqslant 2$, which is in contradiction with HR.

By VC-cred, we have that for all $b \in \mathsf{P_{BB}}$, there is an $cred \in \mathcal{C}$ such that $\mathsf{GOING\_TO\_TALLY}(*, cred, b) \in \mathcal{T}$ and since UoB-PBB is verified, this $cred$ is uniquely defined. Still by VC-cred:

$$b \in \mathsf{P_{BB}} \Rightarrow \exists!\, cred \in \mathcal{C}. \begin{pmatrix} \mathsf{VOTER}(*, cred, *) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \end{pmatrix}$$

$$\text{By HR} \Rightarrow \begin{matrix} \exists\, ID & \in \mathcal{N} \\ \exists!\, cred & \in \mathcal{C} \end{matrix} \cdot \begin{pmatrix} \mathsf{VOTER}(ID, cred, *) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \end{pmatrix}$$

$$\Rightarrow \begin{matrix} \exists\, ID & \in \mathcal{N}_{\mathsf{GTT}} \\ \exists!\, cred & \in \mathcal{C} \end{matrix} \cdot \begin{pmatrix} \mathsf{VOTER}(ID, cred, *) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \end{pmatrix}$$

By HR, $mult_{\mathcal{T}}(\mathsf{VOTER}(*, cred, *)) = 1$ so $ID \in \mathcal{N}_{\mathsf{GTT}}$ is uniquely defined.

Now for all $ID \in \mathcal{N}_{\mathsf{GTT}}$, since HR ensures that $mult_{\mathcal{T}}(\mathsf{VOTER}(ID, *, *)) = 1$ and by definition of the event $\mathsf{GOING\_TO\_TALLY}$, we have the following implications:

$$ID \in \mathcal{N}_{\mathsf{GTT}} \Rightarrow \exists!\, cred \in \mathcal{C}. \begin{pmatrix} \mathsf{VOTER}(ID, cred, *) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, *) & \in \mathcal{T} \end{pmatrix}$$

$$\text{By VC-cred} \Rightarrow \begin{matrix} \exists\, b & \in \mathcal{B} \\ \exists!\, cred & \in \mathcal{C} \end{matrix} \cdot \begin{pmatrix} \mathsf{VOTER}(ID, cred, *) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \end{pmatrix}$$

$$\Rightarrow \begin{matrix} \exists\, b & \in \mathsf{P_{BB}} \\ \exists!\, cred & \in \mathcal{C} \end{matrix} \cdot \begin{pmatrix} \mathsf{VOTER}(ID, cred, *) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \end{pmatrix}$$

Thanks to the previous result, we can also state that this b is uniquely defined. □

Defining a map between $\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}}$ and $\mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}}$

We define ψ as follows:

$$\begin{array}{rccl} \psi: & \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}} & \to & \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}} \\ & b & \mapsto & (ID, v).\, \exists\, cred \in \mathcal{C}. \begin{pmatrix} \mathsf{VOTER}(ID, cred, H) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \\ \mathsf{VOTE}(ID, v) & \in \mathcal{T} \\ unwrap(b) = v & \end{pmatrix} \end{array}$$

We begin by proving that ψ is well defined.

Proof. By definition of $\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}}$ and HV, we have the following implication:

$$b \in \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}} \Rightarrow \begin{matrix} \exists\, ID & \in \mathsf{HV} \\ \exists\, cred & \in \mathcal{C} \end{matrix} \cdot \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \\ \mathsf{VOTER}(ID, cred, H) & \in \mathcal{T} \end{pmatrix}$$

Since CAI-cred is verified:

$$b \in \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}} \Rightarrow \begin{array}{lll} \exists\, ID & \in \mathsf{HV} \\ \exists\, cred & \in \mathcal{C} \\ \exists\, v & \in \mathcal{V} \end{array} . \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred, H) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \\ \mathsf{VOTE}(ID, v) & \in \mathcal{T} \\ unwrap(b) = v & \end{array} \right)$$

In the previous demonstration of the bijection between $\mathsf{P_{BB}}$ and $\mathcal{N}_{\mathsf{GTT}}$, we saw that $\forall\, b \in \mathsf{P_{BB}},\ \exists!\, cred \in \mathcal{C}.\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) \in \mathcal{T}$ and that $\exists!\ ID \in \mathcal{N}_{\mathsf{GTT}}.\ \mathsf{VOTER}(ID, cred, *) \in \mathcal{T}$.So ID is uniquely defined for each given $b \in \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}}$. And since HV is verified, we can also state that for a given $ID \in \mathsf{HV}$, there is a unique $v \in \mathcal{V}$ such that $\mathsf{VOTE}(ID, v) \in \mathcal{T}$. So all in all, the pair $(ID, v) \in \mathcal{N} \times \mathcal{V}$ is uniquely defined and is indeed an element of $\mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}}$. □

ψ is injective

Let's see why we can state that ψ is injective.

Proof. Let's suppose that we have b_1 and b_2, elements of $\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}}$ such that $\psi(b_1) = \psi(b_2) = (ID, v)$. By definition of ψ:

$$\psi(b_1) = \psi(b_2) = (ID, v) \Rightarrow \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred_1, H) & \in \mathcal{T} \\ \mathsf{VOTER}(ID, cred_2, H) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred_1, b_1) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred_2, b_2) & \in \mathcal{T} \end{array} \right)$$

Since we have HR, we can state as previously done that $cred_1 = cred_2$. So $extract(b_1) = extract(b_2)$. Finally, by PBB-Verif, $b_1 = b_2$.

$$\psi(b_1) = \psi(b_2) = (ID, v) \Rightarrow b_1 = b_2$$

□

Finding a subset of $\mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}} \backslash \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}}$ that was tallied

Similarly than in the previous proof, we have that: $\psi(\mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{ID}}_{\mathsf{BB-HV_{CH}}}) \subset \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}} \backslash \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}}$.

A bijection between $\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}}$ and $\mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}}$

Let's now prove that $\psi_{|\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}}}$ is a bijection between $\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}}$ and $\mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}}$.

Proof. The injectivity of $\psi_{|\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}}}$ is a consequence of ψ's injectivity.

By definition: Let's consider $v \in \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}}$. By definition, and since HV is verified:

$$(ID, v) \in \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}} \Rightarrow \left(\begin{array}{l} ID \in \mathsf{HV_{CH}} \\ \mathsf{VOTE}(ID, v) \in \mathcal{T} \end{array} \right)$$

$$\Rightarrow \left(\begin{array}{ll} ID \in \mathsf{HV} & \\ \mathsf{VOTE}(ID, v) & \in \mathcal{T} \\ \mathsf{VERIFIED}(ID, v) & \in \mathcal{T} \end{array} \right)$$

So by TAC-cred, we can state that:

$$(ID, v) \in \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}} \Rightarrow \begin{array}{ll} \exists\, b & \in \mathcal{B} \\ \exists\, cred & \in \mathcal{C} \end{array} \cdot \left(\begin{array}{ll} ID \in \mathsf{HV_{CH}} & \\ \mathsf{VOTER}(ID, cred, H) & \in \mathcal{T} \\ \mathsf{VOTE}(ID, v) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \\ unwrap(b) = v & \end{array} \right)$$

$$\Rightarrow \exists b \in \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}}.\ \psi(b) = (ID, v)$$

$$\Rightarrow \psi(b) \text{ is surjective}$$

□

So, $\psi :\ \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}} \to \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV}}$ is an injection such that:

$$snd \circ \psi = unwrap$$
$$\psi_{|\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}}} :\ \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}} \to \mathcal{V}^{\mathsf{ID}}_{\mathsf{HV_{CH}}} \text{ is a bijection.}$$

Controlling that $|\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-C}}| \leqslant |\mathcal{C}|$

We now prove that $|\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-C}}| \leqslant |\mathcal{C}|$:

Proof. By definition of $\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-C}}$:

$$|\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-C}}| = \left| \left\{ b \in \mathsf{P_{BB}} \;\middle|\; \begin{array}{ll} \exists\, ID & \in \mathcal{C} \\ \exists\, cred & \in \mathcal{C} \end{array} \cdot \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred, C) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \end{array} \right) \right\} \right|$$

Now, as stated previously, we know that there is a bijection between $\mathsf{P_{BB}}$ and $\mathcal{N}_{\mathsf{GTT}}$. More precisely:

$$\begin{array}{ll} \forall\, ID \in \mathcal{C}. & \exists! \, cred \in \mathcal{C}. \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred, C) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, *) & \in \mathcal{T} \end{array} \right) \\ & \Rightarrow \exists! \, b \in \mathcal{B}.\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) \in \mathcal{T} \end{array}$$

$$\forall\, b \in \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-C}}. \quad \begin{array}{ll} \exists! \, ID & \in \mathcal{C} \\ \exists! \, cred & \in \mathcal{C} \end{array} \cdot \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred, C) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, *) & \in \mathcal{T} \end{array} \right)$$

This implies that:

$$|\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-C}}| = \left| \underbrace{\left\{ ID \in \mathcal{C} \;\middle|\; \begin{array}{ll} \exists\, b & \in \mathcal{B} \\ \exists\, cred & \in \mathcal{C} \end{array} \cdot \left(\begin{array}{ll} \mathsf{VOTER}(ID, cred, C) & \in \mathcal{T} \\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \end{array} \right) \right\}}_{\subset \mathcal{C}} \right|$$
$$\leqslant |\mathcal{C}|$$

□

A partition of the public bulletin board

Now let's see why $\underline{\mathsf{P}_{\mathsf{BB}} = \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}} \sqcup (\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}}) \sqcup \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-C}}}$:

Proof. By definition of P_{BB}, and since it it a set as previously stated:

$$\mathsf{P}_{\mathsf{BB}} = \left\{ \; b \in \mathcal{B} \;\middle|\; \exists\, ID \in \mathcal{N}.\, \mathsf{GOING\_TO\_TALLY}(*, *, b) \in \mathcal{T} \; \right\}$$

Since we have VC-cred:

$$\mathsf{P}_{\mathsf{BB}} = \left\{ \; b \in \mathcal{B} \;\middle|\; \exists\, cred \in \mathcal{C}.\, \begin{pmatrix} \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \in \mathcal{T} \\ \mathsf{VOTER}(*, cred, *) & \in \mathcal{T} \end{pmatrix} \right\}$$

Thanks to HR and like in previous proof:

$$= \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}} \sqcup \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-C}}$$

$$= \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}} \sqcup (\mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV}} \backslash \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-HV_{CH}}}) \sqcup \mathsf{P}^{\mathsf{cred}}_{\mathsf{BB-C}}$$

□

Concluding the proof

Finally, since the counting function ρ ensures PT, we obtain V just as in the proof of previous theorem.

Conclusion and discussion

We provided a formalization of the verifiability property. Unfortunately, this property cannot be expressed as a ProVerif security property. So we stated two different sets of *trace properties*[11] - one that applies in the context of a honest voting server, the other in the context of a honest registrar - that imply verifiability. The voting schemes under our theorems scope needed to satisfy some requirements:

- The ballot box of the election must be publicly displayed. This is not always the case for voting schemes, like in the Neuchâtel e-voting protocol [39].
- The users must authenticate themselves to the voting server with individual *authentication credentials*.
- *If* there is a registrar in the protocol's ecosystem, it mus provide individual *voting credentials* that are different from the authentication credentials and are used to compute a valid ballot. Such credentials are *a priori* either public or private.
- Revote is prohibited.

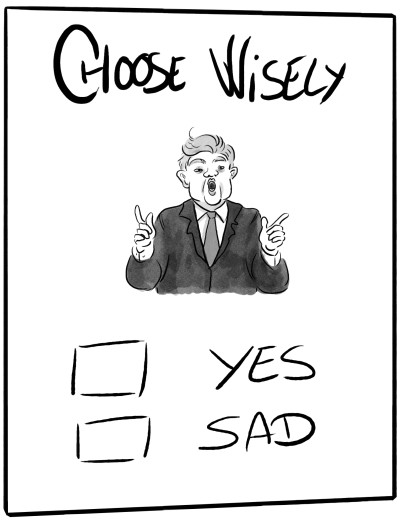
[11] See section 1.3.1.

This last requirement does restrict the application of our theorems for several existing voting protocols authorize revoting [32] - although revoting is prohibited in some countries such as France. Still, it appears that, by adding some kind of order among the cast ballots, we could still adapt our definition of verifiability - the considered ballots for tallying shall be the last ones cast for each voter - and of our properties to provide a more generic theorem that also take voting schemes authorizing revoting into account. This could imply using session number in prover tools, which can often not be expressed and/or managed.

Our next chapter will detail the security analysis of Belenios VS regarding privacy and verifiability. To this purpose, we apply both theorems provided in the chapter.

Chapter 4

Security analysis of Belenios VS



In chapter 2, we gave the specification of the Belenios VS protocol, a variant of Belenios RF that require the voter to use a voting sheet to cast their vote. The fact that a voter does not compute their vote from their own device anymore is justified by our security claim: Belenios VS is still verifiable and private even if the user's device is under the attacker control, something that was not guaranteed by either Belenios [38] or Belenios RF [29]. In order to be precise on our security claims, we considered several corruption cases that were summarized in Fig.2.10 from section 2.3.3. all of these corruption cases can be combined to produce a corruption scenarii. For instance we could consider the scenario where honest voters lost their passwords and the registrar is corrupted, or the one where the voting sheets are known by the attacker and the election private key is leaked. Given the high number of corruption scenarii we have, we automatized our proof using ProVerif.

Yet, in chapter 3, we stated why it was not possible to express the verifiability security property in the ProVerif calculus, which is why we provided two theorems that imply verifiability if some trace properties, easily expressible in ProVerif, are satisfied. The first section of this chapter focuses on how we applied our theorems to prove Belenios VS' verifiability. We go over the model choices and explore the results and what they can tell us. Mainly, the improvement compared to Belenios RF is that we now

can mistrust a domestic device and still have a verifiable protocol.

The final section deals with the privacy of our protocol. We explain the model choices we made - which were more drastic than for the verifiability models and present the results of our analysis. From the study of the attacks against our protocol's privacy, we were able to raise what appears to be a generic attack that would be made possible for any non verifiable voting scheme.

All models and proofs files are available at [1].

4.1 Verifiability of our protocol

We apply our two theorems from sections 3.2.2 and 3.3.3 to the security analysis of our voting protocol. Since we considered quite an extensive threat model[12], we needed to study our protocol for each corruption combination possible to define its limits regarding the verifiability. To this purpose, we scripted the ProVerif files generation to create all 60 plausible corruption scenarii. This also allowed us to study the unsuspected attacks against the verifiability of our protocol.

We begin this section by explaining the formal model of each one of our entities acting in our voting scheme. We then provide the formal properties matching our sets of properties defined in section 3.2 and in section 3.3. We finally present the results of our study.

4.1.1 ProVerif models

We model the protocol described in section 2.2 in the ProVerif calculus. As stated in section 2.3.1:

- Five entities are corruptible:
  - the registrar (R),
  - the voting server (VS),
  - the voter (V_k),
  - the voting device (VD),
  - the auditing device (AD).
- Two objects can be leaked to the attacker:
  - the authentication credentials ($\mathsf{ID}_k, \mathsf{pwd}_k$),
  - the voting sheet (VSh_i).
- The voter can forget to audit their voting sheet.
- The election private key can be output.

We need to study to what extent we can allow the corruption of our entities or information to be leaked. Thus we need to not only model our entities honest processes but also their corrupted behaviour by giving as much power as possible to the attacker.

As the reader can extrapolate from chapter 3, the verifiability of a voting scheme in our theorems does not depend on the tally process as long as we assume the election audit to be honest. Thus, we did not need to model the tally process in this particular study.

This sections provides some insights to understand our ProVerif models available at [1].

[12] See section 2.3.1.

Equational theory

We need to model the randomizable encryption and signature scheme described in section 2.2.2. Equations defining the functions $\mathsf{aenc_p}$ (2.1) and $\mathsf{sign_p}$ (2.2) could actually be used as they were previously stated, the randomization function $\mathsf{rand_p}$ had to be adapted.

We first tried modeling it with an equation of the form: $\mathsf{rand_p}(\mathsf{aenc_p}(m, \mathsf{pk}, r), r') = \mathsf{aenc_p}(m, \mathsf{pk}, \mathsf{plus}(r, r'))$. Unfortunately, this stuck ProVerif on a loop when computing the proof files. So instead, following an idea suggested on a private communication by Vincent Cheval, we defined the randomization with the following equations:

$$\begin{aligned}
&\forall\, m : \text{bitstring}, pk_a : \text{pkey}, k_s : \text{sskey}, r, r_1, r_2, s, s_1, s_2 : \text{rand};\\
&\quad \mathsf{rand_p_aenc_p}\left(\begin{array}{l}\mathsf{aenc_p}(m, pk_a, \mathsf{spk}(k_s), r, r_1),\\ \mathsf{sign_p}\big(\mathsf{aenc_p}(m, pk_a, \mathsf{spk}(k_s), r, r_1), pk_a, k_s, s, s_1\big),\\ pk_a, \mathsf{spk}(k_s), r_2, s_2\end{array}\right)\\
&= \quad \mathsf{aenc_p}(m, pk_a, \mathsf{spk}(k_s), r, r_2)
\end{aligned}$$

$$\begin{aligned}
&\forall\, m : \text{bitstring}, pk_a : \text{pkey}, k_s : \text{sskey}, r, r_1, r_2, s, s_1, s_2 : \text{rand};\\
&\quad \mathsf{rand_p_sign_p}\left(\begin{array}{l}\mathsf{aenc_p}(m, pk_a, \mathsf{spk}(k_s), r, r_1),\\ \mathsf{sign_p}(\mathsf{aenc_p}(m, pk_a, \mathsf{spk}(k_s), r, r_1), pk_a, k_s, s, s_1),\\ pk_a, \mathsf{spk}(k_s), r_2, s_2\end{array}\right)\\
&= \quad \mathsf{sign_p}\big(\mathsf{aenc_p}(m, pk_a, \mathsf{spk}(k_s), r, r_2), pk_a, k_s, s, s_2\big)
\end{aligned}$$

$$\begin{aligned}
&\forall\, c_b : \text{bitstring}, s_b : \text{bitstring}, pk_a : \text{pkey}, spk_s : \text{sskey}, r, s : \text{rand};\\
&\quad \mathsf{rand_p}\big(c_b, s_b, pk_a, spk_s, r, s\big)\\
&= \quad \big(\mathsf{rand_p_aenc_p}(c_b, s_b, pk_a, spk_s, r, s), \mathsf{rand_p_sign_p}(c_b, s_b, pk_a, spk_s, r, s)\big)
\end{aligned}$$

The two first equations model the randomizable asymmetric encryption and signature. Both of them take the same kind of inputs:

- A ciphertext, the randomized encryption of a bitstring m with nonces r and r_1: $\mathsf{aenc_p}(m, pk_a, \mathsf{spk}(k_s), r, r_1)$.
- The randomized signature of the previous ciphertext with nonces s and s_1: $\mathsf{sign_p}\big(\mathsf{aenc_p}(m, pk_a, \mathsf{spk}(k_s), r, r_1), pk_a, k_s, s, s_1\big)$.
- An asymmetric public key pk_a.
- A signature verification key $\mathsf{spk}(k_s)$.
- Two nonces r_2 and s_2 that are used to randomize the encryption and the signature.

With this equational theory, the randomization happens by replacing the right nonces (r_1 and s_1) by the new ones (r_2 and s_2). The left nonces (r and s) are never replaced. Their presence is necessary to model our randomizable cryptographic system: if we only had ciphertexts of the form $\mathsf{aenc_p}(m, pk_a, \mathsf{spk}(k_s), r_1)$, then it is possible to create a collision. Let us assume that a voter computed the ballot encrypting the vote "0" for instance: $\mathsf{aenc_p}(0, pk_a, \mathsf{spk}(k_s), r_1)$. Then an attacker could randomize it with a known nonce r_a, thus obtaining the term $\mathsf{aenc_p}(0, pk_a, \mathsf{spk}(k_s), r_a)$. They could then compare it with the terms $\mathsf{aenc_p}(0, pk_a, \mathsf{spk}(k_s), r_a)$ and $\mathsf{aenc_p}(1, pk_a, \mathsf{spk}(k_s), r_a)$, that they can compute from public values, and guess the voter's vote.

Communication channels

Three kinds of communication channels are used in our protocol.

- **Public channel:** the public channel is under the attacker control. We set it as a public name "public" of type channel.
- **Secure channels:** because of performances issues, we could not use the private channels ProVerif offers. Thus we modeled exchanges over a secure channel as the exchanges of ciphertexts encrypted with symmetric private keys shared by each entities. For the reader's convenience, we still chose to describe our processes here with the private channel syntax, so know that whenever an I/O process of the form in/out (private_channel, m) appears in our description, it is actually modeled as: in/out (public, $\{m\}_k$) with k a private key shared between the two parties and $\{m\}_k$ the encryption of the message m with this key. Whenever the channel is corrupted, the key is simply output.
- **Authenticated channels:** the channel between the voting device and the voting server is an authenticated channel the attacker can eavesdrop on. We modeled it as a secure channel so an attacker cannot modify the information transmitted on the channel but we output said information on the public channel. Since we model our channels as mentioned previously, messages can be dropped by the attacker - something that cannot be done by using classical private channels in ProVerif.

Election setup

Since we assume the election administrator to be a honest entity, we did not model per say the whole election setup process and we also made some few approximations.

Regarding the *valid voting options list*, because of performance issues, we modeled it as a set of two elements: $\mathcal{V} = \{0, 1\}$ with "0" and "1" of type vote. This is a limitation due to our protocol and the way to model it. Since we rely on a voting sheet, we could not offer the possibility to compute an arbitrarily long list of voting options. However, the proof files can easily be adapted to prove the security of our protocol for an arbitrarily finite set of voting options.

Credentials specific to each eligible voter are created during the election setup: logins and passwords, and signature verification keys. To manage them, we need a way to create an arbitrary long list of eligible voters (VL), a list of authentication credentials - linked to a specific voter and a public list of valid signature verification keys ($\mathcal{C}$) - not linked to a specific voter. Although ProVerif allows the use of tables (see section 1.1.4), it can cause some performance issues. Moreover, an attacker cannot read a table in the ProVerif calculus, which makes them inefficient to model a list of public values. so instead, we relied on functions to create and manage those values.

Eligible voters are created from a public name with a private function: $\mathsf{eligible\_voter}(n)$: name with n of type name. To check if a voter is indeed eligible, we defined the function is_eligible with the equation:

$$\forall\, n : \mathsf{name}\ ;\ \mathsf{is\_eligible}(\mathsf{eligible\_voter}(n)) = \mathsf{true}$$

The term $\mathsf{eligible\_voter}(n)$ is also used as the login of the user for the server.

Passwords are also created from a name with the private function: $\mathsf{password}(n)$: bitstring with n of type name. Note that this function allows the creation of passwords for an arbitrary name, this will for example allow a voting server to create authentication credentials for uneligible voters. Thus, valid authentication credentials will be of the form:

$$(\mathsf{eligible\_voter}(n), \mathsf{password}(\mathsf{eligible\_voter}(n)))$$

Finally, the *signing public key* are created from signing private keys of type sskey with the function spk described earlier. In our model, whenever a public key is sent or used, it comes along a value that acts as a certificate. This value is computed from the private function: $\mathsf{cert}(s)$: bitstring with s of type spkey. The use of this private function is obviously restricted to the registrar only. To check the validity of a signing public key, we rely on the function is_valid defined by the equation:

$$\forall\, s : \text{spkey} ;\ \mathsf{is\_valid}(s, \mathsf{cert}(s)) = \mathsf{true}$$

For the sake of readability, we omit the certificates in the following model description. Whenever a signing public key s is transmitted, note that it implicitly comes with its certificate value $\mathsf{cert}(s)$. Also, from now on, the validity checking will be shortened to $\mathsf{is\_valid}(s)$ instead of $\mathsf{is\_valid}(s, \mathsf{cert}(s))$.

Registrar

The registrar is a corruptible entity in charge of publishing the valid public credentials list ($\mathcal{C}$) and generating the voting sheet for each voter. It is modeled as a replicable process executed in parallel to all other processes.

- **Honest registrar:** we modeled the registrar with the process;

$$\begin{array}{l}
\mathsf{let}\ \mathrm{Honest_Registrar}(\mathrm{registrar_channel}) = \\
\quad \mathsf{new}\ \ ssk : \text{sskey}; \\
\\
\quad \mathsf{new}\ \ r_0, t_0 : \text{rand}; \\
\quad \mathsf{let}\ \ c_0 = \mathsf{aenc_p}(0, \mathsf{pk_e}, \mathsf{spk}(ssk), r_0, r_0)\ \mathsf{in} \\
\quad \mathsf{let}\ \ s_0 = \mathsf{sign_p}(c_0, \mathsf{pk_e}, \mathsf{spk}(ssk), t_0, t_0)\ \mathsf{in} \\
\quad \mathsf{let}\ \ \mathsf{voting_entry}_0 = \big(0, r_0, c_0, s_0\big)\ \mathsf{in} \\
\\
\quad \mathsf{new}\ \ r_1, t_1 : \text{rand}; \\
\quad \mathsf{let}\ \ c_1 = \mathsf{aenc_p}(1, \mathsf{pk_e}, \mathsf{spk}(ssk), r_1, r_1)\ \mathsf{in} \\
\quad \mathsf{let}\ \ s_1 = \mathsf{sign_p}(c_1, \mathsf{pk_e}, \mathsf{spk}(ssk), t_1, t_1)\ \mathsf{in} \\
\quad \mathsf{let}\ \ \mathsf{voting_entry}_1 = \big(1, r_1, c_1, s_1\big)\ \mathsf{in} \\
\\
\quad \mathsf{out}\ \ (\mathrm{registrar_channel}, (\mathsf{spk}(ssk), \mathsf{voting_entry}_0, \mathsf{voting_entry}_1)); \\
\quad \mathsf{out}\ \ (\mathrm{public}, (\mathsf{spk}(ssk))).
\end{array}$$

  The process depends on the private channel $\mathrm{registrar\_channel}$ shared with an arbitrary voter. The voting sheet is sent over it as the message $(\mathsf{voting\_entry}_0, \mathsf{voting\_entry}_1)$. As specified, it contains the nonces (r_0 and r_1) used to encrypt the votes, the encrypted votes (c_0 and c_1) along with their signatures (s_0 and s_1) and the signature verification key $\mathsf{spk}(ssk)$.

  At the end of the process, the signing verification key $\mathsf{spk}(ssk)$ of the voter is output on the public channel. Note that we modeled the fact that this key is part of a valid credential list because it will pass the is_valid test previously mentioned.

- **Rogue registrar:** the rogue registrar is controlled by the attacker who then has control over the private channel $\mathrm{registrar\_channel}$.

Voter

The voter processes - honest and rogue - are executed in parallel to all other processes.

- **Honest voter who audits the voting sheet:** we modeled honest voters with two processes V_0 and V_1 depending on their vote choice ("0" or "1"), we here give the model for a voter voting for "0" who audits their voting sheet:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{let}\ \mathrm{V}_0(ID, \mathrm{registrar\_channel}, \mathrm{audit\_channel}, \mathrm{voting\_device\_channel}) =} \\
\quad \mathsf{in} & \big(\mathrm{registrar\_channel},\ (spk, \mathsf{voting\_entry}_0, \mathsf{voting\_entry}_1)\big); \\
\quad \mathsf{event} & \mathsf{VOTER}(ID, spk, H); \\
 & \\
\quad \mathsf{let} & \big(v_0 : \mathrm{vote}, r_0 : \mathrm{rand}, c_0 : \mathrm{bitstring}, s_0 : \mathrm{bitstring}\big) = \mathsf{voting\_entry}_0\ \mathsf{in} \\
\quad \mathsf{let} & \mathsf{audit\_material}_0 = (v_0, r_0, c_0)\ \mathsf{in} \\
\quad \mathsf{let} & \mathsf{voting\_material}_0 = (spk, c_0, s_0)\ \mathsf{in} \\
 & \\
\quad \mathsf{let} & \big(v_1 : \mathrm{vote}, r_1 : \mathrm{rand}, c_1 : \mathrm{bitstring}, s_1 : \mathrm{bitstring}\big) = \mathsf{voting\_entry}_1\ \mathsf{in} \\
\quad \mathsf{let} & \mathsf{audit\_material}_1 = (v_1, r_1, c_1)\ \mathsf{in} \\
\quad \mathsf{let} & \mathsf{voting\_material}_1 = (spk, c_1, s_1)\ \mathsf{in} \\
 & \\
\quad \mathsf{out} & \big(\mathrm{audit\_channel},\ (spk, \mathsf{audit\_material}_0, \mathsf{audit\_material}_1)\big); \qquad (1) \\
\quad \mathsf{in} & \big(\mathrm{audit\_channel},\ (= spk, = 0, = 1, = \mathsf{true})\big); \qquad (2) \\
 & \\
\quad \mathsf{new} & n : \mathrm{rand}; \\
\quad \mathsf{event} & \mathsf{VOTE}(ID, 0, n); \\
\quad \mathsf{out} & \big(\mathrm{voting\_device\_channel},\ (ID, pwd, \mathsf{voting\_material}_0)\big); \\
 & \\
\quad \mathsf{get} & \mathsf{P}_{\mathsf{BB}}(= spk, b : \mathrm{bitstring}); \\
\quad \mathsf{event} & \mathsf{VERIFIED}(ID, 0).
\end{array}
$$

  The process depends on the voter's identifier (ID) and three channels ($\mathrm{registrar\_channel}$, $\mathrm{audit\_channel}$ and $\mathrm{voting\_device\_channel}$). After receiving the voting sheet from the $\mathrm{registrar\_channel}$, the event $\mathsf{VOTER}(ID, spk, H)$ is triggered for the voter received their voting credential (spk).

  Then the voter audits their voting sheet. If the audit is successful, they process to the vote, triggering the event $\mathsf{VOTE}(ID, v, n)$ (with $v \in \mathcal{V}$), by sending the voting material matching their choice through the $\mathrm{voting\_device\_channel}$ along their authentication credentials (ID and pwd). Note that our event "VOTE" depends on three parameters: the voter's identifier (ID), their vote (v) and a nonce (n). This had to be done because ProVerif does not handle well our revote restriction even if we modeled it, so we tagged each vote with a unique nonce. We will further explain why we used this tip in section 4.1.2.

  Finally, the voter can process to the vote verification by reading the entries on the public bulletin board (the table P_{BB}) and finding their public key (spk). The event $\mathsf{VERIFIED}(ID, 0)$ is then triggered.

- **Honest voter who does not audit the voting sheet:** we modeled this kind of voter by the exact same process as the previous one, except for the part where the voting sheet audit is done (lines tagged (1) and (2) in blue) that we omitted.

- **Rogue voter:** to specify verifiability, we need to distinguish a honest voter from a rogue one. a

rogue voter ID simply sends all their material to the attacker:

$$\begin{array}{ll}\text{let } \mathrm{V}_{\mathsf{C}}(ID : \text{name}, \text{registrar\_channel} : \text{channel}) = \\ \quad \text{in} & \big(\text{registrar\_channel},\ (spk, \mathsf{voting\_entry}_0, \mathsf{voting\_entry}_1)\big); \\ \quad \text{event} & \mathsf{VOTER}(ID, spk, C); \\ \quad \text{out} & \big(\text{public},\ (ID, pwd, spk, \mathsf{voting\_entry}_0, \mathsf{voting\_entry}_1)\big);\end{array}$$

They receive their voting sheet, trigger the event $\mathsf{VOTER}(ID, spk, C)$ that declares them as rogue voters and output everything to the attacker, including their authentication credentials.

Auditing device

The auditing device is a corruptible entity modeled as a process executed in parallel to all other processes. It is used by the voter to audit their voting sheet.

- **Honest auditing device:** we modeled the audit process as follows:

$$\begin{array}{ll}\text{let Honest\_Auditing\_Device}(\text{audit\_channel}) = \\ \quad \text{in} & \big(\text{audit\_channel},\ (spk, v_0, r_0, c_0, v_1, r_1, c_1)\big); \\ \quad \text{if} & \mathsf{aenc_p}(0, \mathsf{pk_e}, spk, r_0, r_0) = c_0 \ \wedge\ \mathsf{aenc_p}(1, \mathsf{pk_e}, spk, r_1, r_1) = c_1 \\ \quad \text{then} & \text{out}\ \big(\text{audit\_channel},\ (spk, v_0, v_1, \mathsf{true})\big).\end{array}$$

  After receiving the audit material from the audit_channel, the audit device processes to the verification described in section 2.2.3 - it verifies the encrypted votes do indeed encrypt the right votes. If the verification is successful, it gives the approval to the voter.

- **Rogue auditing device:** a rogue auditing device outputs everything it gets from the audit_channel as well as the private channel audit_channel itself to the attacker.

Voting device

The voting device is a corruptible entity used by the voter to cast a ballot. It is modeled as a process executed in parallel to all other processes.

- **Honest voting device:** the honest voting device process is described as follows:

$$\begin{array}{ll}\text{let Honest\_Voting\_Device}(\text{voting\_device\_channel}, \text{voting\_server\_channel}) = \\ \quad \text{in} & \big(\text{voting\_device\_channel},\ (ID, pwd, (spk, c_b, s_b))\big); \\ \quad \text{if} & \mathsf{verify_p}(c_b, s_b, spk) = \mathsf{true} \\ \quad \text{then} & \text{new } r, s : \text{rand}; \\ & \text{out}\ \big(\text{voting\_server\_channel},\ ((ID, pwd), (spk, \mathsf{rand_p}(\mathsf{pk_e}, spk, c_b, s_b, r, s)))\big) \\ & \text{out}\ \big(\text{public},\ (spk, \mathsf{rand_p}(\mathsf{pk_e}, spk, c_b, s_b, r, s))\big).\end{array}$$

  It verifies the signature of the ballot it received and if valid, it sends the authentication credentials as well as the randomized ballot through the authenticated channel $\text{voting\_server\_channel}$. Since we consider this channel to be authenticated with the login and password of the voter but not secure, the randomized ballot is output to be the public.

  Remember that our private channels are modeled with symmetric encryption, so the fact that the message is output first on the private channel is not a limitation for the attacker, they can still drop it *and* receive the plaintext from the last line of this process.

- **Rogue voting device:** a rogue voting device outputs everything it receives over the $\text{voting\_device\_channel}$ and $\text{voting\_server\_channel}$ as well as those channels to the attacker.

Voting server

The voting server process is a replicable process executed in parallel to all other processes. It can interact with anyone through an authenticated channel and publishes valid ballots on the public bulletin board.

- **Honest voting server:** the honest voting server process is described hereby:

$$
\begin{array}{l}
\mathsf{let}\ \mathrm{Honest\_Voting\_Server}(\mathrm{voting\_server\_channel}) = \\
\quad \mathsf{in} \quad \big(\mathrm{voting\_server\_channel},\ ((ID, pwd), (spk, c_b, s_b))\big); \\
\quad \mathsf{if} \quad\ \ \mathsf{is\_eligible}(ID) = \mathsf{true} \\
\qquad\quad \wedge\, pwd = \mathsf{password}(ID) \\
\qquad\quad \wedge\, \mathsf{is\_valid}(spk) = \mathsf{true} \\
\qquad\quad \wedge\, \mathsf{verify_p}(c_b, s_b, spk) = \mathsf{true} \\
\quad \mathsf{then} \quad \mathsf{new} \quad\ r, s : \mathsf{rand}; \\
\qquad\quad \mathsf{let} \quad\ (c_{b'}, s_{b'}) = \mathsf{rand_p}(\mathsf{pk_e}, spk, c_b, s_b, r, s)\ \mathsf{in} \\
\qquad\quad \mathsf{event}\ \ \mathrm{GOING\_TO\_TALLY}(ID, spk, c_{b'}) \\
\qquad\quad \mathsf{insert}\ \ \mathsf{P_{BB}}(spk, c_{b'}, s_{b'}) \\
\qquad\quad \mathsf{out} \quad\ \big(\mathrm{public},\ (spk, c_{b'}, s_{b'})\big).
\end{array}
$$

  After receiving a ballot, it proceeds all verification specified in section 2.2.3. If successful, the ballot is added to the bulletin board after being randomized. Note that the voting server interacts with any voter - rogue or honest - through a dedicated voting_server_channel.

- **Rogue voting server:** the rogue voting server is under the attacker control and outputs everything it receives from the voting_server_channel. However, since the attacker has no control over table processes in the ProVerif calculus, we still needed to model it.

$$
\begin{array}{l}
\mathsf{let}\ \mathrm{Rogue\_Voting\_Server}(\mathrm{voting\_server\_channel}) = \\
\quad \mathsf{in} \quad \big(\mathrm{voting\_server\_channel},\ ((ID, pwd), (spk, c_b, s_b))\big); \\
\quad \mathsf{if} \quad\ \ \mathsf{is\_valid}(spk) = \mathsf{true} \\
\qquad\quad \wedge\, \mathsf{verify_p}(c_b, s_b, spk) = \mathsf{true} \\
\quad \mathsf{then} \quad \mathsf{event}\ \ \mathrm{GOING\_TO\_TALLY}(ID, spk, c_b); \\
\qquad\quad \mathsf{insert}\ \ \mathsf{P_{BB}}(spk, c_{b'}, s_{b'}) \\
\qquad\quad \mathsf{out} \quad\ \big(\mathrm{public},\ (spk, c_{b'}, s_{b'})\big).
\end{array}
$$

  The rogue voting server process is basically the same as the honest one, yet, it does not proceed to any authentication credentials verification nor does it randomize a ballot it adds on the bulletin board - processes that are in blue in the previous model. Yet, a rogue voting process still verifies the credential is a valid one generated by the registrar and that the ballot signature is valid. We chose to do so because if both the registrar and the voting server are rogue, the attack on verifiability is pretty obvious, so this is a corruption scenario we were not interested in. In the case of a honest registrar, those verification about the voting credential are publicly done during the bulletin board audit, which is why we imposed the inserted ballots to be compliant with those criteria in our model of a rogue voting server.

Leaked information

We also needed to model 3 kinds of leaked information.

- **Authentication credentials:** authentication credentials may be leaked. We model this by adding the subprocess:

$$\mathsf{out}\ \big(\mathrm{public}, (\mathsf{eligible\_voter}(n), \mathsf{password}(\mathsf{eligible\_voter}(n)))\big)$$

in parallel with our main process.

- **Voting sheet:** the voting sheet might also be leaked. We model this by adding the line:

$$\mathsf{out}\left(\mathrm{public}, (\mathsf{voting\_entry}_0, \mathsf{voting\_entry}_1)\right)$$

at the end of our $\mathrm{Honest\_Registrar}$ process. Note that when considering a case where the registrar is rogue, there is no need to consider the case when the voting sheet is leaked.

- **Election key:** finally, we output the election secret key directly by adding the following subprocess in parallel to our main process:

$$\mathsf{out}\left(\mathrm{public}, \mathsf{sk_e}\right)$$

Audit and Tally

Since our proofs of the verifiability theorems do not rely on the tallying and election audit processes - except for the hypothesis of a good audit performed in the case of a honest registrar - we let them under the adversary control by giving them the election private key.

Main process

The whole election process is given by the main process described in Fig.4.1.

4.1.2 Formal properties in the ProVerif calculus

We hereby explain why our voting scheme fulfill the assumptions of our theorems and then provide the formal properties used to prove the verifiability of our schemes on ProVerif.

Compliance of our protocol to the theorems hypothesis

As stated in sections 3.1, 3.2 and 3.3, our verifiability theorems depends on the compliance of a voting scheme to a specific set of hypothesis. We will explain here why our voting protocol fits the scope of those theorems.

- *Hypothesis:*
  - *Honest voter behaviour* (HV): revoting is forbidden in our voting scheme. A honest voter votes only once, which satisfies our condition (1). A voter verifies their vote only if they effectively previously voted, which satisfies condition (2).
  - *Uniqueness of a ballot on the public bulletin board* (UoB-PBB): each ballot displays the signing public key as the first term on a ballot. Plus, the randomization of each ballot guarantees that ciphertexts are all different from one another.
  - *Partial tallying of the election counting function* (PT): this hypothesis is satisfied by our decryption scheme.
  - *Honest registrar behaviour* (HR): a honest registrar will produce one voting sheet per eligible voter (3) and no more (2). Moreover the credentials used to generate the voting sheet are all different from one another (1).
- *Trace properties we had to admit:* We had to admit some trace properties for our security proofs. they could be expressed, but ProVerif could not terminate them when computing the proof files. However, those properties are verified by our protocol.

```
process (   out   (public, (pk_e, sk_e))
        |!  new   (n : name);
            out   (public, (eligible_voter(n), password(eligible_voter(n))))                    (1)
        |!  in    (public, n : name);
            new   (registrar_channel : channel);
            new   (voting_device_channel : channel);
            new   (voting_server_channel : channel);
            new   (auditing_device_channel : channel);                                  (2.1) (2.2)
            !(    Honest_Registrar(registrar_channel)                                           (3.1)
            !(    Rogue_Registrar(registrar_channel)                                            (3.2)
             |(   V_0(n, registrar_channel, auditing_device_channel, voting_device_channel)
              |   Honest_Auditing_Device(auditing_device_channel))(2.1)
              |   Rogue_Auditing_Device(auditing_device_channel))(2.2)
              |   Honest_Voting_Device(voting_device_channel, voting_server_channel)             (4.1)
              |   Rogue_Voting_Device(voting_device_channel, voting_server_channel)             (4.2)
              |   Honest_Voting_Server(voting_server_channel)                                   (5.1)
              |   Rogue_Voting_Server(voting_server_channel) )                                  (5.2)
        |!  in    (public, n : name);
            new   (registrar_channel : channel);
            new   (voting_device_channel : channel);
            new   (voting_server_channel : channel);
            new   (auditing_device_channel : channel);                                  (2.1) (2.2)
            (     Honest_Registrar(registrar_channel)                                           (3.1)
            (     Rogue_Registrar(registrar_channel)                                            (3.2)
             |(   V_1(n, registrar_channel, auditing_device_channel, voting_device_channel)
              |   Honest_Auditing_Device(auditing_device_channel))(2.1)
              |   Rogue_Auditing_Device(auditing_device_channel))(2.2)
              |   Honest_Voting_Device(voting_device_channel, voting_server_channel)             (4.1)
              |   Rogue_Voting_Device(voting_device_channel, voting_server_channel)             (4.2)
              |   Honest_Voting_Server(voting_server_channel)                                   (5.1)
              |   Rogue_Voting_Server(voting_server_channel) )                                  (5.2)
        |!  in    (public, voting_server_channel : channel);
                  Honest_Voting_Server(voting_server_channel)                                    (5.1)
                  Rogue_Voting_Server(voting_server_channel)                                     (5.2)
        |!  in    (public, n : name);
            new   (registrar_channel : channel);
            (     Honest_Registrar(registrar_channel)                                           (3.1)
            (     Rogue_Registrar(registrar_channel)                                            (3.2)
                  Rogue_Voter(registrar_channel) ) )
```

(1) : if authentication credentials leaked
(2.1) : if voter audits voting sheet with honest device
(2.2) : if voter audits voting sheet with rogue device
(3.1) : if honest registrar
(3.2) : if rogue registrar
(4.1) : if honest voting device
(4.2) : if rogue voting device
(5.1) : if honest voting server
(5.2) : if rogue voting server

Figure 4.1: Our ProVerif main process for the verifiability proof of Belenios VS

- *Proper voter list* (PVL): a honest voting server checks that a voter voted once and that its credential was not previously used which is in accordance to this hypothesis.

- *Honest voting server behaviour* (HS): the same argument can be used for this hypothesis.

- *Public bulletin board verification* (PBB-Verif): the public credential (spk) is publicly displayed on the bulletin board (1). Plus, they are all different from one another (2).

Honest voting server

Given our *Verifiability based on ID* theorem, we need to prove properties VC-ID (Valid Credential), CAI-ID(Cast as Intended) and TAC-ID (Tallied as Cast) in order to prove the verifiability of our protocol in the context of a honest voting server.

ProVerif does not directly handle properties of type $(A \wedge B) \implies C$[13]. Hence, instead of proving separately VC-ID and CAI-ID, we combined them into one equivalent property. First, note that VC-ID-(1) is automatically verified since ProVerif triggers the event GOING_TO_TALLY for all its three parameters (see the previous section 4.1.1 for the ProVerif model). Also, note that since by definition, $\mathcal{N} = \mathsf{HV} \sqcup \mathsf{C}$, VC-ID-(2) is equivalent to:

$$\begin{array}{l}\forall\, \mathcal{T} \in Trace(\mathcal{P}) \\ \forall\, b \in \mathcal{B},\ \forall\, ID \in \mathcal{N}. \\ \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T} \implies (\mathsf{VOTER}(ID, *, H) \in \mathcal{T} \vee \mathsf{VOTER}(ID, *, C) \in \mathcal{T})\end{array}$$

Finally, if both VC-ID and CAI-ID are satisfied, it is equivalent to the property:

$$\begin{array}{l}\forall\, \mathcal{T} \in Trace(\mathcal{P}) \\ \forall\, b \in \mathcal{B},\ \forall\, ID \in \mathcal{N}. \\ \mathsf{GOING\_TO\_TALLY}(ID, *, b) \in \mathcal{T} \implies \left(\begin{array}{ll} & \mathsf{VOTER}(ID, *, H) \in \mathcal{T} \implies \left(\begin{array}{l} \mathsf{VOTE}(ID, v) \\ unwrap(b) = v \end{array} \right) \\ \vee & \mathsf{VOTER}(ID, *, C) \in \mathcal{T} \end{array} \right)\end{array}$$

We translated this in the ProVerif calculus into the property:

$$\begin{array}{l}\forall\, ID : \text{bitstring}, v : \text{vote}, b : \text{bitstring}; \\ \mathsf{event}\ \mathsf{GOING\_TO\_TALLY}(ID, *, b) \\ \qquad\qquad \mathsf{event}\ \ \mathsf{VOTER}(ID, *, C) \qquad\qquad \text{(PV-VC-CAI-ID)} \\ \qquad \vee \\ \implies \quad \mathsf{event}\ \ \mathsf{VOTER}(ID, *, H) \\ \qquad\quad \wedge\ \mathsf{event}\ \ \mathsf{VOTE}(ID, v, *) \\ \qquad\quad \wedge\ v = \ \mathsf{adec_p}(b, \mathsf{sk_e})\end{array}$$

As for the Tallied as Cast (TAC-ID) property, since we cannot effectively prevent revote in a ProVerif

[13] See ProVerif manual, section 4.3 Basic correspondences p 43

model, we proved the trace property:

$$\begin{array}{l}\forall\, ID : \text{bitstring}, v : \text{vote}, b : \text{bitstring}, n_1, n_2 : \text{rand};\\ \\ \mathsf{event}\ \mathsf{VERIFIED}(ID, v)\\ \\ \qquad\qquad\quad \mathsf{event} \quad \mathsf{GOING\_TO\_TALLY}(ID, *, b)\\ \qquad\qquad \wedge \quad v \quad = \mathsf{adec_p}(b, \mathsf{sk_e})\\ \qquad \vee\\ \Longrightarrow\\ \qquad\qquad\quad \mathsf{event} \quad \mathsf{VOTE}(ID, *, n_1)\\ \qquad\qquad \wedge \quad \mathsf{event} \quad \mathsf{VOTE}(ID, *, n_2)\\ \qquad\qquad \wedge \quad n_1 \neq n_2\end{array} \qquad \text{(PV-TAC-ID)}$$

It states that either TAC-ID is verified, either a honest voter revoted, which is an impossible case scenario in our protocol.

Honest registrar

We also needed to express the security properties VC-cred (Valid Credential), CAI-cred(Cast as Intended) and TAC-cred (Tallied as Cast) in the ProVerif calculus so that, in the context of a honest registrar, the three of them imply the verifiability of our protocol thanks to our *Verifiability based on credential* theorem.

Similarly to the context of a honest server, we combined VC-cred and CAI-cred into one equivalent property. VC-cred-(1) is automatically satisfied in our ProVerif models and thanks to hypothesis HR-(3), VC-cred-(2) is equivalent to:

$$\begin{array}{l}\forall\, \mathcal{T} \in Trace(\mathcal{P})\\ \forall\, b \in \mathcal{B},\ \forall\, cred \in \mathcal{C}.\\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) \in \mathcal{T} \implies \exists\, ID \in \mathcal{N}. \left(\begin{array}{ll} & \mathsf{VOTER}(ID, cred, H) \in \mathcal{T}\\ \vee & \mathsf{VOTER}(ID, cred, C) \in \mathcal{T} \end{array} \right)\end{array}$$

A property which if satisfied as well as CAI-cred is equivalent to:

$$\begin{array}{l}\forall\, \mathcal{T} \in Trace(\mathcal{P})\\ \forall\, b \in \mathcal{B},\ \forall\, ID \in \mathcal{N}.\\ \mathsf{GOING\_TO\_TALLY}(*, cred, b) \in \mathcal{T} \implies \exists\, ID \in \mathcal{N}. \left(\begin{array}{ll} & \mathsf{VOTER}(ID, cred, H) \in \mathcal{T}\\ & \quad \implies \mathsf{VOTE}(ID, v) \wedge unwrap(b) = v\\ \vee & \mathsf{VOTER}(ID, cred, C) \in \mathcal{T} \end{array} \right)\end{array}$$

In the ProVerif calculus, this property becomes:

$$\begin{array}{l}\forall\, ID : \text{bitstring}, v : \text{vote}, b : \text{bitstring};\\ \\ \mathsf{event}\ \mathsf{GOING\_TO\_TALLY}(*, cred, b)\\ \\ \qquad\qquad\quad \mathsf{event} \quad \mathsf{VOTER}(*, cred, C)\\ \qquad \vee\\ \Longrightarrow\\ \qquad\qquad\quad \mathsf{event} \quad \mathsf{VOTER}(ID, cred, H)\\ \qquad\qquad \wedge \quad \mathsf{event} \quad \mathsf{VOTE}(ID, v, *)\\ \qquad\qquad \wedge \quad v = \quad \mathsf{adec_p}(b, \mathsf{sk_e})\end{array} \qquad \text{(PV-VC-CAI-cred)}$$

Like TAC-ID, we transformed TAC-cred into the security property:

$$\forall\, ID : \text{bitstring}, v : \text{vote}, b : \text{bitstring}, n_1, n_2 : \text{rand};$$

$$\mathsf{event}\ \mathsf{VERIFIED}(ID, v)$$

$$\begin{array}{rlll} & \mathsf{event} & \mathsf{GOING\_TO\_TALLY}(*, cred, b) & \\ \wedge & \mathsf{event} & \mathsf{VOTER}(ID, cred, H) & \text{(PV-TAC-cred)} \\ \wedge & v & = \mathsf{adec_p}(b, \mathsf{sk_e}) & \\ \Longrightarrow \quad \vee & & & \\ & \mathsf{event} & \mathsf{VOTE}(ID, *, n_1) & \\ \wedge & \mathsf{event} & \mathsf{VOTE}(ID, *, n_2) & \\ \wedge & n_1 \neq n_2 & & \end{array}$$

It states that either TAC-cred is verified, either a honest voter revoted, which is an impossible case scenario in our protocol.

4.1.3 Results

We present here our results of our security proofs.

Corruption cases we considered and analysis results

We generated ProVerif files for all plausible combinations of processes, some case scenarii were not considered:

- Since we loose verifiability for of our protocol as soon as both the registrar and the voting server are compromised, we considered corruption scenarii where *at least the registrar or the voting server is honest*.
- Whenever the registrar is corrupted, the voting sheet can be considered as leaked - for the attacker has all control over its generation. Hence, we considered corruption scenarii where *at least the registrar is honest or the voting sheet has not been leaked*.
- Finally, if the voter does not audit the voting sheet, there is no need to model the auditing device.

All in all, this result in a total of 60 different corruption cases summarized in Table 4.1.3. Performing all those proofs - trace properties that did not require a lot of computation time in total - allowed us to extensively study the limits of our protocol and the attacks against it as well as providing us an accurate picture of which maximal corruption cases it could stand against regarding verifiability.

Security of our protocol

As claimed in the previous chapter, our protocol is verifiable as soon as one of these conditions is satisfied:

- The registrar is honest *and* the voting sheet has not been leaked.

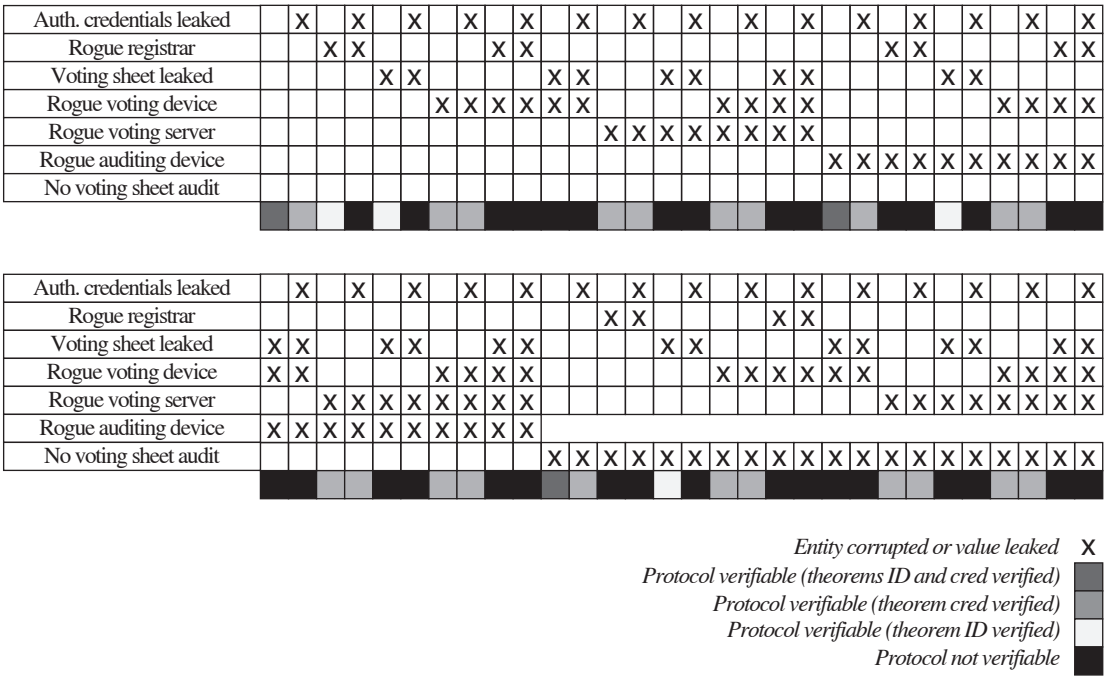


Table 4.1: Our protocol verifiability in different corruption cases *(in all cases, the election private key is leaked)*.

- The voting server *and* the voting device are honest *and* the authentication credentials have not been leaked *and* either the registrar is honest *or* the voter audits their voting sheet with a honest auditing device.

Whether the election secret key is leaked or not has *no* impact on our protocol verifiability.

We can see from Table 4.1.3 that the verifiability holds mostly thanks to the good use of voting credentials. Indeed, the *Cast as Intended* (CAI-ID) property quickly fails as soon as the voting server is rogue - it could easily register a ballot coming from a honest voter ID under another identifier. However, in those corruption cases, because we use public voting credentials, it would not matter if the voting server displayed a ballot registered for another voter than the original one, for it shall display a valid ballot regarding the election audit - with a valid signature done with a valid voting credential. This condition can be interpreted as less restrictive in some voting scheme because we proved that the verifiability could be implied by properties on the voting credential only.

Also note that there are some cases where both the auditing device and the voting device can be corrupted yet the verifiability still holds. This is an improvement on latest version of Belenios since now we can hope for achieving verifiability even though domestic devices are untrustworthy.

Vulnerabilities

In a nutshell, there are two kinds of attacks on our protocol, in both, the *Cast as Intended* property does not hold. Basically, an attacker is able to force the voter to vote for another vote.

In the first kind of attack, the attacker has both the authentication credential of the voter and access to their voting sheet, whether it has been leaked or the registrar is compromised. Getting the authen-

tication credentials happen whenever the voting server, the voting device are rogue or the credentials are stolen. On the other hand, compromising the voter's voting sheet happens whenever the registrar is under the attacker's control or the sheet has been leaked. Whenever the attacker gets both the voting and authentication credential, they can vote instead of the voter.

For instance, if the voting sheet is leaked and the attacker has the voter's authentication credential. Then the attacker can impersonate the honest voter by connection with the authentication credentials. The attacker can cast whatever vote they want using the voter's voting sheet. The vote will be validated by the voting server and published, thus compromising the *Cast as Intended* property.

The other kind of attacks happen whenever the registrar is rogue and either the voter does not audit the voting sheet or their auditing device is also compromised. In this case, the attacker has full control over the voting sheet and can, for instance, encrypt the same vote v in the whole voting sheet instead of the full set $\mathcal{V}$ and the voter would not detect it.

For instance, if the two voting options are "0" and "1", the attacker could randomly encrypt the same vote "0" for all entries in the voting sheet. If the voter does not audit the voting sheet or if their voting device is compromised, the voter could think they voted for candidate "1" and instead cast a ballot for "0".

Now that our protocol verifiability has been discussed, we are going to expose and prove the guarantees it provides regarding vote confidentiality.

4.2 Privacy of our protocol

Intuitively, if a protocol satisfies vote confidentiality, it means that an attacker should not learn how the voter voted. This section exposes how to formalize this property and express it in the ProVerif calculus. Since privacy is an equivalence property, some adaptations had to be made in our model to help the tool achieving the proofs. The results are discussed at the end of the section.

4.2.1 Formalizing the vote confidentiality

Vote confidentiality - which we will call privacy from now on - was formalized in [45] and [78] as an equivalence property and is the standard definition of privacy. The intuition behind it is that privacy is achieved as long as the attacker cannot distinguish the instance where Alice voted "0" and Bob "1" from the instance where Alice voted "1" and Bob "0". So if $\mathcal{P}_{\mathrm{V}}(ID, v)$ defines a protocol where voter ID votes for candidate v, we can say that it achieves verifiability if the following equivalence is satisfied:

$$!\big(\mathcal{P}_{\mathrm{V}}(A,0) \mid \mathcal{P}_{\mathrm{V}}(B,1)\big) \;\sim\; !\big(\mathcal{P}_{\mathrm{V}}(A,1) \mid \mathcal{P}_{\mathrm{V}}(B,0)\big)$$

Expressing privacy in the ProVerif calculus

This equivalence can be seen as the equivalence between two processes that only differ by their terms (ID and v). Thus, it is possible to express it in the ProVerif calculus as a diff-equivalence security property[14] by using biprocesses.

$$!\big(\; \mathcal{P}_{\mathrm{V}}(A, \mathsf{choice}[0,1]) \mid \mathcal{P}_{\mathrm{V}}(B, \mathsf{choice}[1,0]) \;\big)$$

[14] See section 1.3.2.

Proving privacy with only two voters

Proving diff-equivalence properties for an unbounded number of sessions can be quite a task for the ProVerif tool. Thankfully, [6] demonstrated that, in some conditions, it is enough to prove privacy for two honest voters and an unbounded number of dishonest voters. Since our protocol satisfies the conditions required by [6], we can rely on it.

In a nutshell, we need to prove the diff equivalence of our voting protocol for two honest voters who votes differently and an unbounded number of dishonest voters. In the ProVerif calculus, we could write this as the diff-equivalence property:

$$\mathcal{P} \mid \mathrm{V}(A, \mathsf{choice}[0,1]) \mid \mathrm{V}(B, \mathsf{choice}[1,0]) \mid !\mathrm{V}_{\mathsf{C}}$$

With:

- $\mathrm{V}(ID, v)$: the honest voter process that models voter ID who votes for v.
- V_{C}: the corrupted voter process.
- The other entities part of our protocol whose behaviour are not impacted.

4.2.2 ProVerif models

We based our ProVerif models for privacy on the models used for verifiability. we first tried to use the same proof files adapted to privacy but ProVerif could not terminate. Because of those performance issues, each corruption case we studied needed an individual further approximation an could not be generated automatically as we did it for our verifiability proofs. Because we had more knowledge about the limits of our protocol thanks to the verifiability study, we could target what appeared to be the *extreme corruption cases scenarii*: the maximal corruption cases our protocol could handle and the minimal corruption cases it would not regarding privacy. Those limit cases are:

- Privacy holds if the election private key is *not* leaked and:
  - The registrar is honest and the voting sheet is not leaked and the voter does not audit the voting sheet (1 proof file).
  - The registrar is honest and the voting sheet is not leaked and either the voter audits their voting sheet with a honest auditing device or the voting device is honest (2 proof files).
  - The authentication credentials are not leaked, the voting server and the voting device are both uncompromised and either the the voter audits their voting sheet with a honest auditing device or the registrar is honest (2 proof files).
- Privacy is not guaranteed if:
  - The election private key is leaked (1 proof file).
  - Both the authentication credentials are compromised - if they are leaked or if either the voting device or the voting server are under the attacker control - and the voting sheet is compromised - by being leaked or if the registrar is rogue. We only need to verify here that a leaked password and voting sheet are enough to compromise the privacy, since other corruption cases are stronger(1 proof file).
  - The registrar is rogue and either the voter does not audit the voting sheet or audits it with a rogue device (2 proof files).
  - Both the auditing and voting device are corrupted (1 proof file).

All ProVerif files can be found at [1].

"Chain of trust" model

The first main approximation was the blending of "neighbor" honest processes altogether. For instance, the honest voter interacts with the registrar and its devices. If, in the corruption case considered, those "neighbors" processes are honest, in order to prevent the heavy use of private channel, we chose to model all those honest processes under one same process.

Re-randomization dilemma

We used the same equational theory defined in section 4.1.1. Because we have seen that re-randomization as specified in our protocol description could not be handled by ProVerif - we randomize a ballot by "adding" a nonce on both the vote encryption and its signature - we had to make an approximation that produced no false attacks on verifiability. A randomized encrypted vote is modeled as a term of the following form:

$$\mathsf{aenc_p}(v, pk, spk, r_0, r_1)$$

With v the vote, pk the public election key, spk the voter's public signature verification key and r_0 and r_1 nonces. We randomize the encrypted vote by replacing the right nonce r_1 by another one r_2 and letting r_0 in place:

$$\mathsf{aenc_p}(v, pk, spk, r_0, r_2)$$

Same goes for the ciphertext signature.

This was not a problem to prove the verifiability of our protocol, however it produces false attack traces regarding privacy:

Let us assume that the voting sheet is leaked while all other entities are honest and that the attacker knows the identity of the voter who used it.

1. The attacker knows the voter's public verification key spk and the ciphertexts of the form $\mathsf{aenc_p}(v, pk, spk, r_0, r_1)$ with $v \in \{0, 1\}$ along the nonce r_1.

2. After the voter voted, their ballot appears on the public bulletin board: it has the form $\mathsf{aenc_p}(v', pk, spk, r_0, r)$ and comes along the voter's key spk.

3. The attacker spots and retrieves the voter's ballot $\mathsf{aenc_p}(v, pk, spk, r_0, r)$.

4. The attacker uses $\mathsf{rand_p\_aenc_p}$ from our equational theory with the nonce r_1. The output result is of the form $\mathsf{aenc_p}(v', pk, spk, r_0, r_1)$.

5. Since the attacker knows all values of the form $\mathsf{aenc_p}(v, pk, spk, r_0, r_1)$, they can compare it to the one they just computed and guess the voter's vote

This is indeed a false attack on privacy, for the ballot displayed on the bulletin board would here have been re-randomized twice - by the voting device and the voting server - which would make it unlinkable with the original ballot. With the randomizable encryption scheme we use, no collision of this sort can be computed.

The first solution we tried was to create a $\mathsf{rand_p\_left}$ function used only by honest devices and honest voting servers that would replace the left nonce with a fresh new one in addition to our $\mathsf{rand_p}$ function. Sadly, this solution would make ProVerif enter an infinite loop.

So, with a lot of precaution, whenever the voting sheet was compromised and the voting device was honest, we chose instead to insert the following lines in our voter's process to model the leak:

$$
\begin{array}{ll}
\mathsf{new} & r_0, t_0 : \text{rand}; \\
\mathsf{let} & c_0 = \mathsf{aenc_p}(0, \mathsf{pk_e}, \mathsf{spk}(ssk), r_0, r_0) \text{ in} \\
\mathsf{let} & s_0 = \mathsf{sign_p}(c_0, \mathsf{pk_e}, \mathsf{spk}(ssk), t_0, t_0) \text{ in} \\
\mathsf{let} & \mathsf{voting\_entry}_0 = \big(0, r_0, c_0, s_0\big) \text{ in} \\
 & \\
\mathsf{new} & r_1, t_1 : \text{rand}; \\
\mathsf{let} & c_1 = \mathsf{aenc_p}(1, \mathsf{pk_e}, \mathsf{spk}(ssk), r_1, r_1) \text{ in} \\
\mathsf{let} & s_1 = \mathsf{sign_p}(c_1, \mathsf{pk_e}, \mathsf{spk}(ssk), t_1, t_1) \text{ in} \\
\mathsf{let} & \mathsf{voting\_entry}_1 = \big(1, r_1, c_1, s_1\big) \text{ in} \\
 & \\
\mathsf{new} & r_v, t_v : \text{rand}; \\
\mathsf{let} & c_v = \mathsf{aenc_p}(v, \mathsf{pk_e}, \mathsf{spk}(ssk), r_v, r_v) \text{ in} \\
\mathsf{let} & s_v = \mathsf{sign_p}(c_v, \mathsf{pk_e}, \mathsf{spk}(ssk), t_v, t_v) \text{ in} \\
\mathsf{let} & \mathsf{voting\_entry}_v = \big(v, r_v, c_v, s_v\big) \text{ in} \\
 & \\
\mathsf{out} & \big(\text{public}, (\mathsf{spk}(ssk), \mathsf{voting\_entry}_0, \mathsf{voting\_entry}_1)\big); \\
\mathsf{out} & \big(\text{public}, (\mathsf{spk}(ssk), c_v, s_v)\big);
\end{array}
$$

Intuitively, we create three ballots: $\mathsf{voting\_entry}_0$ and $\mathsf{voting\_entry}_1$ which are displayed on the voting sheet and $\mathsf{voting\_entry}_v$ which represents the ballot that encrypts the voter's choice v randomized by a honest device. The voting entries (ballots and their root nonce) from the voting sheet are output whereas only the ballot is output - as specified - for the randomized ballot.

This solution has the advantage of effectively make a ballot unlinkable to the one it was randomized from, nonetheless, it shall be used carefully (only in the case of a voting sheet leak *and* when the voting device is honest).

This also means that we had to make a serious assumption regarding the registrar: we could not allow the rogue registrar process to input whichever nonces it wanted for vote generation *and* whichever signature private key, those were instead generated *inside* the voter model. This also meant that we assumed that there were *at least* two different set of voting credentials for the two honest users - of course, the signing secret keys were output. This approximation can seem as quite strong but reasonably, when the voting device is honest, we can hope for this unlinkability property between different instances of randomized ballots to hold. Also, the two different credentials, regarding our protocol specification - a credential for each voter - is not a so strong assumption. Plus, if the registrar is under the attacker's control, the voting server at least shall be honest otherwise privacy would fail for reasons we will expose further on. This falls under our assumption for a honest voting server behaviour assumption HS - that we did make for verifiability bu is yet still relevant - which imposes that a honest voting server would only register ballots coming from a pair (ID, spk) such that ID did not previously cast a vote and spk was never used to cast a vote.

Adapting the honest voter process

For the verifiability study, we wrote two processes for honest voters V_0 and V_1 for honest voters voting 0 and 1 respectively. We used the same kind of processes with a little twitch: process V_{01} (respectively V_{10}) where $\mathsf{choice}[0, 1]$ (respectively $\mathsf{choice}[1, 0]$) is hardcoded in the voter's process: "0" (respectively "1") is replaced by $\mathsf{choice}[0, 1]$ (respectively $\mathsf{choice}[1, 0]$) in V_0 (respectively V_1) to obtain V_{01} (respectively V_{10}).

To help the tally process, we need to link the voter to their signature verification key. Thus we output it on a private channel along the voter's identifier.

Modeling the tally process

Following an suggestion from Mathieu Turuani, we separated the tally process into two subprocesses, as it is usually done: one that tallied the ballots from honest voters (based on their voting credential) and the other one that tallied all other ballots except from the ones coming from honest voters. This distinctions between ballots coming from honest or rogue voters can be done because we output the honest voter's identifier along the signature key they used when casting their ballot.

Our tally process for the honest voters A and B is then modeled as follows:

$$
\begin{array}{l}
\mathsf{let}\ \mathrm{Tally\_Honest\_Voters}() = \\
\quad \mathsf{in} \quad \big(\mathrm{tally\_channel},\ (= A, spk_A : \mathrm{spkey})\big); \\
\quad \mathsf{in} \quad \big(\mathrm{tally\_channel},\ (= B, spk_B : \mathrm{spkey})\big); \\
\quad \mathsf{if} \quad spk_A \neq spk_B \\
\quad \mathsf{then} \quad \mathsf{get}\ \mathsf{P_{BB}}[= spk_A, b_A]; \\
\qquad\quad \mathsf{get}\ \mathsf{P_{BB}}[= spk_B, b_B]; \\
\qquad\quad \mathsf{new}\ \mathrm{mixnet} : \mathsf{channel}; \\
\qquad\quad (\mathsf{out}(\mathrm{mixnet}, \mathsf{choice}[b_A, b_B]) \mid \mathsf{out}(\mathrm{mixnet}, \mathsf{choice}[b_B, b_A])) \\
\qquad\quad \mid \mathsf{in}(\mathrm{mixnet}, b_1 : \mathrm{bitstring}); \\
\qquad\quad\ \ \mathsf{in}(\mathrm{mixnet}, b_2 : \mathrm{bitstring})) \\
\qquad\quad \mathsf{out}\ (\mathrm{public}, (\mathsf{adec_p}(b_1, \mathsf{sk_e}), adecp(b_2, \mathsf{sk_e})))
\end{array}
$$

Our generic tally process - for corrupted voters only - is then:

$$
\begin{array}{l}
\mathsf{let}\ \mathrm{Tally}(spk) = \\
\quad \mathsf{in} \quad \big(\mathrm{tally\_channel},\ (= A, spk_A : \mathrm{spkey})\big); \\
\quad \mathsf{in} \quad \big(\mathrm{tally\_channel},\ (= B, spk_B : \mathrm{spkey})\big); \\
\quad \mathsf{if} \quad spk_A \neq spk_B \ \wedge\ spk \neq spk_A \ \wedge\ spk \neq spk_B \\
\quad \mathsf{then} \quad \mathsf{get}\ \mathsf{P_{BB}}[= spk, b]; \\
\qquad\quad \mathsf{out}\ (\mathrm{public}, (\mathsf{adec_p}(b, \mathsf{sk_e})))
\end{array}
$$

4.2.3 Results

Results of our security proofs are displayed in Table 4.2.3 and are going to be discussed in this section.

Privacy of our protocol

The comparison of Tables 4.1.3 and 4.2.3 shows us that privacy is guaranteed for roughly the same case scenarii as verifiability *when the secret key has not been leaked*, with the exception of the case where both the auditing and voting devices are compromised - which, regarding verifiability, would not be a liability as long as the registrar is honest and the voting sheet has not been leaked.

To sum it up, privacy holds as long as:

- The election secret key is not leaked *and* the registrar is honest *and* the voting sheet is not leaked *and* the voter does not audit the voting sheet.
- The election secret key is not leaked *and* the registrar is honest *and* the voting sheet is not leaked *and* either the voter audits their voting sheet with a honest auditing device *or* the voting device is honest.

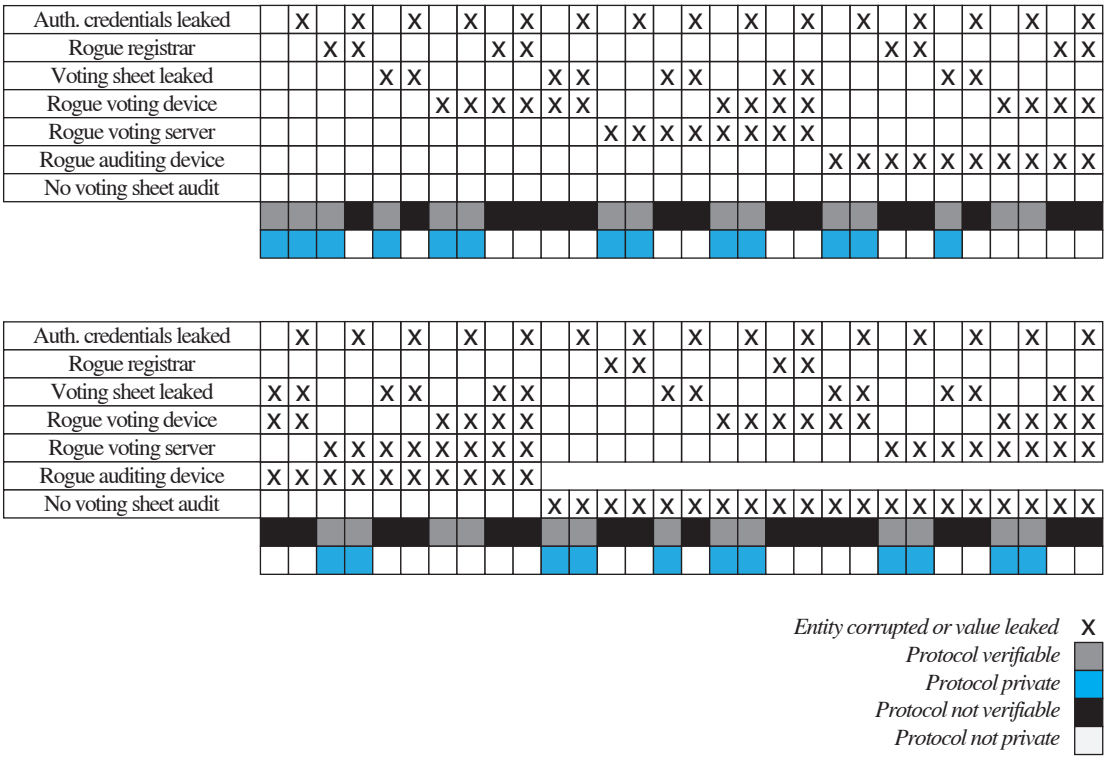


Table 4.2: Our protocol privacy in different corruption cases.

- The election secret key is not leaked *and* the authentication credentials are not leaked *and* the voting server *and* the voting device are honest *and* either the the voter audits their voting sheet with a honest auditing device *or* the registrar is honest.

About the attacks against privacy

We can mainly distinguish three kinds of attacks against privacy. Two of them are quite intuitive when the last one can be considered as a little more subtle.

If the election private key is leaked, we automatically loose privacy. This is quite understandable since the attacker can decrypt all ballots and the channel between the voting device and the voting server is authenticated but can be read by the attacker who can link a ballot to a voter.

If both the voting and auditing devices are compromised, the attacker knows the different ballots encryption from the auditing device and what the voter voted from the rogue auditing device - which would not output a randomized ballot to the attacker.

The last kind of attack is perhaps the least intuitive one. We can see it happen whenever the verifiability is compromised. Considering our two honest voters A and B and assuming the verifiability does not hold, the attacker forces A to vote for v which means that observing the tally for the honest voters it can find out what was B original vote.

Let us assume a case where verifiability is compromised, for instance when A's authentication credentials are lost and the registrar is compromised. A and B are our two honest voters and all other voters are under the attacker control, meaning the attacker knows what their votes are. For the sake of simplicity, we do not consider their vote in the final tally (because the attacker has them it can compare the tally output with all its knowledge). Let us assume that B already voted for candidate v with $v \in \{0, 1\}$.

1. Since the attacker has A's authentication credentials and voting sheet, it can cast a vote for A. Let us assume that it casts a ballot encrypting a vote for candidate "0".
2. The voting server accepts the attacker's ballot as A's. A is already linked to the signature verification key spk_A for out tally process. Thus the attacker's ballot is tallied as a honest voter's ballot with the process $\mathrm{Tally\_Honest\_Voters}$ described in our previous section.
3. The attacker deduces B's vote from the tally result:
   - If the result is two votes for candidate "0", then B voted "0".
   - If the result is one vote for each candidate "0" and "1", then B voted "1".

This is one example, but the attack is essentially the same whenever the *Cast as Intended* property fails.

Does privacy imply verifiability?

This last attack can raise some questions about whether *the verifiability of a protocol is necessary for its privacy*.

During our study of verifiability and specifically the attacks against the protocol, we found out that each time the verifiability was compromised, it was first because the *Cast as Intended* property did not hold. As a reminder, the Cast as Intended property states that if a ballot was registered for a honest voter and is about to be tallied - by being published on a bulletin board for instance - then the voter indeed voted for the candidate encrypted by the ballot. An attack on this property would imply that a ballot registered for a honest voter does not encrypt their voting choice. The attacker would have the possibility to "switch" the voter's vote.

Now, although satisfying the Cast as Intended property is not a sufficient condition for Verifiability, it does appear as a necessary condition. This is merely an intuition that has not been proven in this thesis. Finding an attack against this property could imply that the attacker has the possibility of impersonating a voter and vote for them.

But, if this is a possibility for an attacker, the attack against privacy we found on our protocol appears more as a generic attack an attacker could perform on any voting scheme that does not guarantee this Cast as Intended property. The attacker would impersonate the honest voter Alice to guess the vote of Bob.

Of course, the question we raised here is not proven, and we did not have the time to treat at the moment this thesis was written. However it would seem interesting to investigate further on whether vote confidentiality imply verifiability.

Conclusion and discussion

This chapter exposed our protocol's formal analysis regarding verifiability and privacy.

We applied our theorems stated in the previous chapter to the study of our protocol's verifiability in an extensive study of each possible corruption scenarii. Then, the same kind of analysis was done

regarding its privacy. While two kinds of attacks against privacy showed that the attacker could directly guess the voter's vote, one other kind allowed the attacker to indirectly guess this vote.

This last attack showed us that the attacker would enjoy the fact that the protocol does not guarantees the Cast as Intended property by casting a vote instead of a honest voter and use this knowledge to guess the other voter's vote. We gave some thought on why we thought this attack could be a generic attack for voting protocols that are not verifiable.

First, we would need to question the importance of the Cast as Intended property regarding verifiability. Could it be a necessary condition for verifiability? The study could began for protocols that do not satisfy this property - to show that they would not be private because of this attack that appear as quite generic.

Finally, one could argue that the attack we found, that is detailed in section 4.2.3, seems unpractical: an attacker would need to corrupt all votes but Alice's to guess her vote. While we could debate on this, we have to admit that the attacker indeed does not *directly* learn Alice's vote. Thus, we could also work on a weakest definition of vote confidentiality, where we would expect a protocol not to allow an attacker to learn a voter's vote using all other ones.

Part II

Payment Protocol

Chapter 5

A landscape of the mobile payment industry and its main limitations

By the time this thesis was written, the mobile payment industry was in full growth.

Indeed, contactless payments have grown steadily for almost a decade now and the first mobile payment applications go back to 2011 with the launch of Google's solution: Google Wallet. Most actors and wannabees of the bank and payment industries predict that the majority of transactions in the occidental world will be done with a smartphone by 2020.

Nowadays, a profusion of mobile payment solutions exists and such applications require two different worlds to interact: the classical payment infrastructures - banks and payment networks, processing every transaction made at a merchant point of sale - with the payment service providers - developers and providers of mobile payment services. On part of the classical payment infrastructures, we can see that very few modifications have been done for the past twenty years. Banks and standardization committees await for the mobile payment solution which will overcome the stiff competition to actually adapt the payment ecosystem. Meanwhile, each payment service provider is proposing their new application and the security can be handled quite differently from one solution to another. From basically developing

an application that imitates Chip and PIN cards behaviours to involving more recent concepts such as creating an alias for payment data to avoid data breach (tokenisation) or relying on new security components available in modern smartphones which could for instance allow the secure identification of a user, nowadays technologies allow the application providers some latitude for implementation even if at the end, they have to comply with decades-old payment standards.

The main goal of this chapter is to give the reader some insights about what is technically at stake when dealing with the nowadays mobile payment industry, specially regarding the security of mobile applications. We will begin by stating what are the technical constraints one can encounter when trying to propose a new mobile payment application, on both the payment network and the mobile sides. We will also expose a quick survey of four existing payment solutions and how they each deal with security. We believe those payment applications' approaches to security are quite representative of what technical solutions the industry offers nowadays. Finally we will discuss some improvement possibilities regarding security.

5.1 Technical Constraints

When designing and proposing a new payment application, one shall have in mind that the payment ecosystem set by major payment network (like Visa and MasterCard) and bank companies for the last three decades is more or less untouchable. Unless one of those entities that have a say in the standards ruling the payment industry is backing up a specific solution, conformity with such standards is the best way to hope for a scalable solution.

Speaking of standards, there comes the first big restriction. EMV is the international standard for most card based transactions. As most of mobile payment solutions rely on an account created with a card and since contactless terminals are for the vast majority EMV-certified, for a payment solution to be as much scalable as possible, it shall respects those standards. This section will provide an overview of the essential points of EMV standard - the actual payment ecosystem, what is a transaction and how the payment data authentication is processed.

We will also discuss the security management of payment applications on mobile devices. For such versatile devices, several solutions exists, all with their own pros and cons. We can more or less separate them into hardware and software based security solutions. We will try to explain the main security issues they could be confronted to but also the matters of costs and performances related to each of them.

Although the payment ecosystem has almost not evolved since the 90's, it has recently - actually at the beginning of this thesis - been quite massively modified with the advent of tokenisation applied to payment. Because card data are nowadays used in several ways outside of the pure contact and terminal-based transaction (e.g. internet payment, contactless, mobile payment...), EMVCo, the consortium in charge of mobile payment standards, introduced tokenisation which is basically a framework to create aliases for payment cards number. However, the framework does not provide any real specification or protocol for tokenisation which could be standardized and available to the public. We will provide the reader with an overview of what is actually part of the framework and what is lacking.

5.1.1 EMV compliance

EMV is an international security standard for smart card-based transactions. The initials stand for Europay, MasterCard and Visa, the three companies that originally wrote the standard's first draft back in 1994. It is now managed by the EMVCo consortium [50] controlled by Visa, MasterCard International, JCB, American Express, China Unionpay and Discover (Europay has since been merged with MasterCard). The standard's first scope was to provide a technical specification for card-present transactions,

when the payment card needs to be physically inserted on the merchant point of sale (such cards are also known as Chip and PIN or Chip and Signature cards). However, EMV specification now offer technical frameworks for a larger spectrum of transactions such as contactless payment (NFC or QR-code-based), internet payment (3D-Secure) or tokenised payment. Yet, those frameworks do not offer end-to-end security protocols for every kind of transactions, those are left for the service provider to design. Only the original EMV specification propose concrete protocols which are supposed to be supported by payment smart cards and EMV-certified terminals. This is understandable since most of those new kind of transactions are actually quite recent, the industry is most likely waiting for the most practical design to stand out before standardizing it.

There is though a common ground for all those specifications. No matter what, all transaction information that need to transit over the payment network must respect a specific format. Also, since there is little chance a payment solution will be disruptive enough to impose a massive update on all terminals, a new payment design better shall respect the already existing EMV requirements.

This section will detail what kind of requirements should be fulfilled by a mobile payment application.

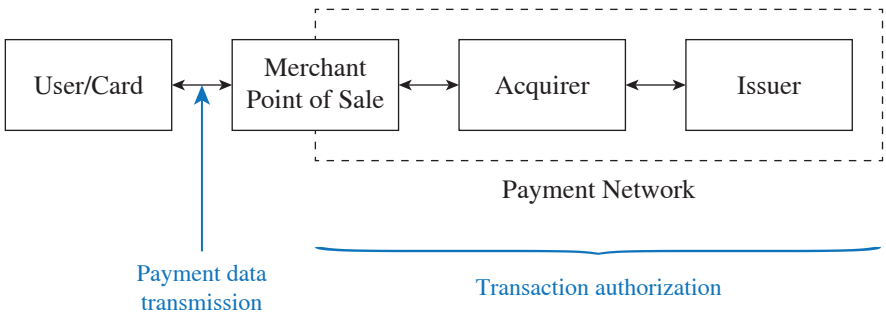


Figure 5.1: The EMV payment ecosystem.

The EMV Payment Environment

Fig.5.1 details the classical payment ecosystem for a card-based transaction. Basically the *acquirer* is the merchant's bank and the *issuer* is the payer's bank (that provided them the payment card). The *payment network* is the network to which the merchant's EMV-certified point of sale is connected. When designing a payment solution, there is little to no chance on having any influence on anything part of the payment network.

Essentially, for a transaction to be valid, the merchant needs to obtain the cardholder's Primary Account Number (PAN) and its expiry date. Those are the minimal information required to validate a transaction but the issuer could ask for other additional data. The PAN is actually the 16-digits number that can be found on a payment card. It holds all the information to be correctly routed over the payment network on its first six digits (also known as the Issuer Identification Number ; for instance, Visa payment cards all begin by 4 followed by five digits that identify the actual Visa issuer of the card).

An EMV transaction relies on three operations: the payment data authentication, the user identification and the transaction mode setup.

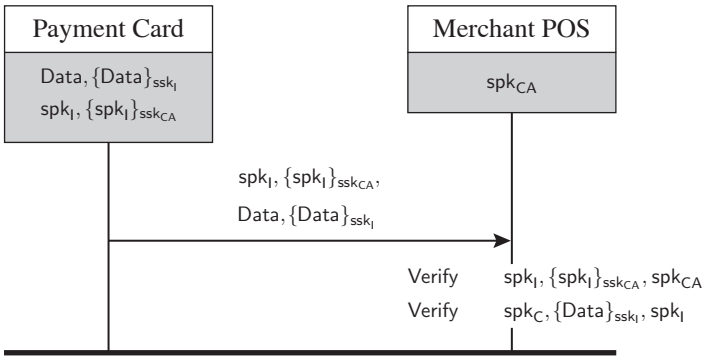


Figure 5.2: The EMV-SDA protocol

Payment Data Authentication

Several stages are at stake during an EMV transaction - initiation, retrieval of data, approving or not the transaction - but the core protocols are the payment data authentication ones.

The payment data transmitted to the merchant terminal shall be authenticated to prove they were provided by an actual issuer. Three EMV protocols are supposed to ensure this [52]: the Static Data Authentication (EMV-SDA), the Dynamic Data Authentication (EMV-DDA), already described as our running example in chapter 1 (see Example 10) and the Combined Data Authentication (EMV-CDA). EMV-SDA is the default go-by data authentication protocol for smart cards if no higher version is implemented on it. It is thus available in every EMV-certified payment terminal and would be a smart choice for high scalability of a new payment solution. Fig.5.2 provides an overview of the SDA protocol.

During an SDA transaction, the card provides its data (PAN, expiry date and other data required by the bank) signed by the card issuer. The card also gives the merchant terminal the signing public key of the card issuer, certified by a *certification authority* (CA). Since the terminal owns the CA public key, it can verify the authenticity of the issuer's public key and then, use this key to verify the authenticity of the transaction data provided by the card.

The EMV-SDA protocol does not guarantee the payment data authentication since it is vulnerable to a replay attack detailed in Fig.5.3, which is actually used to produce YesCards.

The attacker can retrieve the client's payment data - by sniffing them from a point of sale during a transaction for instance - and replay such data during another transaction. If an EMV-compliant payment protocol is designed based on the EMV-SDA scheme, it should address this replay attack and solve it.

User Identification and Transaction Mode

The user identification is an *optional* process that can be performed during an EMV transaction. It can be a simple signature from the client - as usually done in the United States - or a PIN verification. The client inputs the PIN on the merchant terminal and the verification is either done online by the card issuer or offline by the payment card which will either receive it encrypted or as a plaintext for verification. Interestingly, among the fact that this operation is indeed optional, the user identification is usually performed *after* the card authentication is processed in both EMV-SDA and EMV-DDA. So the actual payment data are sent to a merchant before the user identified themself to validate the transaction. This design flaw was discovered in 2010 by Murdoch, Drimer, Anderson and Bond [92] who demonstrated that even the EMV-DDA protocol which was supposed to be invulnerable could be bypassed. However, although the attack is now well understood and public, the EMV-DDA protocol is still used in Europe.

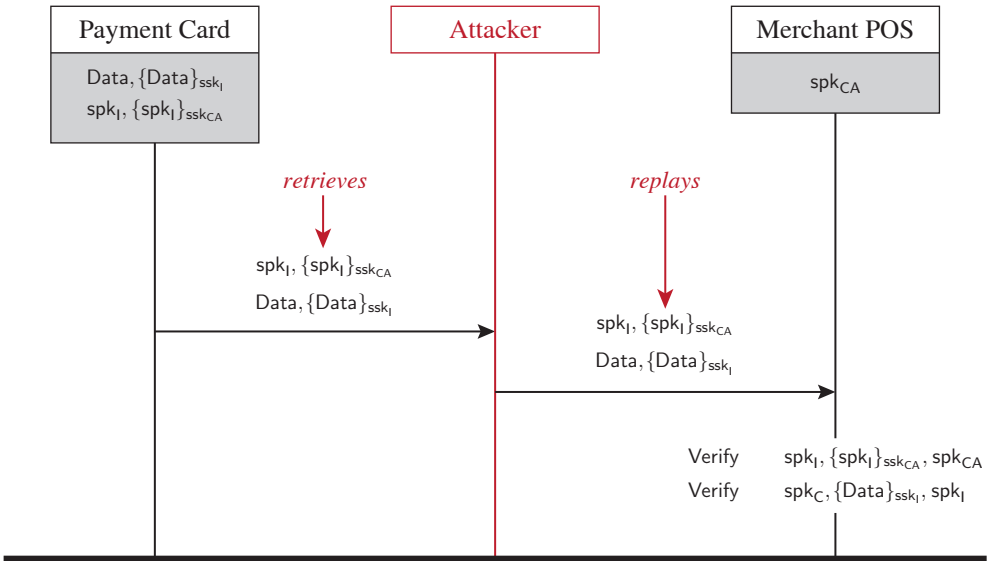


Figure 5.3: The EMV-SDA replay attack

The transaction mode set by both the payment card and the merchant terminal defines whether the transaction authorization will be performed online (on the fly) or offline. When performed online, the merchant point of sale asks the issuer through its acquirer if the transaction is valid. For contactless transaction, best practices indicate that the online mode is required.

5.1.2 Security Management of Mobile Payment Solutions

EMV's security strongly relies on the asymmetric keys and certificates held by the payment card. While extracting keys from a credit card is a strenuous task, this is no longer the case for keys held by applications running directly from a mobile device operating system. To fight against frauds and cloning, a mobile payment protocol shall be secure even if it runs on a device that offers low level of security and/or contains malicious applications.

Although it is quite hard to truly assess the security of proprietary mobile payment solutions, we can observe two main trends followed by major companies to manage security.

Secure Element Centric

The first trend shows a growing interest in the use of Secure Elements (SE) [65]. A Secure Element is a hardware and software tamper-resistant dedicated platform that can securely host and run applications and store (cryptographic) critical data. It can either be a UICC card (widely known as SIM card), a micro-SD card or be embedded in the device (as in Xiaomi or Apple smartphones) and can be compatible with NFC. Since 2011, the access to a Secure Element from the smartphone main (Android) OS has been the scope of a standardized specification [106].

It should be noted that a secure element only operates in slave mode thus it cannot be used alone. Its activation is usually done after the user authenticated themself (through a PIN code or a fingerprint for instance).

One solution when developing a mobile payment application would be to implement a full existing payment solution, such as the EMV applet which can be found on any payment card, onto the Secure

Element. This solution has the advantage of its simplicity: it inherits the security guarantees of an existing, well understood and already widely deployed protocol.

However, in terms of costs, the implementation of a full and complex applet on the Secure Element can be highly off-putting from an industrial point of view. It has to be an optimized solution since Secure Elements do not support too intensive computations compared to mobile devices main OS. This kind of limitation also make the maintenance and updating of applets quite tedious: any update implies to revise the Secure Element content through erase and write operations on the non-volatile memory, which supports only a limited number of write and/or erase operations. In contrast, it is way simpler to update an application on a mobile platform. In addition to all that, any application implemented for the Secure Element needs to be certified before being uploaded on it. This is a long and costly process and another reason for refraining from updating the solution: any update on the application source implies a new certification process. Last but not least, the memory size available on a Secure Element is yet another limitation, the bigger the application, the more space is required and this implies more expenses on development and certification. For such big applications as the classical EMV ones, it can very quickly come at a high cost in terms of time and expenses.

Host Card Emulation

The other main trend relies on Host Card Emulation (or HCE) solutions which get rid of the device-based SE and provide a full software application. In this case, the contactless card is directly emulated by the mobile device.

Nowadays market offers two main options regarding HCE solutions.

- **Cloud-Based:** Cloud-based solutions deport the sensitive operations (those executed by the Secure Element in the previous kind of solutions) on the cloud. The main drawback of such an approach is that the user needs to be connected to the cloud, through cellular connectivity (3G or 4G) or Internet, in order to process a payment. This comes with all the costs and availability issues it involves (like roaming costs if the client is outside the country, network coverage problems that can happen on the underground or elsewhere...).

- **Obfuscation (White-Box Cryptography):** The other kind of HCE-based applications rely on white-box cryptography designs, obfuscating application keys and cryptographic operations inside the application code itself. One first issue of this approach is the setup (or enrollment) of the user which is in fact an unsolved issue at this point. It is actually impossible to pre-share personal keys between the phone and the payment service, since the application will be downloaded directly from a play store. The personalisation of the payment application will indeed be as problematic as the personalisation of a Secure Element although is the latest case, the operator can have a direct access to the Secure Element thus bypassing this problem.

  But the main issue comes actually from the use of white-box cryptography itself. All current solutions implementing white-box-designed cryptographic primitives are broken as demonstrated by [27], which provides a generic attack called Differential Computation Analysis (DCA) which allows the extraction of key material from any published white-box implementation. The DCA Attack is significantly fast, since being automated, and requires no specific knowledge of the white-box design from the attacker.

We can then assess the security management for mobile payment solutions could be improved. To be practical and ease the developers work, it shall rely on HCE. However, since existing HCE solutions

either lack availability or security guarantees, it seems the use of a Secure Element is still the safest option.

5.1.3 Tokenisation

In 2014, EMVCo introduced the concept of tokenisation applied to payment solutions by providing a framework specification [55]. In essence, tokenisation is the replacement of the Primary Account Number (PAN) with another 16-digits number (and optionally other payment data) so that the circulating payment data are not the actual original ones but aliases. However, EMV Specification does not provide a full end-to-end example of payment protocol based on tokenisation, in fact, the main scope of this document is to provide a level of commonality across the payment ecosystem to support the adoption of token-based payment solutions. So it focuses on the requirements regarding the environment and infrastructure of tokenisation for better scalability in existing payment networks but does not provide details about the actual mechanisms behind token provisioning and token-based payment.

This section will provide an overview of the main information available on the document and thus enhance which are the lacking ones necessary to define an actual payment protocol based on tokenisation.

Tokenised payment environment

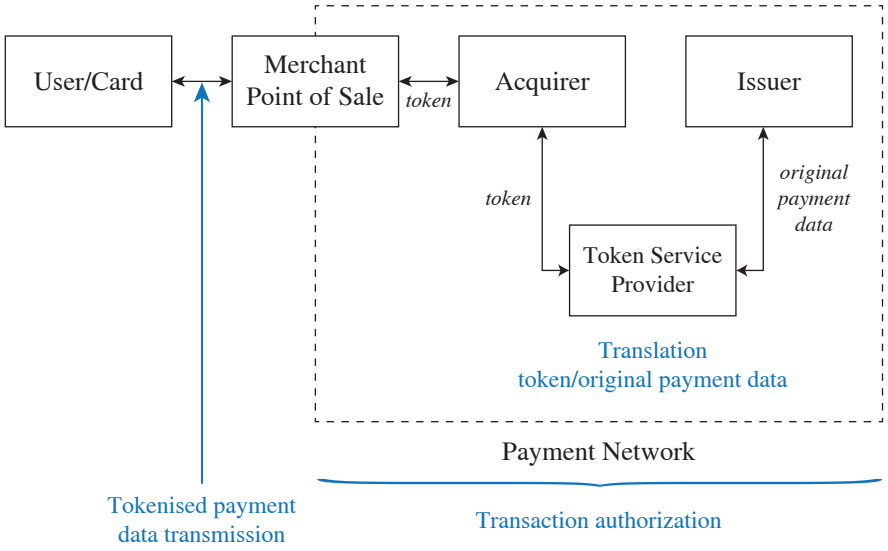


Figure 5.4: The tokenised payment ecosystem.

Fig.5.4 describes the tokenised payment ecosystem as defined in EMVCo specification. It actually is quite the same as the classical one previously depicted in Fig.5.1.

As before, the user holds a payment medium but instead of holding a PAN, it holds another 16-digits number used as an alias for the PAN: the token. Using an alias instead of real static credential can be used to improve the security - by diminishing the impact of data breaches - and anonymize transaction on part of the merchant. As a classical PAN, the token contains all the information to be routed over the payment network on its first 6 digits. The only thing differentiating an original PAN from a token is its

Luhn check-digit: a PAN has a Luhn check-digit[15] of "0" whereas it is "1" for a token. So technically, a token shall be used to run a classical EMV transaction.

The payment process can then be performed either on the internet or at a physical point of sale, the payment data are then the tokenised one (even the expiry date matches the token's one, not necessarily the original PAN's one). As before, the transaction authorization request is sent by the merchant's acquirer over the payment network, however, instead of transmitting it directly to the issuer, the token has to be translated into the original PAN by a new entity: the *Token Service Provider*.

The Token Service Provider

The Token Service Provider is a new actor in the payment ecosystem which is located on the payment network. This entity is in charge of generating tokens and provisioning them to the *token requestor*. The token requestor could either be a cardholder wanting their PAN to be tokenised or a merchant which could not comply with security measures [72] to store original payment data but would still like to store some useful payment information.

Aside from the provisioning part, the TSP manages a secure vault to store the tokens and the original user's payment data. This allows the TSP to ensure the translation of tokens to PAN between the acquirer and the issuer during a transaction as well as processing some payment verification not defined by the specification.

Advantages of tokenisation for security matters

The EMV tokenisation framework lacks some clear description about how the security management shall operate to let each service provider define their own solution. However, one can still extrapolate some new basic security policies that could be set from what is described on the document.

- **Long or Short-Term Tokens:** Each time a token is created, it comes with its own expiry date, a 4-digits variable. A service could rely on long-term tokens which could be used similarly to a classical PAN with the same longevity or short-term tokens. A solution could even rely on one-shot tokens created and/or usable for only one specific transaction.
- **Service-based Tokens:** Since the TSP is in charge with all the security management around tokens, it could easily provide tokens usable for a specific service (such as internet transactions or mobile payments), a specific technology (NFC, Magnetic Stripe...) or even with a specific merchant. Such policies could also be set in agreement with the user.
- **Privacy:** Data tracking is no surprise nowadays. Merchants could even track customer habits with their payment data since they are static ones (and even if the name is an optional EMV field, it is still transmitted in most cases). However, with a one-shot token, we could hope for some unlinkability between client and payment from a merchant point of view.

5.2 A survey of existing mobile payment solutions

Several payment solutions have been developed such as Google Wallet, Apple Pay, or Orange Cash. Since these applications are proprietary, it is very difficult to assess their security. Nonetheless, this section will try providing a quick survey of four mobile payment applications which are quite representative on how security can be managed in the nowadays industry.

[15] a checksum formula used to validate numbers such as PAN, user's SIM numbers...

5.2.1 Apple Pay

Apple launched its mobile payment application in October 2014 [82, 83, 75, 76]. Although there is no open specification of the in-depth security management inside the application, it is still possible to understand at a high level what kind of mechanism operates in this solution.

Users sign up to Apple Pay by first entering their original payment card data. Those payment cards data are sent by Apple's server to the user's issuer. It then appears that the issuer itself chooses the Token Service Provider which will be in charge of generating a token and the additional cryptographic data which will be useful when a transaction is processed. This token, called Device Account Number (or DAN), is supposed to be different from the original PAN except for the last four digits which they both share. The DAN is sent back to Apple which will redirect it to the device's Secure Element along the cryptographic data. The DAN inherits the original PAN expiry date, so we are here dealing with a long term token. Once the registration is over, Apple is not supposed to store the original payment data.

During a transaction, the DAN is sent instead of the actual credit card number, as a token is supposed to be, along a dynamic transaction cryptogram generated by the Secure Element with the cryptographic data sent during the setup phase. The issuer (or its TSP) will verify the cryptogram to validate or not the transaction. Moreover, a transaction will not occur unless the user previously identified themself with the device identification medium (Touch ID, lateral button of the Apple Watch...).

A token without the dynamic code is supposedly useless for a transaction.

5.2.2 Google Wallet and Android Pay

Google payment solutions where respectively released in 2011 and 2015 [74, 61, 63, 62, 79, 13, 46, 12, 11]. Google Wallet work as a peer-to-peer payments service: it allows money transfers between individuals through their mobile device or desktop computer. It is not, per say, a pure mobile payment solution, for it also works on computers - a user would only need an e-mail address, Gmail, obviously - and is closer to PayPal regarding the way it works. The application used to also allow NFC-based payment on physical merchant terminals until the launching of Android Pay, which is now the actual mobile - and smartwatch - payment application from Google.

Like Apple Pay, and actually like every mobile payment solution, the user registers by entering their original payment card data. The payment data are then stored on a Google cloud, and there we have the main difference with Apple Pay: Android Pay is a full HCE solution whose security relies strongly on the operating system and connectivity.

Android Pay also uses tokens to replace payment data. However, rather than a long term token stored on the device, during each transaction - NFC or online - a unique randomly generated 16-digit number is sent. These tokens are generated on the cloud. If the user has internet or data connectivity, the tokenisation request as well as its provisioning on the mobile phone will be done on-the-fly. However, in case the user is in a dead zone, a limited number of tokens are directly stored on the device. Where and when this provisioning of backup tokens is, to the best of our knowledge, not publicly known for now.

As additional security features, a transaction cannot happen while the phone is locked and a correct mobile fingerprinting - to see if the user's behaviour on the phone is consistent- is required.

5.2.3 Samsung Pay

Samsung Electronics launched its mobile payment and digital wallet solution in 2015 [70, 96, 67, 114, 112, 113, 71]. Like all other solutions, it can handle NFC payment however, the system also work with terminals that do not support NFC payments but can process magnetic stripe transactions: indeed, in 2015, the company acquired the mobile payment start up LoopPay which developed the specific Magnetic

Secure Transmission (MST) technology that Samsung Pay uses to be compatible with magnetic-stripe-only terminals.

Similarly to Apple Pay, the user registers their original payment data which are then redirected by the Samsung's Token Request service to the user's issuer which will itself choose the Token Service Provider. A long-term token will then be provided to the device, as well as cryptographic keys that will come in useful during a transaction. Everything is stored in Samsung's Trusted Execution Environment (TEE): Samsung KNOX. The security management is not Secure-Element centric per say, since Samsung's TEE is able to provide a Trusted User Interface which allows the user's PIN entry and/or fingerprint reading to be done securely as the TEE takes control of the device screen, a feature that cannot be provided by secure elements.

If the registration of the user and the token provisioning is similar to Apple Pay, the token handling is actually closer to Google's solution. If the user is online, the application does not use the keys stored in its TEE. It will make a request for single use keys to Samsung's service. Limited use keys will then be generated and provided to the device which will use them for the transaction. However, when the device is offline, the TEE itself will generate the transaction cryptogram with a key calculated from a static key stored in the TEE. Unlike Google's solution, the token itself is not a single-use one, it is in fact the static key that has a limited time and/or number of use according to the TSP policy. How the replacement of the key itself happens is not clearly described.

Several flaws and issues were actually found on Samsung Pay in 2016 [88]. During an online transaction, the key randomization appeared to be quite predictable which allowed an attacker to guess a token's key from a previous one. Moreover it appeared that once the long term-token were stolen with one key, there was no actual protection against their use on another device.

5.2.4 Orange Cash

Orange Cash was launched during the fall of 2015. The payment application can only be used by the operator client with a compatible SIM card for Windows and Android smartphones which will act as a Secure Element (no specific requirements for iOS on the last matter, since the payment application is run in conjunction with Apple Pay, the security will be managed by the embedded iOS Secure Element). Orange Cash is a Secure Element centric payment solution the operator developed in collaboration with WireCard, a German financial service provider with an e-money license. Orange is in charge of the main OS application development and maintenance whereas Wirecard shoulder the responsibility for the Visa applet implemented on the Secure Element.

During the registration process, the user sends its personal details (name, birth date and address) to Orange's servers which redirect them to WireCard. WireCard will then create a virtual payment card (a long term token with the same expiry date as a physical card) with associated EMV-compliant data (like cryptographic keys). Next, WireCard personalises an EMV-compatible Visa applet with those data which will be sent encrypted to Orange which, as an operator, will process to install it on the mobile device's Secure Element.

Once everything is setup, the client has a virtual wallet on their smartphone that needs to be recharged, either with a direct transaction inside the application with a payment card (and Orange will charge a small fee for payment card transaction) or by proceeding to a bank transfer (longer but with no charge).

During an NFC transaction, a payment card will be emulated by the Secure Element and will transmit the tokenised payment data through the device NFC antenna. A classical EMV transaction based on NFC will occur and the money will be deduced from the wallet. Payments under 20 euros do not require any identification from the user, otherwise the user will have to enter a PIN code from the device main OS which will be verified by the Secure Element.

5.3 Improvement possibilities regarding mobile payment applications

As we stated in the previous sections, although little could be done to actually transform the payment ecosystem, there are still some latitude left to mobile payment service providers. We list here three improvement possibilities for mobile payment services.

5.3.1 Devising an open mobile payment protocol specification

As previously seen, there are no actual open specification for a payment solution adapted to the mobile environment. Information about what is actually going on proprietary solutions can be tedious to gather and therefore, it would be an almost impossible task to independently analyze the security of a payment application. However, we shall acknowledge that some efforts have been done to provide at least some knowledge on how to implement some payment-related applications for Android and Samsung developers [66, 49]. But those information do not offer an in-depth understanding of what is going on inside the mobile device and are far from providing a clear overview of the whole mechanics ruling applications.

We can also raise a paradox from the registration methods used for the vast majority of application. If the whole point of using tokenisation is to avoid the disclosure of original payment data, how come that users still have to provide them when registering to a payment application. Sure, in some cases - Apple Pay, Samsung Pay and other payment solutions based on a trust zone - the registration process is done through an allegedly trusted user interface, and after all, the services only require payment data to be sent once over the network, but still how do we ensure real end-to-end security when the first part of the process is fragile?

Nonetheless, we may indulge a little this lack of transparency, for it appears that actors of the traditional payment industry are actually waiting to see emerge the most competitive application before settings some actual standards. After all, at least EMV data authentication protocols have been made public and their publication allowed the unveiling of security breaches. And let us not forget that certification authorities might still want to keep this market to themselves.

5.3.2 Improving the security management

HCE solutions offer huge cost reduction regarding development and certification. Moreover, an ideal HCE solution offers more flexibility regarding updates than Secure Element centric applications. However, as evoked in the previous sections, they lack real security and/or convenience for the user regarding connectivity.

On the other hand, Secure Element solutions lack flexibility and the certification process can be heavy, long and expensive. But they appear to be the safest solutions so far. However, since Secure Elements only operate in slave mode and some improvements could be made on the access control from the main device's OS, one could argue that the identification processes of users made from the main OS is not satisfying enough.

One solution could be exploring the possibilities offered by Trusted Execution Environment (TEE) on smartphones and tablets. TEE are secure hardware-based enclaves that can, similarly to Secure Elements, host sensible data and perform cryptographic operations. However, the technology is still young and only available in most recent device. Yet, they bring some undeniable asset Secure Elements don't: a Trusted User Interface, which could allow a user to securely identify themself.

5.3.3 Adding some privacy

The last few years have raised some actual concerns about consumers' privacy. Data tracking is becoming a mainstream issue as reflected in the numerous initiatives one can now find on the topic.

Of course, tokenisation as defined by EMVCo still requires a centralized payment and transaction data management for the TSP has a central role in the scheme. However, we could enjoy some privacy regarding consumption habits on part of a merchant thanks to the ephemerality of tokens.

Conclusion and discussion

We tried to provide the reader with some insights about the payment industry. It is indeed a vast industry ruled by standards that have been in place since the middle of the 90's. Compliance to the EMVCo standards is one of the key ingredient to provide a scalable payment solution.

We also discussed the security management trends existing in the payment industry and gave a quick survey of existing mobile payment applications that exposed how those trends were actually implemented "in real life". One of the main limitation in our short outline was the fact that those are proprietary solutions therefore there is no open specification that would allow us to extensively comprehend the way they manage their end-to-end security, instead, we needed to rely on patents and public communication.

In addition to this, we also proposed some improvements that could be made regarding the possibilities of nowadays technology. Indeed, although there is little to no chance to sway the existing payment infrastructures, a mobile payment provider can still enjoy some flexibility on part of the user terminal. Moreover, with the emergence of tokenisation applied to payment data, payment infrastructure added a new entity to their ecosystem: the Token Service Provider which is in charge of translating payment (card) data into aliases. By the time this thesis was written, only the role of the Token Service Provider was defined. Yet, the way it operates was left to the service provider discretion. Since the Token Service Provider can be seen as the ling between a client device and the payment ecosystem, one could create new security policies based on specificities of the mobile environment - restrict payment data usage to a specific kind of transaction based on technology, time and/or merchants - that would be checked by this new entity.

Because of this lack of open specification and the fact that there could be some improvements regarding security, we proposed an end-to-end mobile payment specification implementing tokenisation. This is the scope of our next chapter.

Chapter 6

Designing an EMV-compliant Payment Protocol for Mobile Devices

As stated in chapter 5, by the time this thesis was written, there was no open end-to-end specification of a mobile payment protocol. Moreover, We identified some improvement possibilities regarding the security of a mobile payment solution.

- Proposing an open specification of an end-to-end mobile payment application.
- Improving the security management by finding a compromise between full hardware and full software-based solutions.
- Adding some privacy for the users regarding their consumption habits.

We tried to answer to those issues by proposing a transparent specification of a mobile payment protocol with the practical constraints formerly expressed in mind.

For more scalability, the solution had to be fully compatible with the EMV-SDA protocol, since it is the default go-by protocol for payment data authentication at a point of sale. Yet, as seen in section 5.1.1,

the EMV-SDA protocol is vulnerable to a replay attack: the payment card transmits static payment data. Such values could be exploited by an attacker who could use them in another transaction to validate a payment. In order to prevent from such an attack, we needed to "inject" some dynamic data in the payment ones while still being compliant with the EMV-SDA specification.

This was made possible by the use of tokenisation: instead of transmitting original - and static - payment data, we use *aliases* (tokens) respecting the simple principle "one transaction = one token". This approach does not only address the problem of privacy, it can also be used to add some privacy for the user: because the payment data are unique for each transaction and since merchants cannot translate those token to the original payment values, they would not be able to track a client's consumption habits.

Thus, we wanted to propose a mobile payment protocol relying on the use of dynamic tokens without impeding the user experience: offline payment should be allowed so that even when the network is not available during the transaction - because of connectivity issues or because of its cost - the client would still be able to pay from anywhere at any time.

There came the question of implementation possibilities. Because of the flexibility and practicality Host Card Emulation (HCE) solutions offer to developers - and therefore to the industrials who employ them - we wanted our payment protocol to run as an HCE application on the user's mobile device. Yet, because of formerly explained security issues HCE solutions could raise[16], we needed to rely on some hardware-based security. Hence we needed to include the use of a Secure Element in our solution.

One could ask why we did not choose to exclusively rely on Secure Elements to implement our solution. While it is the case of some solutions in the market - Apple Pay and Orange Cash for instance - the costs, in terms of time and money, of such solutions increase with the length of the application because of certification issues. Moreover, Secure Elements are usually not as powerful as smart devices, thus it would be a shame not to enjoy the possibilities of nowadays technologies. This is why we chose to design our protocol as a *hybrid* solution that run as an HCE application but partly rely on the (limited) use of a Secure Element.

Thus, we hardened the cryptographic operations performed by the Secure Element as well as the data stored on it to the minimal with no loss of security. In fact, we formally proved our protocol guarantees payment security for both the client and the merchant even if a malicious application dumps the mobile device memory. We only require from the Secure Element to store two symmetric keys and to perform very basic operations: MAC computation and counter management. We believe that such functionalities are likely to be available on most Secure Elements and should be easier to certify than a full implementation of the EMV protocol [43]. Plus, since the functions are generic, they could be used by other applications for other purposes than payment.

We begin this chapter by establishing and defining the role of the different parties of our protocol. We also provide a detail specification of said protocol. We will then explain what kind of threats were taken into consideration and what kind of trust assumptions were made regarding the security of the protocol. We then provide what we believe our protocol offers in terms of security - for the user, the merchant, the bank... And we finish by presenting a prototype of payment application we developed at Orange in order to give some perspective regarding performance impacts of a hybrid HCE-SE solution.

6.1 Presentation of our protocol

This section provides a detailed overview of the token-based payment protocol adapted to mobile devices that we designed. *We do not cover the enrollment part of the process*, as it requires private agreements between the bank industries and the mobile device constructors and/or operators.

[16] See section 5.1.2

As previously explained in section 5.1.3, the EMV specification for tokenisation does not explicitly state whether a token shall be a long or short term value, leaving this choice to the implementer. Thus, we decided to design tokens as one-time surrogate values for the original Primary Account Number (PAN)[17]. They are used during a transaction in an EMV-SDA-compliant message. Since each one of these tokens are generated by the Token Service Provider[18] and are consumed after one transaction, the user needs to regularly provision their mobile device with them.

Thus, our protocol is divided into two phases: first, the *Token Provisioning*. From their mobile device, the user will send a request to the Token Service Provider for fresh tokens. They will be sent and stored encrypted on the device. Then comes the *Token-Based Payment*. The user will process to a payment with one of the token received during the previous phase. Those processes will be described in detail in this section.

Our protocol technically relies on the use of a Secure Element and provides an open specification compatible with the EMV tokenisation framework. For more scalability of the solution, it was designed to be compliant with the EMV-SDA protocol.

The design as well as the results regarding this protocol were published in [36].

6.1.1 Entities

Several entities intervene during the execution of our protocol, we will describe their role in the following section. As mentioned earlier, the protocol does not cover the user registration to the mobile payment service.

Token Service Provider (TSP)

The first entity we rely on had its role defined in the EMV-tokenisation framework [55]. The *Token Service Provider* (or TSP) is located in the payment network, between the merchant acquirer and the client issuer, as described in section 5.1.3. The TSP manages tokens: it generates them, ensures their translation from/to PAN between acquirers and issuers is correctly done, provides the tokens to the client, securely stores them, processes to payment verification and ensures the obsolete tokens are deactivated and can not be used for further payments.

We assume the user is already registered to a TSP and is identified by their Token Requestor IDentifier (or $\mathsf{TR_{ID}}$). Each registered $\mathsf{TR_{ID}}$ on the TSP is associated to a pair of symmetric keys ($\mathsf{K_{ID}}$ and $\mathsf{K_{Pay}}$) shared with the user's mobile device as well as three counters (c_{TSP}, c_{Tok} and c_{Pay}) whose role will be defined later.

The TSP also holds a pair of asymmetric encryption keys ($\mathsf{sk_{TSP}}$ and $\mathsf{pk_{TSP}}$) and a pair of signature keys ($\mathsf{ssk_{TSP}}$ and $\mathsf{spk_{TSP}}$). As required by EMVCo, the signature key is certified by a *Certification Authority* ($\mathsf{Cert_{spk_{TSP}}} := \{\mathsf{spk_{TSP}}, \mathsf{sign}(\mathsf{spk_{TSP}}, \mathsf{ssk_{CA}})\}$). This certificate is sent along the TSP public keys to each mobile application during a registration process.

Cardholder (CH)

The *Cardholder* (or CH) is actually the user owner of both the original PAN account and the device through which payments will be performed. They are registered under $\mathsf{TR_{ID}}$ to the TSP. Since the CH is identified before processing the token provisioning request and before starting a transaction, we assume they hold an *identification value* (or ID_{val}) that they are the only one to know or able to provide,

[17] See section 5.1.1.
[18] See section 5.1.3.

depending on the identification method chosen. It could either be a PIN code, a biometric fingerprint, a scheme or any other identification method supported by a mobile device.

Mobile Device (MD) and Mobile Application (MA)

Under the term *Mobile Device* (MD), we consider smartphones, tablets or even recent IOT devices such as smartwatch that can host a Secure Element and perform payment. The *Mobile Application* (MA) is hosted on the mobile device's main OS (also called rich OS). It is the payment application through which the CH will require new tokens or process payments and is based on HCE [19]. Since the rich OS is considered as untrustworthy, the MA only manages public information that do not endanger the security of our protocol. Mainly, the MA holds the user's $\mathsf{TR}_{\mathsf{ID}}$ and the TSP public keys - one for asymmetric encryption ($\mathsf{pk}_{\mathsf{TSP}}$) during the Token Requesting process and the other one for signature verification ($\mathsf{spk}_{\mathsf{TSP}}$) during the Token reception and the payment processes - as well as the TSP signature verification key certificate $\mathsf{Cert}_{\mathsf{spk}_{\mathsf{TSP}}}$.

Trusted Enclave (TE)

What we call the *Trusted Enclave* (or TE) is actually the combination of two security tools that can currently be found on most of the recent smartphones: the *Trusted Execution Environment* (or TEE) [64] and the *Secure Element* (or SE).

The TEE is a secure area residing in the main processor of a mobile device ans ensuring the secure management and computation of sensitive data. the main difference with a SE rely on the fact that it can provide a Trusted User Interface: by taking control of the touch keys and the screen of the device it can build a secure path between the user and the SE. Since the rich OS is assumed to be untrustworthy, the TEE is summoned every time the user needs to be identified. So the TE is the entity holding the ID_{val} for verification. In practice, the ID_{val} verification could be performed by either the SE or the TEE but since in both cases a secure channel is needed between the TEE and the SE, we consider this technicality out of scope. It is only after a successful identification of the user that the TEE allows requests to be sent to the SE. So it has also some kind of an access control role.

The SE could either be a SIM card [2], an embedded SIM [68, 69], a Secure SD-card [9] or an embedded Secure Element. It holds two symmetric keys used after identifying the user. One is used during the token provisioning process (K_{ID}) and the other one during the payment process ($\mathsf{K}_{\mathsf{Pay}}$). It also manages a counter (c_{CH}) to prevent replay attacks. The SE can increment a counter and calculate a MAC value with variables provided from the main OS of the mobile device.

Merchant Point of Sale (POS)

A user pays through a *Merchant Point of Sale* (or POS). It could be a physical terminal as much as an Internet platform for payment. The POS - or its related merchant - is identified with a Merchant ID, M_{ID} which will be used as a payment information by the SE to sign a transaction. As specified by the EMV protocols, the merchant holds the public key of the Certification Authority ($\mathsf{spk}_{\mathsf{CA}}$) in order to verify the TSP public key validity when handled by the mobile application. We assume the merchant to be connected. this means the transaction mode is assumed to be online so that it can almost instantly be verified by the TSP. this shall not be a big constraint since NFC-based transaction are usually performed with an online transaction mode.

[19] See section 5.1.2.

6.1.2 Token provisioning request and process

You can see a token as an alias for the PAN that can only be used for payment once. so because of their ephemeral value, the user needs to be regularly provisioned with tokens on their device, hence the token provisioning process, first phase of our protocol described in Fig.6.1.

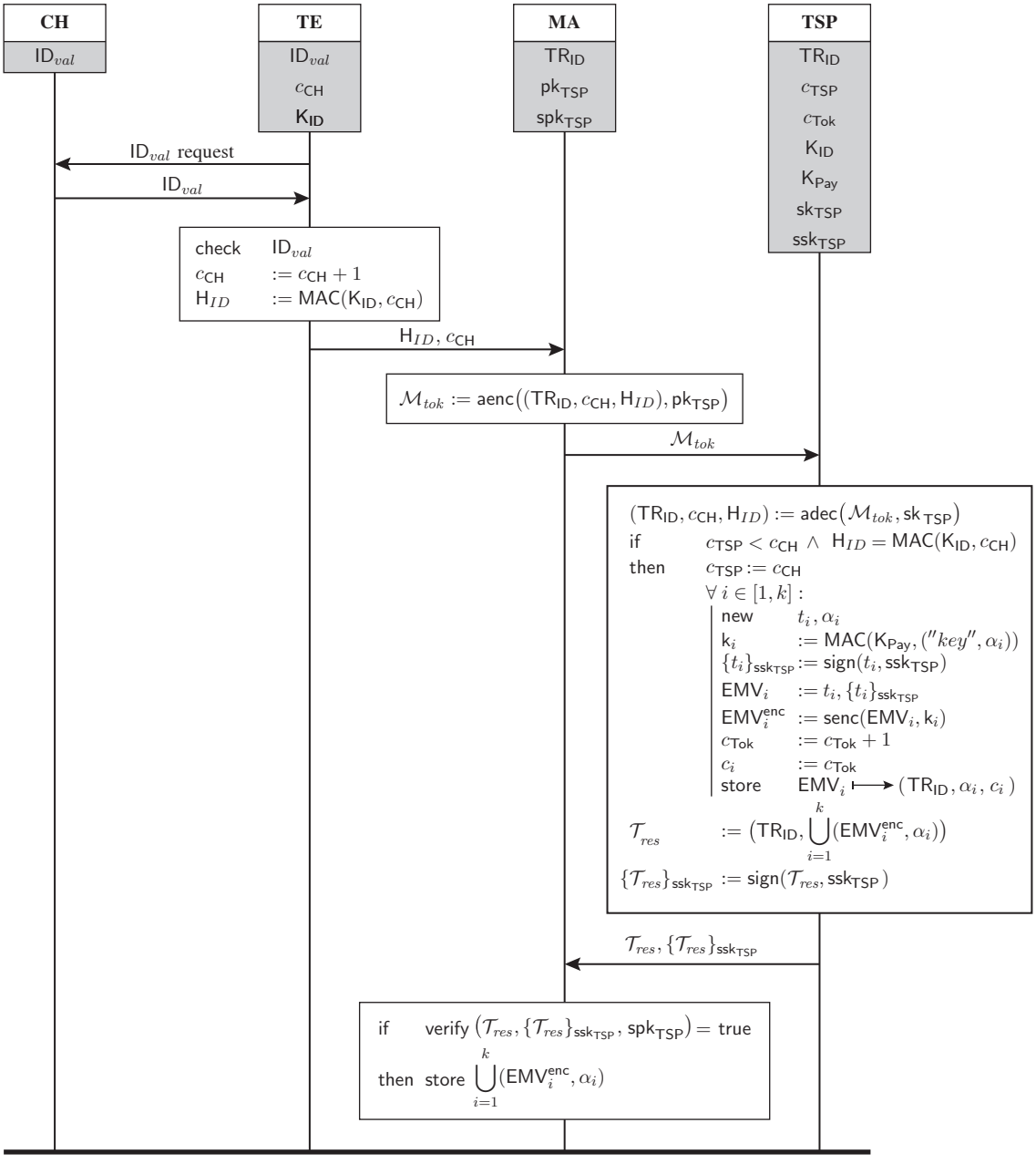


Figure 6.1: Token Provisioning request and process

First, the cardholder authenticates themself through their device using ID_{val}. After this first authentication, and only after that, the mobile application is granted access to the secure element to request the value H_{ID} computed from the symmetric key K_{ID} and the ongoing value of the counter c_{CH}:

$$\mathsf{H}_{ID} := \mathsf{MAC}(\mathsf{K}_{\mathsf{ID}}, c_{\mathsf{CH}})$$

This value will attest that both the right user and the right secure element were involved in the request.

The mobile application can then send the tokenisation request $\mathcal{M}_{tok}$ which is the encryption of $\mathsf{TR_{ID}}$, c_{CH} and H_{ID} concatenation with $\mathsf{pk_{TSP}}$, the TSP public key:

$$\mathcal{M}_{tok} := \mathsf{aenc}\big((\mathsf{TR_{ID}}, c_{\mathsf{CH}}, \mathsf{H}_{ID}), \mathsf{pk_{TSP}}\big)$$

After decryption, the TSP can first check that the counter value it receives (c_{CH}) is greater than the one it holds (c_{TSP}), with respect to a certain tolerance limit to be defined by the service. The TSP can then check the correctness of H_{ID}, since it also holds the key $\mathsf{K_{ID}}$. If the verification is successful, the TSP increments its own counter to the value of c_{CH}.

After these verification, the TSP generates k tokens: $t_{i,\ i\in[1,k]}$. The number k of tokens could be defined by the user when initiating the request or be set by the service as a default value. Each token is signed by the TSP, as required by the EMVCo specifications on payment data authentication:

$$\forall\, i \in [1,k].\ \{t_i\}_{\mathsf{ssk_{TSP}}} := \mathsf{sign}(t_i, \mathsf{ssk_{TSP}})$$

The TSP then computes k tokenised EMV payment packets:

$$\forall\, i \in [1,k].\ \mathsf{EMV}_i := t_i, \{t_i\}_{\mathsf{ssk_{TSP}}}$$

Each one of these tokenised EMV packets are ready-to-use payment values that are going to be stored on the user's mobile device's main OS.

Because we consider the mobile device to be an untrustworthy entity, the tokenised EMV packets cannot be stored as plaintext on it. Thus, the TSP encrypts them with a unique symmetric key. For each tokenised EMV packet, the TSP generates a fresh nonce (α_i) it then computes a symmetric key using the tag $''key''$, the nonce α_i and the payment symmetric key it shares with the SE:

$$\forall\, i \in [1,k].\ \mathsf{k}_i := \mathsf{MAC}(\mathsf{K_{Pay}}, (''key'', \alpha_i))$$

This key is then used to symmetrically encrypt each token it is associated to:

$$\forall\, i \in [1,k].\ \mathsf{EMV}_i^{\mathsf{enc}} := \mathsf{senc}(\mathsf{EMV}_i, \mathsf{k}_i)$$

With this precaution, if the device is somehow infected, stealing the mobile payment application data will have no significant impact on security for the user.

Each one of the generated tokenised payment packet EMV_i is also associated to a counter c_i calculated from c_{Tok} on the part of the TSP:

$$\forall\, i \in [1,k].\ c_{\mathsf{Tok}} := c_{\mathsf{Tok}} + 1$$
$$c_i := c_{\mathsf{Tok}}$$

This counter is used as a testimony of the token freshness when a transaction is validated.

The TSP stores the association of EMV_i with $\mathsf{TR_{ID}}$, α_i and c_i and answers to the tokenisation request by sending all encrypted packets along their associated nonce in the certified message $\big(\mathcal{T}_{res}, \{\mathcal{T}_{res}\}_{\mathsf{ssk_{TSP}}}\big)$, with:

$$\mathcal{T}_{res} := \big(\mathsf{TR_{ID}}, \bigcup_{i=1}^{k} (\mathsf{EMV}_i^{\mathsf{enc}}, \alpha_i)\big)$$
$$\{\mathcal{T}_{res}\}_{\mathsf{ssk_{TSP}}} := \mathsf{sign}(\mathcal{T}_{res}, \mathsf{ssk_{TSP}})$$

The mobile application stores those values directly on the device's main OS after verifying the message's signature.

6.1.3 EMV-compliant token-based payment

If the user has tokens stored on his mobile device, they can initiate an EMV-compliant payment process described in Fig.6.2.

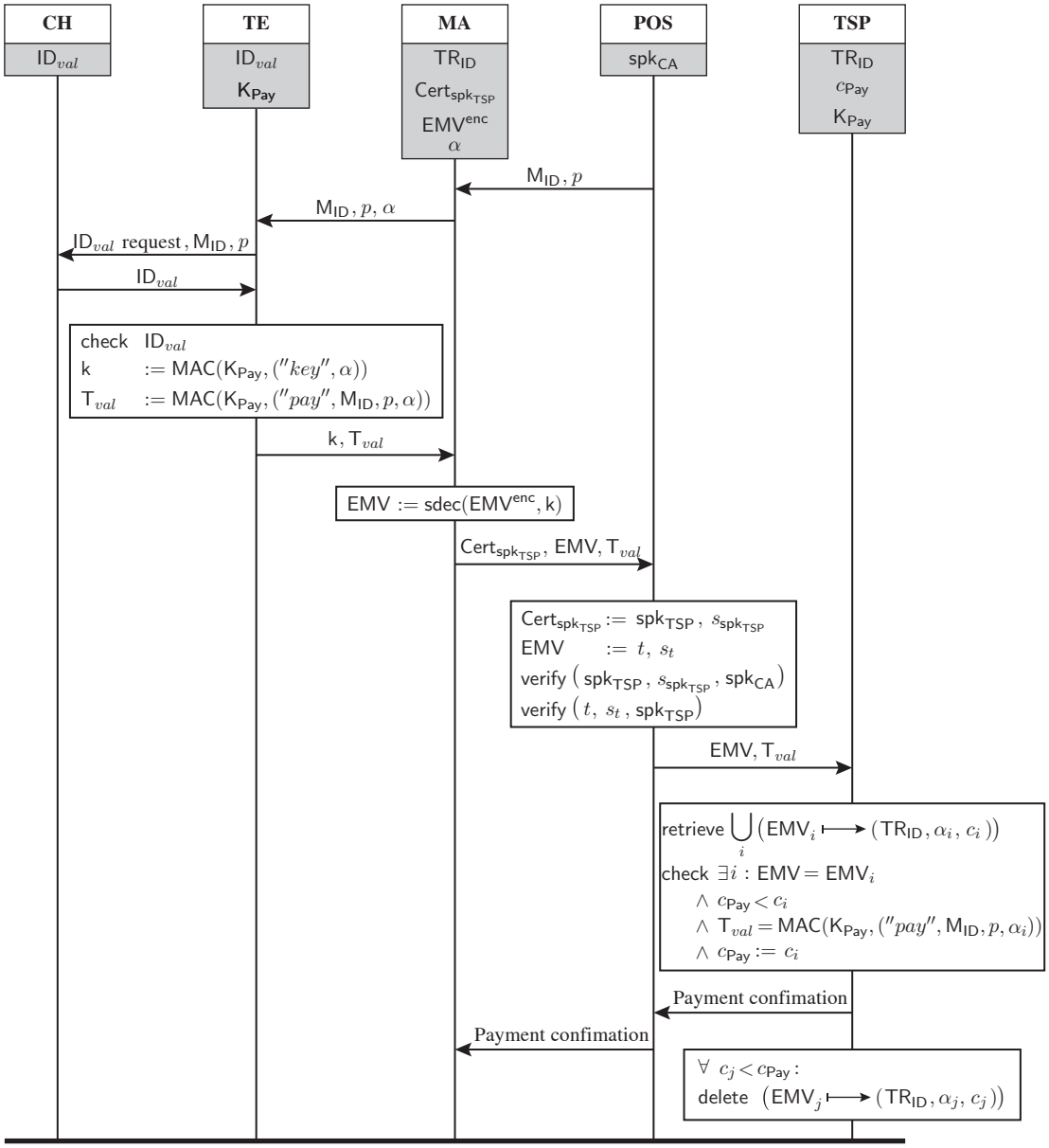


Figure 6.2: Token Payment process

At the beginning of the payment process, the merchant POS first has to transmit the merchant identifier (M_{ID}) - which could for example be an merchant identification value to its acquirer - along the transaction amount (p). This step is described as an optional step in the EMV specification [51], we define it as mandatory exchange for our protocol.

The mobile application chooses the oldest encrypted EMV packet (EMV^{enc}) it holds and transmits the associated nonce (α) along the transaction parameters to the TE. Similarly to the Token Provisioning process, an identification process of the user is required before authorizing the TE to process anything.

The TE displays the transaction amount (p) and the merchant identifier $\mathsf{M_{ID}}$ through a secure interface to the user. If they agree with the transaction parameters, they identify themself with ID_{val}.

Once this identification is successful, the TE computes two values: the encryption key (k) and the transaction fingerprint (T_{val}).

$$\mathsf{k} := \mathsf{MAC}(\mathsf{K_{Pay}}, (''key'', \alpha))$$
$$\mathsf{T}_{val} := \mathsf{MAC}(\mathsf{K_{Pay}}, (''pay'', \mathsf{M_{ID}}, p, \alpha))$$

It transmits those values to the mobile application which is then able to retrieve the ready-to-use EMV payment packet:

$$\mathsf{EMV} := \mathsf{sdec}(\mathsf{EMV}^{\mathsf{enc}}, \mathsf{k})$$

The mobile device can then process to an EMV-SDA-compliant process[20]. It sends the TSP signature verification key certified by a certification authority ($\mathsf{Cert}_{\mathsf{spk_{TSP}}}$), the tokenised EMV packet (EMV) and the transaction fingerprint (T_{val}). After processing the usual EMV signatures verification the POS sends the EMV packet and the transaction fingerprint along its merchant ID ($\mathsf{M_{ID}}$) and the transaction amount (p) to the TSP.

The TSP checks if the token EMV is indeed part of its token vault and if so, retrieves the user ID linked to it. This means that there exists i such that $\mathsf{EMV} = \mathsf{EMV}_i$. The TSP also retrieves the token's associated nonce (α_i) and counter (c_i). It proceeds to the verification of the transaction fingerprint by recomputing $\mathsf{MAC}(\mathsf{K_{Pay}}, (''pay'', \mathsf{M_{ID}}, p, \alpha))$ and comparing it with T_{val}. In addition to the transaction signature verification, the TSP deactivates old tokens: EMV_i is associated to a counter c_i and the TSP holds another counter c_{Pay}. A payment will be validated only if the actual value of c_i is higher than the one of c_{Pay}. c_{Pay} is the counter value of the latest token validated by the TSP for a payment. If the token is too old, the payment will be refused. Otherwise the payment is authorized and the corresponding notification is sent to the POS. The TSP then updates the c_{Pay} value to c_i and deletes (or deactivates) the tokenised EMV packet from its database, as well as the older ones, preventing them to be used for a further payment.

The mobile application can also erase EMV_i, or at least deactivate it.

6.2 Trust Assumptions

Fig.6.3 summarizes the security assumptions we made for our protocol. In essence, from the user perspective, we claim that our protocol is secure event when run with a corrupted mobile device. Yet we made a technically strong assumption about the TEE: it shall indeed do what is expected of it -restrict access control to the SE with a trusted user interface. Although it may, at the time, be a challenging technical task, we hope that in the following years, standard-compatible TEE will be developed [64].

This section presents our threat and communication model and all security assumptions that were made when the protocol was designed.

6.2.1 Threat Model

Honest entities

Since we do not have any influence over the traditional payment ecosystem - acquirer, issuer, payment network - we consider it as one abstract honest entity bound to the *token service provider*. More particularly, it securely stores all the information it manages and cannot be impersonated.

[20] See Fig.5.2.

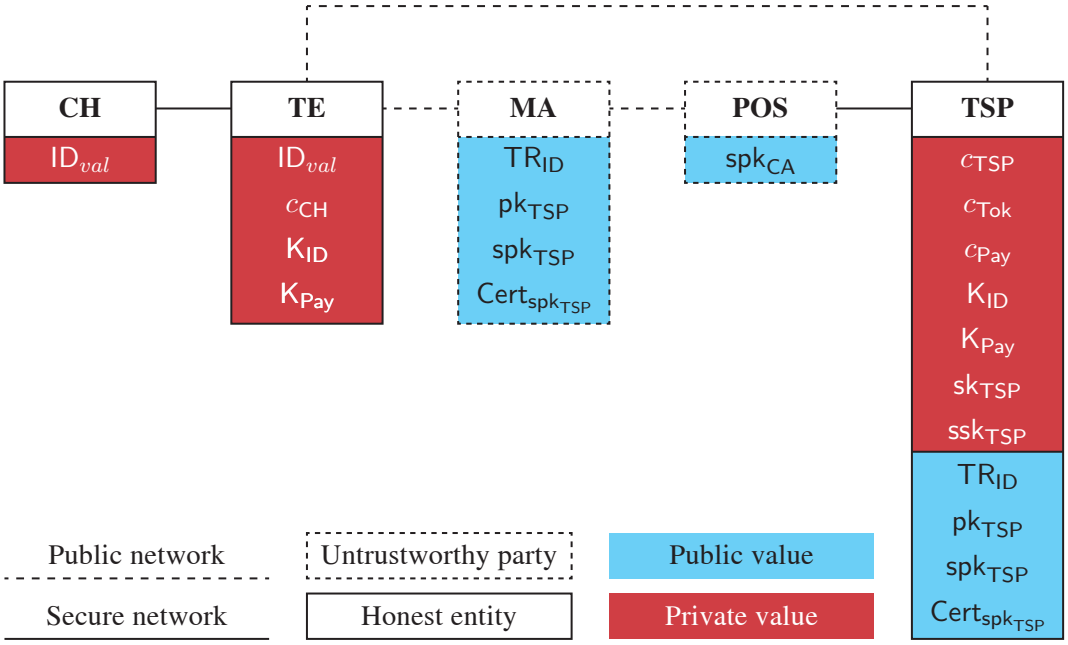


Figure 6.3: Communication and threat model

The *cardholder* is considered as always honest for as soon as they give their identification value (ID_{val}), an attacker could use it to pay from the user's mobile device.

The *trusted enclave* is also considered as a honest entity: it displays the right values to the user during a transaction, securely identifies them and process the computation of critical values - H_{ID}, T_{val}, token decryption keys - only after a rightful identification of the user

A *honest merchant point of sale* behaves as specified in our protocol.

Untrustworthy parties

The *mobile application* is considered under the attacker's control and outputs all knowledge it holds or is given.

A *rogue merchant point of sale* shares its data with the attacker and arbitrarily behaves. For example, it could try to input any payment value it receives from the attacker without processing the EMV verification.

Leaked values

We added three more corruption cases by leaking some critical values: the user's identification value (ID_{val}) and the symmetric keys (K_{ID} and K_{Pay}).

6.2.2 Communication Model

We summarized the interactions between our protocol entities in Fig.6.3.

Secure channels

We consider there is a trusted path *between a user and the trusted enclave* by their mobile device as well as inside the TE itself (between the SE and the TUI).

The channel *between the merchant point of sale* - rogue or not - *and the token service provider* is also a secure authenticated channel. Whether it is honest or compromised, the POS is correctly authenticated to the TSP and the attacker has no control over the payment network through which the communication between the POS and the TSP happens.

Compromised channels

We assume that the attacker has control over all public networks: the network holding the communication *between the TSP and the mobile application*, the one *between the mobile application and the POS* and the one *between the trusted enclave and the mobile application*.

By "control", we mean that an attacker can eavesdrop information and interact with all entities with values obtained or built from the public network, which is the classic Dolev-Yao adversary behaviour[21].

6.3 Security Claims

We list here the six main properties guaranteed by our protocol. The two first ones form the natural agreement properties: whenever a transaction is made, the cardholder did consent to it and similarly, whenever a merchant is notified a transaction is valid, it is guaranteed to be paid. The third property states similarly an agreement for the provisioning part: whenever a token is provisioned, the cardholder did initiate the request. The following two properties are stronger security guaranties offered by our system, in case a decrypted token is leaked. Finally the last property ensures some privacy is guaranteed for the cardholder. A merchant (or an attacker) cannot track a client's consumption habits.

6.3.1 Mandatory transaction agreement by the user

Each time the TSP validates a transaction between a cardholder CH and a merchant M for an amount p, then CH must have initiated the payment request (thus agreed to it) of amount p to the merchant M by authenticating themself through the TE.

6.3.2 Merchant payment assurance

Whenever a merchant M is notified that a transaction of amount p has been validated, the TSP validated the payment of amount p to the merchant M and sent the notification.

6.3.3 Injective Token Provisioning

Each time a token is generated by the TSP for a given cardholder CH, a given key $\mathsf{K_{ID}}$ and a given counter value c_{CH}, then CH must have initiated the provisioning request by authenticating themself through their mobile (on which the SE holds $\mathsf{K_{ID}}$). Moreover, the TSP did not already generate a token for the same counter value.

[21] See section 1.3.

6.3.4 Injective token-based payment

A token can only be used once. If a TSP validates a transaction for a specific token T owned by a client, then such a token T has not been previously used for a payment which was validated by the TSP. Interestingly, this property prevents the replay attacks against the EMV-SDA protocol exposed in section 5.1.1.

6.3.5 Token stealing window

In case a token is stolen (for example by a merchant which maliciously claims that the transaction failed) then the consequences are mitigated by the two following properties.

- The token may be used only until a new transaction is made by CH. In other words, tokens have a limited validity.
- The stolen token may only be used for the merchant and the amount on which the user agreed upon. (Therefore a double payment can be traced.)

6.3.6 Client payment unlinkability

An attacker cannot tell whether two payment sessions are processed by the same client or not. Thus a client's consumption habits cannot be tracked by a merchant or an attacker.

6.4 A Practical Solution

We argued our protocol provided a specification for a mobile payment application which would be easier to implement, therefore reducing the production costs, and was also a practical solution on the industrial part, by being highly scalable, and on the user's part, by providing satisfying performances.

So we implemented a mobile application prototype of our protocol. To compare performances, we implemented two versions of our protocol: one where the whole process runs as a software mobile application and where the Secure Elements operations are emulated by the mobile application. The second version actually implements our design, where the Secure Element operations are processed by a SIM card.

The prototype was tested on a Samsung Galaxy S6 with a 2.1GHz CPU and 3GB RAM. We used a SIM card provided by the Orange company. Since the enrollment part is out of scope of this PhD, all keys and counters are already provisioned to both the application and the Secure Element as hard-coded data. The TSP as well as the POS are both emulated by a PC. We implemented the TSP as prescribed by our specification.

For our TSP's verification signing key to be recognized by a real-life POS, we would have needed an agreement with a payment network and a certification authority, so basically an agreement with a payment network to generate our own signed tokens for testing and adding our TSP keys inside the kernel of a POS, which was not going to happen. This explains why we chose to emulate the POS. The emulated POS conforms with the standards required by the payment standards [108, 51] and performs an EMV-SDA transaction process. Our code use either Java (to emulate the TSP and POS), JavaCard (SIM card) or Android (main mobile application).

We chose the PIN as a method for user identification but other methods could be used, nowadays market for identification methods through a mobile device is actually blossoming. The transaction between the POS and the mobile application is performed through NFC.

	Full software implementation	Software and secure element implementation
Token provisioning		
Token request before PIN validation	40.5 ms	40.5 ms
Token request after PIN validation	24.3 ms	26.6 ms
Server answer processing	9.05 ms	9.05 ms
EMV-SDA transaction		
Payment initiation before PIN validation	209 ms	209 ms
Tokenized EMV transaction	39.0 ms	41.35 ms
Individual MAC processing		
H_{ID}	0.2 ms	2.5 ms
k and T_{val}	0.25 ms	2.6 ms

Table 6.1: Performances of our protocol implementation (average results over 50 measurements)

We can assess that from a user perspective, there is no real difference between a full software implementation and our solution that involves a secure element (columns 2 and 3 of Table 6.1). Indeed, the Secure Element can also compute a MAC in less than 1 millisecond and the round trip time between the Secure Element and the mobile application is less than 2 milliseconds. Therefore, the overhead of our protocol is negligible from a user's perspective and even our early implementation seems practical.

Conclusion and discussion

Regarding the restrictions stated in previous chapter about the payment industry, we proposed a payment protocol specially designed for mobile devices and relying on tokenisation. Although we still use a secure element for security, the major part of our solution can be implemented as an HCE application thus reducing development costs and certification process. The functions implemented on the secure element are basic ones - MAC computation and counter management - so they are easier to certificate than a whole EMV application. Moreover, since those functions are very generic, they could be used by other applications requiring these kind of computations modulo a key management policy.

To prove our point in terms of practicality of the protocol, we implemented a prototype of a mobile application implementing our protocol. The performances of such an application are quite similar to a full HCE solution from the user point of view.

We claim our protocol actually guarantees the same security as EMV-SDA protocol. In fact, it does even more, since it guarantees a protection against replay attacks EMV-SDA is vulnerable to. Moreover, our protocol adds some kind of privacy for the user from the merchant point of view.

Yet, the security claims we made relied on one strong technical assumption. We considered a trusted path between the secure element and the user through the use of of a trusted execution environment that can handle a trusted user interface, as specified in [64].

Achieving such a security measure is indeed a technical challenge. Nonetheless, we did not aim at addressing this issue during this thesis although we did explore a bit what could be done regarding the secure identification of a user with their mobile device, there shall be a more extensive research about it.

Another assumption we made was considering that the payment network-related entities (banks) operated in a secure network and were considered as honest entities. However, we do not specifically know what kind of additional security information each bank would require to validate a payment or how is the risk management handled for each actor of the industry. Those factors cannot appear in our specification so one could argue that our protocol is a rather high level design.

Still, we tried our best to be compatible with the public specifications that are available regarding payment protocol and design requirements. As for our prototype, since we did not have a partnership with a bank, we had to develop our own point of sale tester and make a high level simulation of the payment network. It shall still be noted that we ensured to be compatible with the existing public point of sale specification [51]. In fact, our point of sale emulator can indeed communicate with EMV-compatible contactless cards although we cannot process a whole transaction with them.

We proved the security of our protocol using the Tamarin prover. This work is described in the next - and final - chapter of this part on payment protocols.

Chapter 7

A Formal Analysis of our Payment Protocol

Now the design of our payment protocol has been explained, we need to prove its security. As stated before, several automated tools exist to analyze and prove the security of cryptographic protocols and only some of them can do it for an unbounded number of sessions. ProVerif is one of them.

Unfortunately, ProVerif does not handle mutable states and our protocol security strongly relies on the use of counters. This excludes ProVerif for our security proof. Luckily, the Tamarin prover does handle mutable states as well as proof of trace and diff-equivalence properties for an unbounded number of session. That is why we chose it to perform our security proof. As a reminder, we have six main properties to prove:

- Five trace properties:

  - *Mandatory transaction agreement*: for a payment to be processed, it has to be previously approved by a user.

- *Merchant payment assurance*: ensuring that of the merchant is convinced the payment has been approved, it will indeed get paid.
- *Injective token provisioning*: the TSP will provision the token to the mobile device with some limitations set by a counter value.
- *Injective token payment*: a token can only be used for one payment at most.
- *Limited token stealing window*: in case a token is stolen, we limit its validity in time, thanks to a counter, and by dynamically linking it to a specific merchant and price.

• One equivalence property: the unlinkability of a payment, guaranteeing that a user cannot be tracked by a merchant thanks to their data.

This final chapter provides the security proof of our protocol specified as a Tamarin model. We first set the ground knowledge about the Tamarin prover: what its syntax is and how to specify a protocol in its calculus. We then model our protocol and formalize our properties in the Tamarin calculus. Finally, we give the reader the conclusion of our security analysis and compare our protocol with currently known EMV attacks

All proof files are available at [35].

7.1 Tamarin prover

The Tamarin prover is a security protocol verification tool that can handle an unbounded number of sessions to prove trace and equivalence properties. Its syntax is actually close to ProVerif's. We will here provide an overview of its syntax without going into to much details about its semantics, so the reader can understand how protocol specification is done using ProVerif.

Another interesting aspect of the tool is its interactive mode that allows the user to manually -and machine-checked - prove properties. This broadens the set of provable properties and is actually quite useful when dealing with counter-related properties as we are going to see in the following sections.

7.1.1 Message theory

The message theory Tamarin relies on, is actually quite similar to the one presented in sections 1.1.1 and 1.1.3. Like in ProVerif, and as usually done in formal models, cryptographic messages are represented by terms. However, unlike ProVerif, Tamarin does not type its terms. Cryptographic primitives' properties, on the other hand, are exclusively defined by an equational theory.

Terms

Cryptographic messages are represented by terms, elements an order-sorted algebra with the top sort *msg* (message) and the two incomparable subsorts *fr* (fresh) and *pub* (public).

We assume, for each subsort, an infinite set of names ($\mathcal{N}_{fr}$ and $\mathcal{N}_{pub}$) and an infinite set of variables ($\mathcal{V}_{fr}$ and $\mathcal{V}_{pub}$). Fresh names typically model cryptographic material initially unknown to the attacker such as keys or nonces while public names represent public values known by the adversary.Let $\mathcal{N} = \mathcal{N}_{fr} \sqcup \mathcal{N}_{pub}$ be the set of names and $\mathcal{V} = \mathcal{V}_{fr} \sqcup \mathcal{V}_{pub}$ the set of variables.

We also assume a signature, Σ: a finite set of function symbols declared with their arity.

Example 14 (signature of our payment protocol). Following the description of our payment protocol given in section 6.1, we need to define a signature that will allow us to model: hashing (MAC_1 and MAC_2), asymmetric encryption (pk, aenc and adec), symmetric encryption (senc and sdec) and signature (spk, sign, verify and true).

We propose the following signature:

$$\Sigma = \left\{ \begin{array}{l} \mathsf{MAC}_1/2,\ \mathsf{MAC}_2/4, \\ \mathsf{pk}/1,\ \mathsf{aenc}/2,\ \mathsf{adec}/2, \\ \mathsf{senc}/2,\ \mathsf{sdec}/2, \\ \mathsf{spk}/1,\ \mathsf{sign}/2,\ \mathsf{verify}/3,\ \mathsf{true}/0 \end{array} \right\}$$

Terms are built over names, variables and function symbols and the set of terms is denoted $\mathcal{T}(\Sigma, \mathcal{N}, \mathcal{V})$.

Equational theory

The properties of cryptographic primitives are modeled through an equational theory $\mathcal{E}$, a finite set of equations: $M = N$ with $M,\ N \in \mathcal{T}(\Sigma, \mathcal{N}, \mathcal{V})$. We define the equality modulo $\mathcal{E}$ as the binary relation $=_\mathcal{E}$ on terms such that $=_\mathcal{E}$ is the smallest equivalence relation containing $\mathcal{E}$ that is closed under the application of the function symbols in Σ, the bijective renaming of names and the substitution of variables by same sorted terms.

Example 15 (equational theory of our payment protocol). Using the signature provided in Example 14, we need to equip our term algebra with an equational theory describing asymmetric decryption, symmetric decryption and signature verification.

Actually, Tamarin's internal libraries already propose an equations for those cryptographic schemes. Hence, relying on those equations, our equational theory would be:

$$\mathcal{E} = \left\{ \begin{array}{ll} \mathsf{adec}(\mathsf{aenc}(m, \mathsf{pk}(s)), s) & = m \\ \mathsf{sdec}(\mathsf{senc}(m, s), s) & = m \\ \mathsf{verify}(m, \mathsf{sign}(m, s), \mathsf{spk}(s)) & = \mathsf{true} \end{array} \right\}$$

7.1.2 Protocol representation by state transition systems

Tamarin uses transition systems between *states* to represent protocols. It essentially means that instead of representing protocols by several processes as it is done in ProVerif, we represent them as a set of transitions that define how the protocol progresses from one state to another. Each state contains the attacker knowledge, the messages sent or received from the network, information about freshly generated values and protocol-specific states. Those information are represented by *facts*.

As a simple and informal example one could think of how to represent the incrementation of a counter c after the reception of a message m on a private network with a transition between two states S_0 and S_1: S_i would contain the counter c and a trigger *private reception of*(m) (which would both be represented by facts) and S_{i+1} would contain the counter $c + 1$.

Facts and states

Formally, in addition to the signature Σ, we assume the disjoint unsorted signature $\Sigma_\mathcal{F}$ of fact symbols. We define the set of facts $\mathcal{F}$ as:

$$\mathcal{F} = \left\{ \mathrm{F}(t_1, ..., t_n) \mid \mathrm{F} \in \Sigma_\mathcal{F}\ ;\ t_i \in \mathcal{T}(\Sigma, \mathcal{N}, \mathcal{V}),\ \forall\, i \in [1, n] \right\}$$

$\mathcal{F}$ is divided into two disjoint subsets: *linear* facts and *persistent* facts. Linear facts model resources that can only be consumed once - like, for instance, a counter incrementation or a message sent over a

private network - whereas persistent facts model resources that can be consumed arbitrarily often - like the set of cryptographic keys associated to an agent of the protocol - they are preceded by the symbol !.

Four fact symbols are restricted, they are used to model the interaction with an untrusted network and to model the generation of fresh values:

- $\mathsf{Fr}(r)$: is used when a fresh value $r \in$ *fr* is generated. Tamarin internally ensures that each instanciation of r is different from all others and of sort *fr*.
- $\mathsf{In}(m)$: states that a message m is received from an untrusted network.
- $\mathsf{Out}(m)$: indicates that a message m is output on an untrusted network.
- $\mathsf{K}(t)$: expresses that the attacker knows t.

A state is simply represented by a *multiset of facts*. We design by $vars(s)$ the variables of the state s.

Rules, protocols and adversary

Tamarin represents the transition between states by *labeled multiset rewriting rules*.

Formally, we have the following definitions, that were originally proposed in [14, 104, 87]:

Definition 4. Let p, a and c be three multisets of facts (which are elements of F): $p - [a] \rightarrow c$ is a *labeled multiset rewriting rule*. We call p the *premises*, a the *actions* and c the *conclusions* of the rule.

Intuitively, the premises and conclusions of the rule represents two states of the protocol and the rule defines the progression from the premises to the conclusions. Whenever the facts of the premises are present, the facts of the conclusions are produced. The rule's action is used to either *restrict* this progression - for instance, a transition that could only happen when two terms are equal - or to trigger an *event* represented by a fact. *Restrictions* are user-defined, we will give more details about their syntax further on.

There are three kinds of rules: fresh name generation, message deduction rules and protocol rules.

The fresh generation rules allow the creation of an arbitrary number of fresh names.

Definition 5. *Fresh name generation* rules are of the form: $\varnothing - [\,] \rightarrow \mathsf{Fr}(r)$ with $r \in \mathcal{N}_{fr}$.

Like ProVerif, Tamarin's attacker model is a Dolev-Yao adversary [47], an attacker that can eavesdrop, intercept, compute and send all messages passing over the public network. The message deduction rules model the attacker capabilities.

Definition 6. We say that $p - [a] \rightarrow c$ is a *message deduction* rule if it is of either one of the following forms:

1. $\mathsf{Fr}(r) - [\,] \rightarrow \mathsf{K}(r)$, with $r \in \mathcal{N}_{fr}$
2. $\varnothing - [\,] \rightarrow \mathsf{K}(n)$, with $n \in \mathcal{N}_{pub}$
3. $\mathsf{K}(t) - [\mathsf{K}(t)] \rightarrow \mathsf{In}(t)$, with $t \in \mathcal{T}(\Sigma, \mathcal{N}, \mathcal{V})$
4. $\mathsf{Out}(t) - [\,] \rightarrow \mathsf{K}(t)$, with $t \in \mathcal{T}(\Sigma, \mathcal{N}, \mathcal{V})$
5. $\{\{\mathsf{K}(t_1), ..., \mathsf{K}(t_n)\}\} - [\,] \rightarrow \mathsf{K}(f(t_1, ..., t_n))$, with $f/n \in \Sigma$ and $t, t_1, ..., t_n \in \mathcal{T}(\Sigma, \mathcal{N}, \mathcal{V})$

Rule 1 states that the adversary can generate fresh values. Rule 2 that it knows public values. Rule 3 and 4 allow the adversary to input and output messages to the protocol, the action in rule 3 will make the messages sent by the adversary observable in a protocol's trace. Finally, rule 5 allow the adversary to compute messages from known ones with the function symbols from Σ.

Finally, the protocol rules are user-defined and must satisfy some conditions.

Definition 7. We say that $p - [a] \rightarrow c$ is a *protocol rule* if:

1. $vars(c) \subset vars(p) \cup \mathcal{V}_{pub}$
2. p, a and c do not contain fresh names.
3. p does not contains Out and K facts.
4. c does not contains In, Fr and K facts.

Condition 1 bounds the variables that occur in conclusions in premises (except for public variables). Condition 2 and 4 ensure that fresh name creation is respected for fresh names cannot occur in rules and Fr can only be used in premises. Conditions 3 and 4 is here to guarantee that the semantics of In, Out and K are respected. Protocol facts usage is unrestricted. A protocol is defined as a *finite set of protocol rules*.

Example 16 (Token Request processing). Referring to the token provisioning protocol described in section 6.1.2 we can write the following rule to model the TSP behavior when answering a provisioning request:

$$
\begin{array}{ll}
\mathsf{let} & < \mathsf{TR_{ID}},\ < c_{\mathsf{TE}}, \mathsf{H_{ID}} > > = \mathsf{adec}(\mathcal{M}_{tok}, \mathsf{sk_{TSP}}) \\
& \mathsf{EMV} \quad = < t, \mathsf{sign}(t, \mathsf{ssk_{TSP}}) > \\
& \mathsf{EMV^{enc}} = \mathsf{senc}(\mathsf{EMV}, \mathsf{MAC}(\mathsf{K_{Pay}}, \alpha)) \\
& \mathcal{T}_{res} \quad = < \mathsf{TR_{ID}}, \mathsf{EMV^{enc}} > \\
\mathsf{in} &
\end{array}
$$

$$
\left[\begin{array}{l}
\mathsf{In}(\mathcal{M}_{tok}) \\
!\ \mathrm{TSP\_private\_keys\_set}(TSP, \mathsf{sk_{TSP}}, \mathsf{ssk_{TSP}}), \\
!\ \mathrm{TSP\_user\_data}(SE, \mathsf{TR_{ID}}, \mathsf{K_{ID}}, \mathsf{K_{Pay}}), \\
\mathrm{TSP\_Counter}(\mathsf{TR_{ID}}, TE, TSP, c_{\mathsf{TSP}}), \\
\mathsf{Fr}(\alpha),\ \mathsf{Fr}(t)
\end{array}\right]
$$

$$
-\left[\begin{array}{l}
\mathrm{Eq}(c_{\mathsf{TE}}, c_{\mathsf{TSP}} + 1), \\
\mathrm{Eq}(\mathsf{H_{ID}}, \mathsf{MAC}(\mathsf{K_{ID}}, c_{\mathsf{TSP}} + 1))
\end{array}\right]\rightarrow
$$

$$
\left[\begin{array}{l}
\mathsf{Out}(\mathcal{T}_{res}), \\
\mathrm{Store\_token\_in\_TSP}(TSP, \mathsf{TR_{ID}}, \alpha, t), \\
\mathrm{TSP\_Counter}(\mathsf{TR_{ID}}, TE, TSP, c_{\mathsf{TSP}} + 1)
\end{array}\right]
$$

To alleviate the protocol specification, Tamarin allows the user to bind terms to a rule, which is the purpose of the let command in the beginning of the rule.

Whenever the TSP receives a request $\mathcal{M}_{tok}$, it decrypts it with its private key $\mathsf{sk_{TSP}}$ and retrieves $\mathsf{TR_{ID}}$, c_{TE} and H_{ID}. The transition rule is here defined with two *restrictions* "Eq"(that will be described in section 7.1.4). It essentially means that the transition rule will not execute unless $c_{\mathsf{TE}} = c_{\mathsf{TSP}} + 1$ and

$\mathsf{H}_{ID} = \mathsf{MAC}(\mathsf{K}_{\mathsf{ID}}, c_{\mathsf{TSP}} + 1)$. Those are the security verification processed by the TSP when receiving a Token Provisioning request.

The TSP then generates a fresh token t and its asociated nonce α and sends the response $\mathcal{T}_{res}$.

The semantics of Tamarin, which we will not depict here as we refer to [87, 85] for more detailed information, reflect the executions of rules that define the progression between states. We will only indicate that each execution of a rule is associated with a temporal variable denoted $\#i$.

Example 16 actually provides some example of counter representation. More details about that in further section 7.1.4.

7.1.3 Security properties specification

Like ProVerif, Tamarin can handle both trace and diff-equivalence properties.

Trace properties

Tamarin can handle the verification of trace properties. They are expressed as (reusable) *lemmas*, sorted first-order formulas over the four following kinds of atoms:

- F @$\#i$, with $\#i$ a temporal variable and $\mathrm{F} \in \mathcal{F}$ element of a rule's actions. It states that the fact F happens at a transition rule associated with $\#i$.
- $\#i < \#j$, where $\#i$ and $\#j$ are temporal variables. It defines an order between transition rules.
- $\#i = \#i$, $\#i$ and $\#j$ are temporal variables. It defines an equality between transition rules.
- $t = t'$, with $t,\ t' \in \mathcal{T}(\Sigma, \mathcal{N}, \mathcal{V})$. It states an equality of terms.

We provide examples of Tamarin lemmas in the next section and we refer the reader to [87] for a complete description of the syntax and the semantics of the formulas.

Diff-equivalence

Like ProVerif, Tamarin allows the proof of *diff-equivalence* properties (see section 1.3.2). Since the definition of diff-equivalence is the same as in ProVerif, we will not detail it here. As in ProVerif, the diff operator is applied on terms, examples of its usage will also be given in the next section.

For more information about how equivalence is proven in TAMARIN, we refer the reader to [14].

7.1.4 Counter representation

Our protocol security partly relies on counters. Tamarin allows us to represent them thanks to the built-in *multiset* library. Formally, a counter represented by the term c will be incremented by the operation $c+1$ as seen in Example 16. Yet, the plus sign used here is not a real addition, it indicates that the element $1 \in \mathcal{N}_{pub}$ is added to the multiset c. So our counters could be mathematically represented by a multiset of public messages: $\bigcup\{1, 1, ...1\}$.

Restrictions

As implied by their name, restrictions impose some conditions on the execution of rules. They actually follow the same syntax as restrictions given in section 7.1.3.

Using restrictions to define an equality and an order relation between counters

This representation of counters with multisets allows comparison of values. To this end, and following the idea proposed by [104] and used by SAPIC [77], we use two restrictions defining on one hand the equality between two counter values - which is used during the Token Request processing - and on the other hand an order relation between two counter values - used during the Payment Validation part, to update c_{Pay}. Both these restrictions are defined as follows:

- Equality between two counter values: $\forall\, x,\ y,\ \#i:\ \mathrm{Eq}(x,y)\ @\#i \Rightarrow x = y$
- Order relation between two counter values: $\forall\, x,\ y,\ \#i:\ \mathrm{Sm}(x,y)\ @\#i \Rightarrow \exists\, z:\ x + z = y$

7.2 Protocol model and formal properties

The security of our protocol relies on both trace and diff-equivalence properties that we need to prove. This section provides an overview of both the models produced to this purpose as well as the formal properties matching the security claims exposed in section 6.3.

7.2.1 Protocol model for trace properties

As usual, we assume the adversary fully controls public communication channels, that is, it controls the channel between the mobile platform and the merchant. In contrast, we assume that the merchant communicates with the TSP on an authenticated channel. We also assume that the TE securely communicates with the user for TEE are supposed to provide a trusted path from user to secure elements. Fig.6.3 given earlier sums this up.

The adversary initially knows all identifiers, TR_{ID}, the public keys ($\mathsf{spk}_{\mathsf{TSP}}$ and $\mathsf{pk}_{\mathsf{TSP}}$) and the counters ($c_{\mathsf{TSP}}$, c_{CH}, c_{Tok} and c_{Pay}). Of course, it also learns all keys of dishonest TEs and mobile applications.

In order to help the Tamarin tool and since we consider cardholders to be honest, we chose to merge the roles of CH and TE in our model as depicted in Fig.7.1 in order to avoid unnecessary channel modeling. Of course, corrupted TEs have been modeled by splitting the two roles again. Similarly, we consider a one-by-one token generation per token provisioning session. This is not a limitation since a TSP can answer to arbitrarily many token requests.

7.2.2 Formalizing trace properties

Section 6.3 informally laid out our security claims for our protocol. Six of this statements can be expressed by trace properties for they are basically agreement properties.

This section will expose how we formally expressed those six trace properties as lemmas in our Tamarin model. However, with only those lemmas, Tamarin could not terminate at first for the complexity of our model was too much. So in addition to the lemmas expressing our properties, we had to prove *typing lemmas*, whose usage will be described further on.

We had to label our transition rules with facts - that can be seen as events for the reader used to ProVerif's syntax - corresponding to the different states of each agent. Where these events are placed is described in Fig.7.1.

Proving invariants

Since Tamarin's terms are untyped, we sometimes need to prove invariants to help the tool terminating some proofs. Invariants can be proven using *typing lemmas*, a special type of lemmas that can be re-used

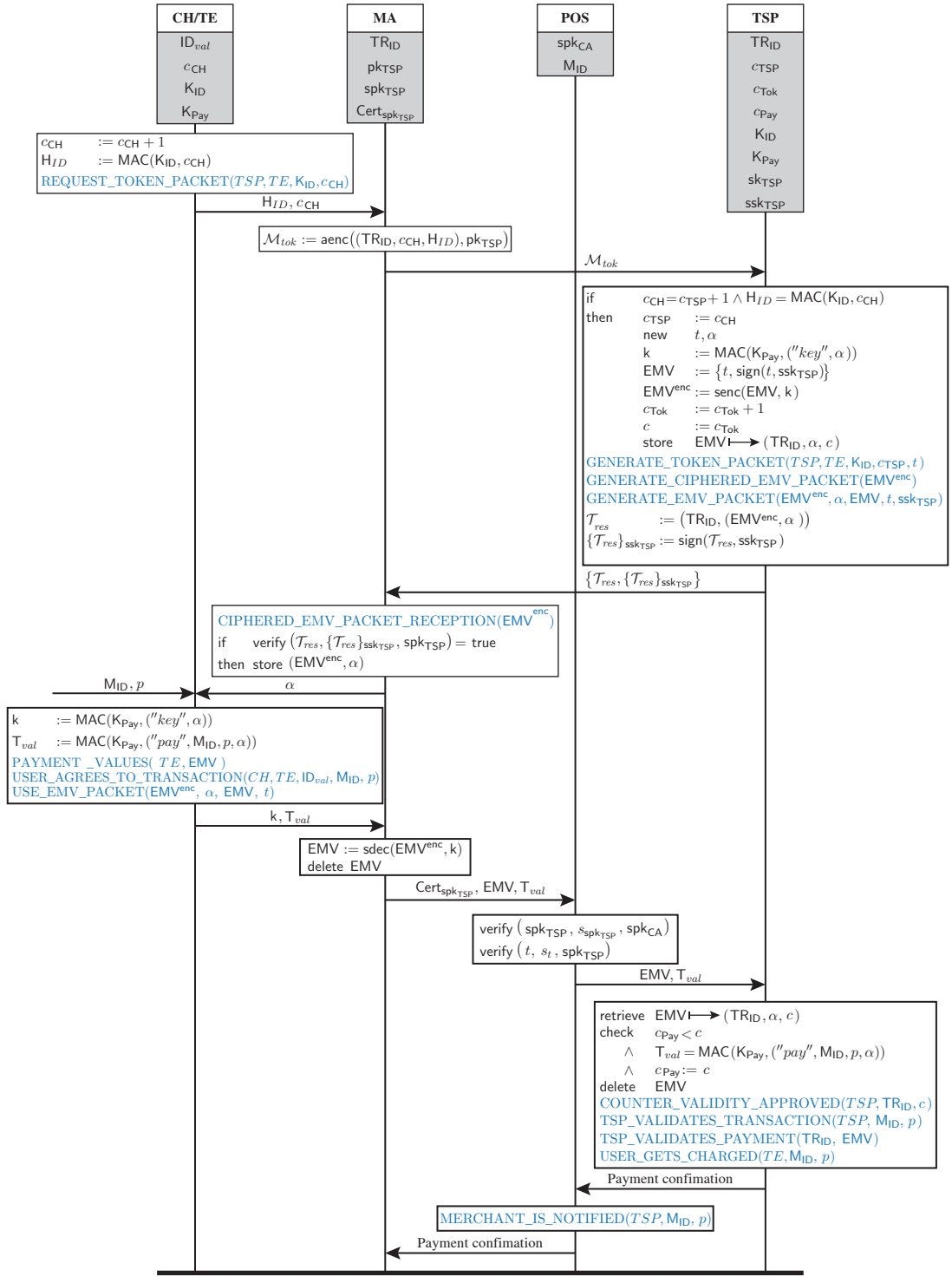


Figure 7.1: Our Tamarin model

by the tool to prove other ones. One of the task for the user using Tamarin is to find out what invariants need be proven.

In our proofs, we needed to prove two typing lemmas, otherwise, our main security properties could not be run.

The first one states that whenever an encrypted token is received by a mobile device as a response to a token request, it has previously been generated by a TSP.

$$\begin{aligned}
&\forall\ \mathsf{EMV}^{\mathsf{enc}},\ \#i: \\
&\text{CIPHERED_EMV_PACKET_RECEPTION}(\mathsf{EMV}^{\mathsf{enc}})\ @\#i \\
&\Rightarrow \exists\ \#j: \text{GENERATE_CIPHERED_EMV_PACKET}(\mathsf{EMV}^{\mathsf{enc}})\ @\#j
\end{aligned}$$

The formula states that whenever the fact CIPHERED_EMV_PACKET_RECEPTION is triggered for the value $\mathsf{EMV}^{\mathsf{enc}}$, there is another transition rule during which the fact GENERATE_CIPHERED_EMV_PACKET was triggered for the same value - actually during the TSP token request processing.

The second typing lemma yields four more invariants.

$$\begin{aligned}
&\forall\ \mathsf{EMV}^{\mathsf{enc}},\ \alpha,\ \mathsf{EMV},\ s,\ t,\ \#i: \\
&\text{USE_EMV_PACKET}(\mathsf{EMV}^{\mathsf{enc}},\ \alpha,\ \mathsf{EMV},\ s,\ t)\ @\#i \\
&\Rightarrow \exists\ \mathsf{ssk}_{\mathsf{TSP}},\ \#j: \quad \text{GENERATE_EMV_PACKET}(\mathsf{EMV}^{\mathsf{enc}}, \alpha, \mathsf{EMV}, s, t, \mathsf{ssk}_{\mathsf{TSP}})\ @\#j \\
&\qquad\qquad \wedge\ \#j < \#i \\
&\qquad\qquad \wedge\ s = \mathsf{sign}(t, \mathsf{ssk}_{\mathsf{TSP}})
\end{aligned}$$

Whenever $\mathsf{EMV}^{\mathsf{enc}}$, α, EMV, s and t are used in the transition rue initiating the payment on part of the mobile application, they have previously been generated by the TSP (holding $\mathsf{ssk}_{\mathsf{TSP}}$) during the token request processing. Moreover, the typing lemma states that s is the signature of the token with $\mathsf{ssk}_{\mathsf{TSP}}$.

Both these typing lemmas are necessary for Tamarin to prove each one of the trace properties matching our security claims from section 6.3.

Mandatory transaction agreement by the user

Whenever the user is charged by the TSP, they must have previously agreed on the transaction. This informal property is expressed by the following lemma in our model:

$$\begin{aligned}
&\forall\ TE,\ M,\ p,\ \#i: \\
&\quad \text{USER_GETS_CHARGED}(TE, M, p)\ @\#i \\
&\wedge\ \neg\ (\exists\ \#r:\ \text{CORRUPTED_Kpay}(TE)\ @\#r) \\
&\Rightarrow \exists\ CH,\ \mathsf{ID}_{val},\ \#j: \quad \text{USER_AGREES_TO_TRANSACTION}(CH, TE, \mathsf{ID}_{val}, M, p)\ @\#j \\
&\qquad\qquad \wedge\ \#j < \#i
\end{aligned}$$

The formula expresses that whenever a user of a mobile device holding an honest TE - such that the payment symmetric key $\mathsf{K}_{\mathsf{Pay}}$ has not been corrupted - is charged for an amount p to pay for a merchant M at a state $\#i$ (fact USER_GETS_CHARGED) then they have previously agreed to pay the merchant M the price p by identifying themself through the device holding the TE with their ID_{val} fact USER_AGREES_TO_TRANSACTION).

Merchant payment assurance

If the merchant is notified that the transaction succeeded, they are guaranteed to be paid.

$$
\begin{aligned}
&\forall\ TSP,\ M,\ p,\ \#i: \\
&\text{MERCHANT\_IS\_NOTIFIED}(TSP, M, p)\ @\#i \\
&\Rightarrow \exists\ \#j: \quad \text{TSP\_VALIDATES\_TRANSACTION}(TSP, M, p)\ @\#j \\
&\qquad\qquad \wedge\ \#j < \#i
\end{aligned}
$$

Intuitively, with this lemma we state that whenever a merchant M is notified by the TSP that they will receive the amount p during at the end of the transaction occurring at the state $\#i$ (fact MERCHANT_IS_NOTIFIED) then, at a previous state $\#j$, the TSP validated the transaction (fact TSP_VALIDATES_TRANSACTION) thus guaranteeing the merchant will indeed receive the money.

Injective token provisioning

Each time a token is generated by the TSP, there is a (unique) corresponding request by a user. This is formalized by the lemma:

$$
\begin{aligned}
&\forall\ TSP,\ TE,\ \mathsf{K_{ID}},\ c,\ t_1,\ \#i: \\
&\quad \text{GENERATE\_TOKEN\_PACKET}(TSP, TE, \mathsf{K_{ID}}, c, t_1)\ @\#i \\
&\wedge\ \neg\ (\exists\ \#r:\ \text{CORRUPTED\_Kid}(TE)\ @\#r\ ) \\
&\Rightarrow \exists\ \#j: \quad \text{REQUEST\_TOKEN\_PACKET}(TSP, TE, \mathsf{K_{ID}}, c)\ @\#j \\
&\qquad\qquad \wedge\ \#j < \#i \\
&\qquad\qquad \wedge\ \neg \left(\begin{array}{ll} \exists t_2,\ \#r: & \text{GENERATE\_TOKEN\_PACKET}(TSP, TE, \mathsf{K_{ID}}, c, t_2)\ @\#r \\ & \wedge\ \neg\ (\ \#i = \#r\ ) \end{array} \right)
\end{aligned}
$$

Informally, whenever the TSP generates a token for a mobile device holding a TE (and thus, $\mathsf{K_{ID}}$) and a specific counter value c at a state $\#i$ (fact GENERATE_TOKEN_PACKET) and if the key $\mathsf{K_{ID}}$ is not corrupted, there exists a state $\#j$ previous to $\#i$ during which a token request was emitted from the mobile device holding the TE (and the $\mathsf{K_{ID}}$) for the same counter value (fact REQUEST_TOKEN_PACKET). Moreover, there is no other state $\#r \neq \#i$ such that the TSP generated a token for those values, which is modeled by the last part of the implication.

Injective token-based payment

Similarly to the previous property, this injective property states that a token owned by the cardholder can only be used once for a payment.

$$
\begin{aligned}
&\forall\ \mathsf{TR_{ID}},\ \mathsf{EMV},\ \#i: \\
&\text{TSP\_VALIDATES\_PAYMENT}(\mathsf{TR_{ID}},\ \mathsf{EMV})\ @\#i \\
&\Rightarrow\ \neg \left(\begin{array}{ll} \exists\ \#j: & \text{TSP\_VALIDATES\_PAYMENT}(\mathsf{TR_{ID}},\ \mathsf{EMV})\ @\#j \\ & \wedge\ \#j < \#i \end{array} \right)
\end{aligned}
$$

Intuitively, whenever the TSP validates a payment for the client identified by $\mathsf{TR_{ID}}$ for a specific tokenised EMV packet at a state $\#i$ (fact TSP_VALIDATES_PAYMENT), then there has not previously been such a validation from the TSP for the same tokenised EMV packet.

Token stealing window

We would like to narrow as much as possible the window during which stolen tokens may be useful to an attacker if they happen to get one. The two following properties are not as generic as the previous one which could be required to be satisfied for any tokenised payment protocol, instead, they are more specific to our design.

- **Token limited validity:** a malicious merchant could ask a client to process to payment, pretending it did not work and proceed again, in order to get one extra token that he could use later on or sell. Our protocol offers a high level of protection even in this case: not only a token can only be used by the merchant and for the amount to which the user agreed upon (see **Mandatory transaction agreement by the user**) but old tokens get deactivated as soon as a payment with a more recent token is validated by the TSP.

$$
\begin{aligned}
&\forall \;\; TSP,\ \mathsf{TR_{ID}},\ c_1,\ c_2,\ \#i : \\
&\quad \text{COUNTER\_VALIDITY\_APPROVED}(TSP, \mathsf{TR_{ID}}, c_2)\ @\#i \\
&\wedge \;\; \text{Sm}(c_1, c_2)\ @\#i \\
&\wedge \;\; \neg \left(\begin{array}{l} \exists\, \#j : \text{COUNTER\_VALIDITY\_APPROVED}(TSP, \mathsf{TR_{ID}}, c_1)\ @\#j \\ \qquad\quad \wedge\ \#j < \#i \end{array} \right) \\
&\Rightarrow \;\; \neg \left(\begin{array}{l} \exists\, \#r : \text{COUNTER\_VALIDITY\_APPROVED}(TSP, \mathsf{TR_{ID}}, c_1)\ @\#r \\ \qquad\quad \wedge\ \#i < \#r \end{array} \right)
\end{aligned}
$$

  The fact COUNTER_VALIDITY_APPROVED is triggered if the TSP validates a payment with a token associated to a counter c owned by the client $\mathsf{TR_{ID}}$. Let c_1 and c_2 be two counters associated to two tokens with $c_1 < c_2$. Let also assume that the token associated to c_1 has never been validated by the TSP for a payment. Whenever the TSP validates a payment with the token associated to the counter c_2 at a state $\#i$, then the TSP will never validate a payment for a token associated to c_1.

- **Stolen token worthlessness:** we also considered a scenario where the attacker knows a decrypted tokenised EMV-packet. We prove that, if the mobile payment symmetric key is not corrupted and if the user did not initiate the decryption of the token with their mobile device, then no payment will be made with the stolen token.

$$
\begin{aligned}
&\forall \;\; TE,\ \mathsf{EMV},\ \#i : \\
&\quad \text{CORRUPTED\_TOKEN}(TE, \mathsf{EMV})\ @\#i \\
&\wedge \;\; \neg\ (\exists\, \#r :\ \text{CORRUPTED\_Kpay}(TE)\ @\#r\ ) \\
&\wedge \;\; \neg\ (\exists\, \#r :\ \text{PAYMENT\_VALUES}(TE, \mathsf{EMV})\ @\#r\ ) \\
&\Rightarrow \;\; \neg\ (\exists\, \#j :\ \text{TSP\_VALIDATES\_PAYMENT}(t))
\end{aligned}
$$

We assume that the TE - and more specifically $\mathsf{K}_{\mathsf{Pay}}$ - has not been corrupted. We also assume that the TE has not been requested to provide the payment values associated with a token t (fact PAYMENT_VALUES). Then, even though the token t was corrupted (fact CORRUPTED_TOKEN), and is fully known by an attacker no payment based on the token t will be validated by the TSP.

7.2.3 The case of payment unlinkability

Most mobile payment protocols include the user's name on the EMV packet although this is an optional field. Such an information given as a plaintext allows any merchant to identify a user and analyze their behavior as a customer. For instance, as it is already done in most commercial websites, a merchant could target more precisely a user with specific advertisements. However, since our tokenisation protocol provides a one-time-use surrogate value for the PAN and since the user's name or any other static data is not required to be EMV-compliant, our protocol guarantees that an attacker, for instance a merchant, that has no access to the provisioning requests (assumed to happen over a secure channel) is no able to tell whether two different payment transactions were made by the same client or not. This is the definition of unlinkability, as we informally stated it in section 6.3.6.

A model for unlinkability properties

We model a "honest but curious" merchant attempting to track a user consuming habits. Unlinkability only holds provided the attacker does not have access to the token provisioning request, which is not in contradiction with the behaviour of our rogue merchant.

Therefore we consider a less powerful attacker model than for trace properties. It fully controls public channels during the provisioning step but the token provisioning is assumed to be privately processed. This can be justified by the fact that the mobile device and the TSP communicate over TLS.

For simplicity, and because the proof in Tamarin is already very complex, we considered only two payment sessions which is not a limitation as we will discuss it further on.

Unlinkability formalization

We model unlinkability in a standard way [5]. If two cardholders CH_1 and CH_2, respectively owning the device TE_1 with K_{Pay_1} and TE_2 with K_{Pay_2}, process a payment $\mathcal{P}(TE, \mathsf{K}_{Pay})$, then an observer seeing two transactions cannot tell whether they come from the same cardholder or not.

$$!\big(\mathcal{P}(TE_1, \mathsf{K}_{Pay_1}) \mid \mathcal{P}(TE_1, \mathsf{K}_{Pay_1})\big) \;\sim\; !\big(\mathcal{P}(TE_1, \mathsf{K}_{Pay_1}) \mid \mathcal{P}(TE_2, \mathsf{K}_{Pay_2})\big)$$

Using the diff operator of Tamarin for equivalence, this property can be expressed as follows:

$$\mathsf{new}(TE_1, \mathsf{K}_{Pay_1}).\mathsf{new}(TE_2, \mathsf{K}_{Pay_2}).\ \big(\ \mathcal{P}(TE_1, \mathsf{K}_{Pay_1}) \mid \mathcal{P}(\mathsf{diff}[(TE_1, \mathsf{K}_{Pay_1}), (TE_2, \mathsf{K}_{Pay_2})])\ \big)$$

7.3 Proving the security of our protocol

All formal properties described in the previous section have been proven using the Tamarin prover. However, not all of them could be proven automatically.

7.3.1 Using the interactive mode to achieve some proofs

Although Tamarin's default heuristic was quite efficient to automatically prove the majority of our security properties, two of them required the use of interactive mode.

Indeed, proving the token limited validity, a proof heavily relying on counters, could not be achieved directly by Tamarin which was stuck in a loop. In essence, when trying to prove the trace properties, the tool focused first on finding where the instanciations of fresh values (such as the tokens) came from, not using the restriction on counters at first. Using interactive mode, we could prioritize the use of those restrictions, allowing us to prove the property in 3 steps.

Proving unlinkability was however more complex than initially expected. Because the restrictions on counters were not taken into account by the default heuristic, it took us more than a thousand interaction to achieve the proof on equivalence. We believe however that in light of our proof, more powerful heuristics could be devised.

7.3.2 Results

Table 7.3.2 summarizes Tamarin results on the proofs.

Properties	Trust Assumptions		Execution time		Steps
	K_{ID}	$\mathsf{K}_{\mathsf{Pay}}$	wall clock time	total CPU time	
Transaction agreement by the user		NC	2min 51s	82min 50s	356
Merchant payment assurance			1min 02s	30min 09s	10
Injective token provisioning	NC		11min 23s	210min 01s	251
Injective token based payment			5min 58s	123min 40s	66
Token limited validity		NC	Manual proof 3 interactions needed		
Stolen token worthlessness		NC	7min 22s	209min 29s	645

NC: Not Corrupted

Table 7.1: Tamarin proof results (40 CPUs, Genuine Intel dual core 1.2 GHz, 200GB of RAM)

It shall be noted that even if the token provisioning key K_{ID} is corrupted, payment security properties hold. Symmetrically, even if the payment key $\mathsf{K}_{\mathsf{Pay}}$ is corrupted, the token provisioning property remains valid.

If ID_{val} is lost, our properties relying on user agreement do not hold - transaction agreement and stolen token worthlessness. This is why we do not considered this corruption case in our result table.

On the other hand, the merchant payment assurance as well as the injectivity of a token-based payment are satisfied no matter what. It provides the user a guarantee in case their data is stolen.

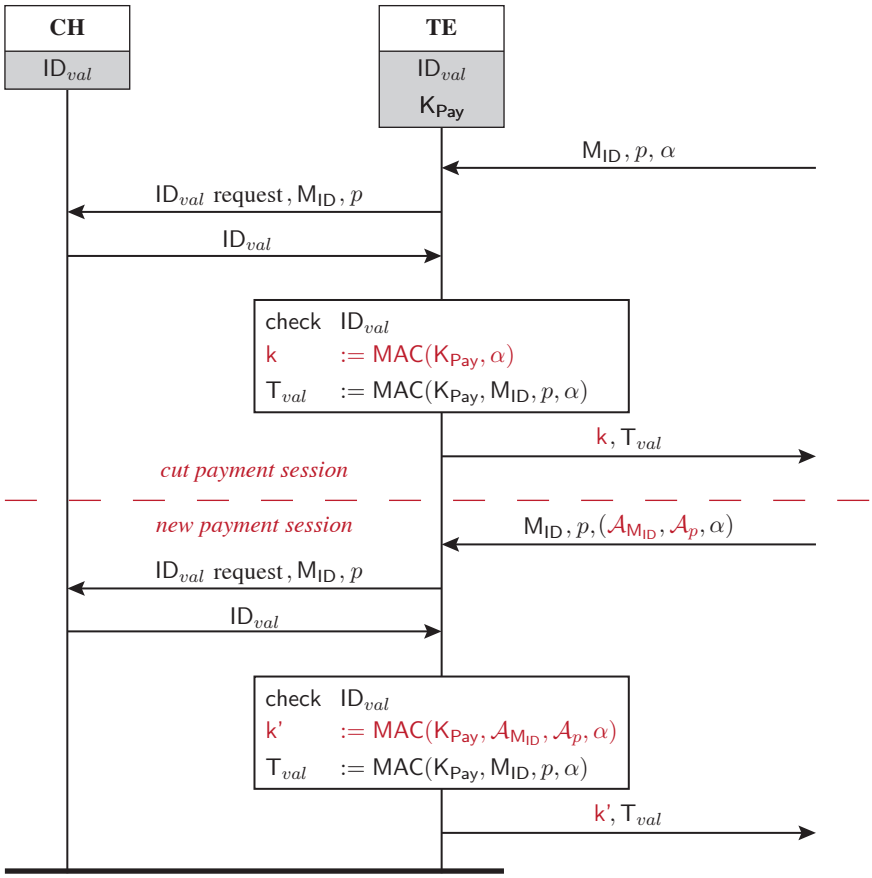


Figure 7.2: Attack on a previous version of our protocol

7.3.3 The importance of tagging

A previous version of our protocol did not rely on tags during the payment process[22]. Vincent Cheval found an attack on the *mandatory transaction agreement by the user* property in September 2017 which is described in Fig.7.2.

This attack was possible because of the lack of tags: the attacker would initiate a payment session with the right transaction parameters to get the token decryption key (k) by sending the nonce (α) associated to a tokenised EMV-packet and the right merchant identifier (M_{ID}) and transaction amount (p) so that the user would validate the transaction. Then the payment session would be interrupted and the attacker would restart a payment session by sending the message: "$\mathcal{A}_{M_{ID}}, \mathcal{A}_p, \alpha$" instead of the associated nonce as a parameter to get the decryption key and the right merchant identifier (M_{ID}) and transaction amount (p) so that the user would still validate the transaction.

The attacker would then get the token decryption key from the TE ($MAC(K_{Pay}, \alpha)$) with the first interrupted payment session and a valid transaction fingerprint ($MAC(K_{Pay}, \mathcal{A}_{M_{ID}}, \mathcal{A}_p, \alpha)$) for a merchant identifier and a price that the user did not agree to.

Thankfully, this attack was avoidable by using tags as parameters during computations that involved the symmetric key K_{Pay}. This is a usual best practice we did not think about at the time we specified our protocol.

[22] See section 6.1.3, Fig.6.2

It shall be noted that Tamarin still proved the *mandatory transaction agreement by the user* property because of an internal parsing of messages that worked to our advantage, which is why we did not detect this attack. In the original specification, the symmetric keys used in the $\mathsf{MAC}(,)$ function were put at the right side of the term. So our token symmetric key (k) and the payment fingerprint T_{val} were defined as:

$$k := \mathsf{MAC}(\alpha, \mathsf{K_{Pay}})$$
$$\mathsf{T}_{val} := \mathsf{MAC}(\mathsf{M_{ID}}, p, \alpha, \mathsf{K_{Pay}})$$

Now, the t-uples in Tamarin are parsed as pairs of pairs. So Tamarin parses the term appearing in T_{val} as:

$$\mathsf{M_{ID}}, p, \alpha, \mathsf{K_{Pay}} = (\mathsf{M_{ID}}, (p, (\alpha, \mathsf{K_{Pay}})))$$

So if the attacker sends the term "$\mathcal{A}_{\mathsf{M_{ID}}}, \mathcal{A}_p, \alpha$" instead of α to compute the second key k', Tamarin parses it as:

$$\mathcal{A}_{\mathsf{M_{ID}}}, \mathcal{A}_p, \alpha = (\mathcal{A}_{\mathsf{M_{ID}}}, (\mathcal{A}_p, \alpha))$$

Hence, we have the two non unifiable terms;

$$\mathsf{T}_{val} := \mathsf{MAC}\big(\mathsf{M_{ID}}, (p, (\alpha, \mathsf{K_{Pay}}))\big)$$
$$k' := \mathsf{MAC}\big((\mathcal{A}_{\mathsf{M_{ID}}}, (\mathcal{A}_p, \alpha), \mathsf{K_{Pay}}\big)$$

Thus making the attack impossible to detect with this model.

7.3.4 Comparing our protocol with existing EMV attacks

Because the EMV-SDA protocol relies on the use of static data, it is vulnerable to a replay attack (see Fig.5.3). If an attacker manages to get those data, it can use them on a fake card in order to process payments from the user part.

Our protocol prevents from such a replay attack. Thanks to our use of tokens as a one-time-only value - still compatible with EMV-SDA - our payment data is not static. In fact, the property *Injective token-based payment* from section 7.2.2 explicitly states that a TSP will only validate a payment with a specific token once.

Another attack on EMV [92] relies on a Man-in-the-Middle attack on Chip-and-PIN cards: a rogue card is located between the real card and the POS, the rogue card will transmit the payment data from the real card and when asked for the PIN verification, it will answer positively to any PIN introduction. However, we stated in our specification that the usage of a TEE would make it impossible to bypass the user identification, thus preventing the provisioning of payment credentials to the POS by the SE if the user identification failed. While this is a strong assumption, this requirement is part of the GSMA definition of TEE in the industry.

Conclusion and discussion

We provided a security proof of our payment protocol in the Tamarin model. After a quick presentation of Tamarin's semantics, we presented the formalized properties matching the security claims we made in the previous chapter. All proof files are available at [35].

It appeared that our protocol ensures the same security guarantees EMV data authentication protocols claim to achieve. Moreover, we also provide some privacy regarding consumption tracking by merchants for the user with our unlinkability property.

While we could argue that the knowledge about transactions is deported and managed by another entity - the token service provider - this could still be seen as a progress comparing to EMV-cards and usual mobile payment applications.

Another similarity with the EMV protocols is that the security of our payment scheme strongly relies on the correct identification of a cardholder for transaction authorization.

During a classical EMV transaction, this identification is performed either by the card - the cardholder submits their PIN code through a merchant point of sale that transmits it to their card which verifies it - either by requiring the cardholder to manually sign the transaction - usual in the US and in South America - or directly by the issuer - the PIN code is the transmitted by the merchant point of sale over the payment network.

A verification of the PIN code by the merchant did not appear as an reliable option to us for it would be a static value seen by a merchant that would impede our protocol's unlinkability. This is why we chose to make the identification *from* the device *by* the device.

We discussed on the conclusion of the previous chapter why this could be seen as technically difficult - for it would require a trusted path between the cardholder and the secure element held by their mobile device. Yet, we could also ask if banks are ready to trust devices manufacturers as well as they thrust cards manufacturers.

Speaking of trust and banks, we did not address the registration problem. While provisioning and maintaining symmetric keys on a secure element shall not be so difficult to perform for an operator - at least on the SIM - we still have the question of how to guarantee that the user registering to a mobile payment application is indeed the one authorized to use the payment card.

Indeed, most modern mobile applications allow an online registration following more or less the same process: the user would provide their original PAN along its expiry date and the CVV code, most of the time over a TLS connection. Sometimes, the user would be required to prove their identity by giving a photograph of an ID document - ID card, license, passport... - as in Orange Cash. While this kind of process is "secure enough" for the industry at the time, we could ask whether it is relevant in the case of a rogue mobile application as we consider it in our security assumptions.

Indeed, if the interface used by the user to provide their original payment data is not secure, the main point of tokenisation - which is hiding those original data from a possible malevolent device - is lost.

Another solution would be a "client present" registration process. Unfortunately, this would represent too high of a cost.

We believe this problem should be explored carefully.

Another important feature we did not consider in this thesis is the revocation problem. How does a user resign from a tokenisation service? Sure, the simple way would be for the token service provider to erase old token but how does a user make sure the revocation was indeed taken into account and on what grounds do we lay the acceptance of a revocation by the whole service.

Last but not least regarding this protocol, it should be noted that, cryptographically speaking, the MAC primitive is not the best option when it comes to key derivation. Instead, a Key Derivation Function such as HKDF should rather be used. Note that our protocol's first version considered the MAC as an abstract function for Key Derivation purposes and with Key Derivation function properties for the security analysis. An update of the protocol and the security proofs can be done as a future work.

We still believe that our tokenisation framework is abstract and siloed enough to allow the addition of such services without compromising the security. Moreover, since the token request process and the payment process are well isolated - as extrapolated from our result - we could use this framework to

apply tokenisation to other use cases - transport, VOD... - that could rely on it. Indeed, we have a generic token provisioning process that could be used in other case scenarii.

Conclusion and perspectives

During this thesis, we designed two security protocols answering to two separate use cases: electronic voting and mobile payment. We aimed for them to be practical from the user's point of view and secure regarding their respective security expectations. Along the specifications, we provided formal security analysis of both protocols using two different model checkers: ProVerif and Tamarin. The attacker model on both cases being as strong as possible.

We describe here future work on both approaches.

1 On web-based voting

This use case is covered in part I of this thesis.

1.1 Belenios VS: a verifiable and private voting protocol that is secure even if the user's device is compromised

We proposed the specification of an improvement of Belenios RF voting protocol in chapter 2. With Belenios VS, we aimed at providing a voting scheme that was verifiable and private even though the device used by the voter to cast the ballot was compromised. Details on the security assumptions of this statement are provided in chapter 4.

The protocol was formally proved and studied with the help of the ProVerif tool. This analysis is given in chapter 4 and the file proofs are available at [1].

Does privacy imply verifiability?

From our security analysis of Belenios VS, we found an attack against privacy that seemed to happen whenever the attacker has the capacity of forcing the vote of a honest voter. The attack, described in section 4.2.3, appears to be quite generic and applicable to other voting schemes that do not guarantee that an attacker cannot vote for a honest voter. Intuitively, the attack is quite easy to understand if we consider an election with two honest voters only A and B. If the attacker is able to cast a vote instead of voter A, then it can deduce from the election result for which candidate B voted. The attacker can *indirectly* guess the vote of any honest voter as soon as it can impersonate other honest voters. In other terms, if a voting protocol is not verifiable, does it imply that it is not private either?

Answering this question could be the focus on a potential future work.

1.2 Proposing a method to automatize the verifiability proof in the symbolic model

Since verifiability cannot be expressed with nowadays automated tools, we stated two theorems that provide sufficient conditions expressed in a way that ease its automatic proof with tools like ProVerif

in chapter 3. In essence, we provided two different sets of simple trace properties that, if satisfied by a voting protocol regarding some specific security assumptions, imply that the protocol is verifiable.

Widen the scope of our theorems

One of the main limitation of our theorems is the fact that they exclude protocols authorizing *revote* (like Civitas). Yet, the *strong verifiability* definition we rely on seems to be adaptable to make sense when revoting is possible - for instance by assuming that only the last ballot that was cast is part of the final tally. Same goes for our sets of trace properties. Assuming we could have a formal tool that would allow us to express an order between the time ballots are cast, we could adapt our theorems consequently. First part of this potential work would be to find out if our theorem can easily be extended to include protocols that allow revoting. And if such an extension is possible, can the security proofs still be managed by a tool like ProVerif? Since intuitively we would need some notion of order between ballot, something that ProVerif does not necessarily handle well at the time, what could improve the tool in this way?

2 On mobile payment

Part II focused on this use case.

2.1 Devising an open end-to-end mobile payment protocol

As seen in chapter 5, the mobile payment industry is quite opaque regarding the specifications of payment applications. At most we could find frameworks and recommendations about how to implement a payment application, analyze the proprietary solutions patents and inspire us from the card-based transactions specifications to picture how security is handled in such applications. Hence, we devised - to the best of our knowledge - the first open end-to-end security protocol for a mobile payment application along its security proof performed with the Tamarin prover tool. Chapter 6 describes this protocol. We formally proved that our protocol was at least as secure as Chip and PIN cards regarding its security property with the addition of client unlinkability from the merchant point of view.

Chapter 7 exposes our security analysis and Tamarin's file proof for our protocol available at [35].

What about the registration process and the payment revocability?

Our protocol assumed that the registration process was already done. While several solutions regarding the registration of users are implemented in the industrial world, there seems to be a lack of comprehensive view about what the security goals are, what are the limits imposed by legislation regarding data protection and what can be guaranteed by nowadays technology. Moreover, although we proved that our protocol was formally secure and practical by implementing a prototype, this does not mean that an application implementing this protocol will necessarily be secure, for technical errors can always happen. This is why we need a possibility to revoke a payment from the user's point of view, a process that has not been described in our scheme.

Further improvement of our payment protocol would imply the specification of at least those two processes.

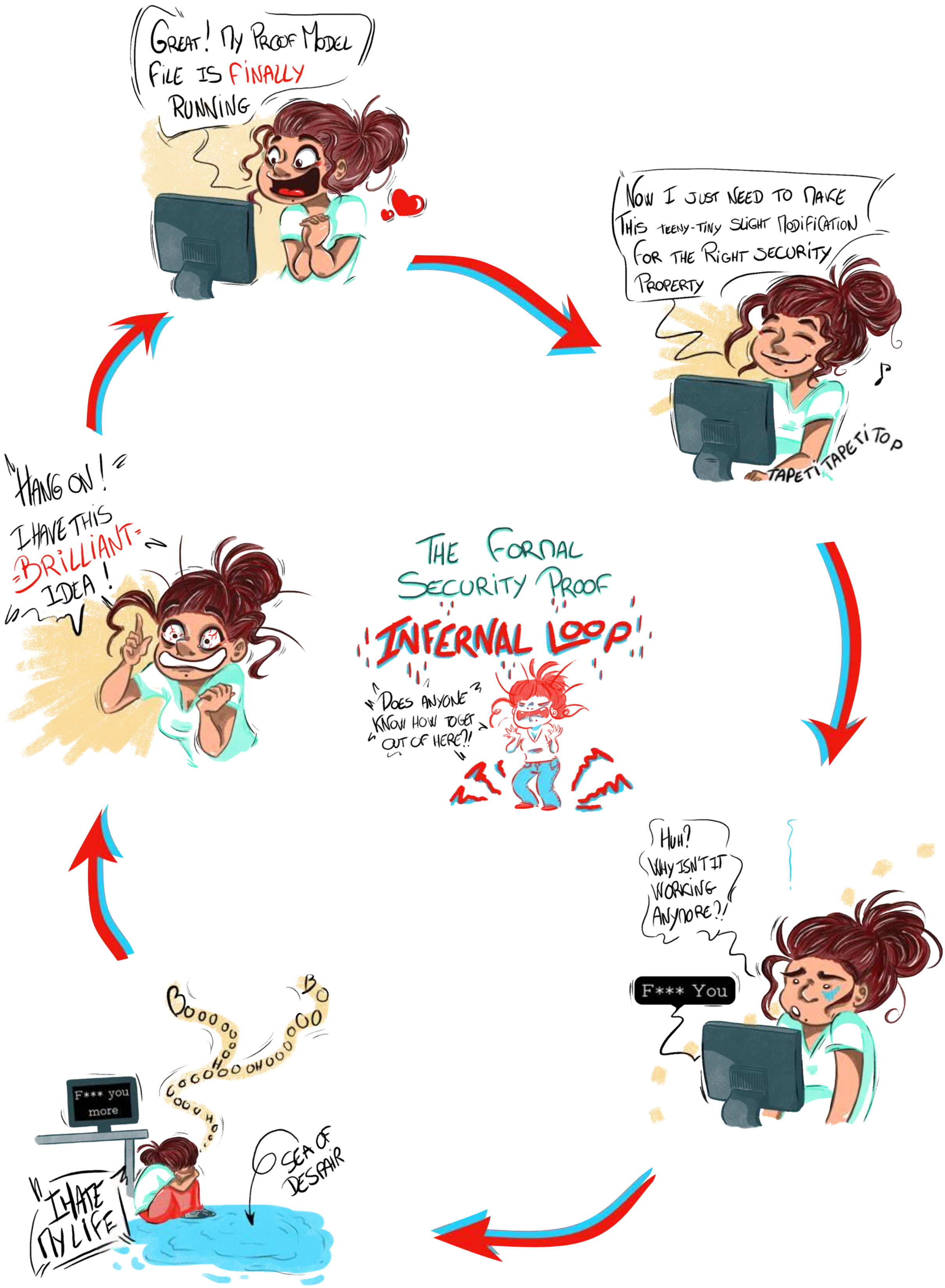
2.2 A framework for tokenised services

Our payment protocol relies on tokenisation, a process that provides ephemeral aliases instead of actual critical values. The design of our payment protocol intrinsically separates the tokenisation process from

the payment process. In other word, we could extrapolate our tokenisation process to other use cases such as ticketing for transportation or on demand services.

Is it possible to implement a generic secure path between the user and a Secure Element at an industrial level?

However, for our system to be sustainable, we made the technical assumption that it was possible to securely identify the user to limit the access to a secure element performing critical operation. Although this job can theoretically be done by Trusted Execution Environment and even though several specification on this need have been provided by the industry, to our knowledge, no modern solution on Android actually implements a trusted user interface for user identification and access control to a secure element. This need is pretty generic if we aim at providing secure applications and we believe that there is room for further technical improvements. Exploring the actual possibilities offered by TEE in the industrial world could be another scope for future work.


Great! My Proof Model File is finally running
Now I just need to make this teeny-tiny slight modification for the right security property
TAPETI TAPETI TOP
Huh? Why isn't it working anymore?!
F*** You
Booooo OHOOOO BOOOOOO
F*** you more
SEA OF DESPAIR
I HATE MY LIFE
"Hang on!" I have this brilliant idea!
THE FORMAL SECURITY PROOF
INFERNAL LOOP
"Does anyone know how to get out of here?!"

Bibliography

[1] ProVerif protocol models and proofs for Belenios VS: https://members.loria.fr/AFilipiak/formal-analysis-of-the-belenios-vs-protocol/.

[2] ETSI TS 102 221. Smart Cards - UICC-Terminal interface - Physical and logical characteristics - v.13, June 2009.

[3] Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08. USENIX Association, 2008.

[4] Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of helios. In *Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, EVT/WOTE'09. USENIX Association, 2009.

[5] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *CSF 2010*, pages 107–121, 2010.

[6] Myrto Arapinis, Véronique Cortier, and Steve Kremer. When are three voters enough for privacy properties? In Ioannis Askoxylakis, Sotiris Ioannidis, Sokratis Katsikas, and Catherine Meadows, editors, *Proceedings of the 21st European Symposium on Research in Computer Security (ESORICS'16)*, Lecture Notes in Computer Science, pages 241–260. Springer, September 2016.

[7] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Rfc 4033 - dns security introduction and requirements. Rfc standard, IETF, March 2005.

[8] Alessandro Armando, Roberto Carbone, and Luca Compagna. Satmc: A sat-based model checker for security-critical systems, 2014.

[9] SD Association. Activating New Mobile Services and Business Models with smart-SD Memory cards, November 2014.

[10] David Baelde, Stéphanie Delaune, Ivan Gazeau, and Steve Kremer. Symbolic verification of privacy-type properties for security protocols with xor. In *Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 234–248. IEEE Computer Society Press, August 2017.

[11] David Preston Baker, Mohamed Reza Hussein, Stanley N. Marshall, Matthew Eric Hiller, Chin Pang Tung, and Andrew Robert Mitchell. Secure storage of payment information on client devices, 12 2013.

[12] David Preston Baker, Mohamed Reza Hussein, Stanley N. Marshall, Matthew Eric Hiller, Chin Pang Tung, and Andrew Robert Mitchell. Optimistic receipt flow, 10 2015.

[13] David Preston Baker, Stanley N. Marshall, Matthew Eric Hiller, Andrew Robert Mitchell, Mohamed Reza Hussein, and Chin Pang Tung. Secure transmission of payment credentials, 05 2016.

[14] David Basin, Jannik Dreier, and Ralf Sasse. Automated Symbolic Proofs of Observational Equivalence. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1144–1155, 2015.

[15] David Basin, Sebastian Mödersheim, and Luca Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In Einar Snekkenes and Dieter Gollmann, editors, *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, Heidelberg, 2003.

[16] David Basin, Sebastian Mödersheim, and Luca Viganò. Ofmc: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, Jun 2005.

[17] David A. Basin, Jannik Dreier, and Ralf Sasse. Automated symbolic proofs of observational equivalence. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1144–1155, 2015.

[18] Josh Benaloh. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, EVT'06, pages 5–5. USENIX Association, 2006.

[19] Bruno Blanchet. An Automatic Security Protocol Verifier based on Resolution Theorem Proving (invited tutorial). In *20th International Conference on Automated Deduction (CADE-20)*, Tallinn, Estonia, July 2005.

[20] Bruno Blanchet. *Security Protocol Verification: Symbolic and Computational Models*, pages 3–29. Springer Berlin Heidelberg, 2012.

[21] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and ProVerif. *Foundations and Trends in Privacy and Security*, 1(1–2):1–135, October 2016.

[22] Bruno Blanchet and Ben Smyth. Automated reasoning for equivalences in the applied pi calculus with barriers. Research report, Inria, April 2016.

[23] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. *Signatures on Randomizable Ciphertexts*, pages 403–422. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[24] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. *Signatures on Randomizable Ciphertexts*, pages 403–422. PKC '11. Springer Berlin Heidelberg, 2011.

[25] Yohan Boichut, Nikolai Kosmatov, and Laurent Vigneron. Validation of Prouve Protocols using the Automatic Tool TA4SP. In *3rd Taiwanese-French Conference on Information Technology*, pages 467–480, Nancy/France, March 2006.

[26] Mike Bond, Omar Choudary, Steven J. Murdoch, Sergei Skorobogatov, and Ross Anderson. Chip and Skim: cloning EMV cards with the pre-play attack. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, 2014.

[27] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. Cryptology ePrint Archive, Report 2015/753, 2015. `http://eprint.iacr.org/2015/753`.

[28] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocol. *ACM Transactions on Computational Logic*, 17(4), November 2016.

[29] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. Beleniosrf: A non-interactive receipt-free electronic voting scheme. In *23rd ACM Conference on Computer and Communications Security (CCS'16)*, pages 1614–1625, Vienna, Austria, October 2016. ACM.

[30] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *Proceedings of the Conference on Electronic Voting Technology*, EVT'08, pages 14:1–14:13. USENIX Association, 2008.

[31] Vincent Cheval. Apte: an algorithm for proving trace equivalence. In Erika Ábrahám and JKlaus Havelund, editors, *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, volume 8413 of *Lecture Notes in Computer Science*, pages 587–592. Springer Berlin Heidelberg, April 2014.

[32] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, pages 354–368. IEEE Computer Society, 2008.

[33] Estonian National Electoral Committee. E-Voting System, 2010.

[34] Véronique Cortier, Stéphanie Delaune, and Antoine Dallon. Sat-equiv: an efficient tool for equivalence properties. In *Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 481–494. IEEE Computer Society Press, August 2017.

[35] Véronique Cortier, Alicia Filipiak, Jan Florent, Said Gharout, and Jacques Traoré. TAMARIN protocol model and proofs: https://members.loria.fr/AFilipiak/designing-and-proving-an-emv-compliant-payment-protocol-for-mobile-devices-formal-analysis/.

[36] Véronique Cortier, Alicia Filipiak, Jan Florent, Said Gharout, and Jacques Traoré. Designing and proving an emv-compliant payment protocol for mobile devices. In *2nd IEEE European Symposium on Security and Privacy (EuroSP'17)*, pages 467–480, 2017.

[37] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Distributed elgamal à la pedersen: Application to helios. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES '13, pages 131–142. ACM, 2013.

[38] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachene. Election verifiability for helios under weaker trust assumptions. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS'14)*, volume 8713 of *LNCS*, pages 327–344, Wroclaw, Poland, September 2014. Springer.

[39] Véronique Cortier, David Galindo, and Mathieu Turuani. A formal analysis of the Neuchâtel e-voting protocol. Research report, Inria, October 2017.

[40] Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. A type system for privacy properties. In *24th ACM Conference on Computer and Communications Security (CCS'17)*. ACM, October 2017.

[41] Véronique Cortier and Ben Smyth. Attacking and fixing helios: an analysis of ballot secrecy. Cryptology ePrint Archive, Report 2010/625, 2010.

[42] Véronique Cortier, Fabienne Eigner, Steve Kremer, Matteo Maffei, and Cyrille Wiedling. Type-based verification of electronic voting protocols. Cryptology ePrint Archive, Report 2015/039, 2015. `https://eprint.iacr.org/2015/039`.

[43] Common Criteria. Common Criteria for Information Technology Security Evaluation, Part 1 – Version 3.1 Revision 4, September 2012.

[44] Joeri de Ruiter and Erik Poll. Formal Analysis of the EMV Protocol Suite. In S. Mödersheim and C. Palamidessi, editors, *Theory of Security and Applications*, volume 6993 of *Lecture Notes in Computer Science*, pages 113–129. Springer Berlin / Heidelberg, 2012.

[45] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols: A taster. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Mirosław Kutyłowski, and Ben Adida, editors, *Towards Trustworthy Elections – New Directions in Electronic Voting*, volume 6000 of *Lecture Notes in Computer Science*, pages 289–309. Springer, May 2010.

[46] Timothy M. Dierks. On-line payment transactions, 08 2017.

[47] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, March 1983.

[48] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18. Springer-Verlag New York, Inc., 1985.

[49] Samsung Electronics. Mobile Tech Insights - Samsung Pay Security.

[50] EMVCo. EMVCo's website.

[51] EMVCo. *Book 1 - Application Independent ICC to Terminal Interface Requirements*, November 2011.

[52] EMVCo. *Book 2 - Security and Key Management*, November 2011.

[53] EMVCo. *Book 3 - Application Specification*, November 2011.

[54] EMVCo. *Book 4 - Cardholder, Attendant, and Acquirer Interface Requirements*, November 2011.

[55] EMVCo. *EMV Payment Tokenisation Specification – Technical Framework*, March 2014.

[56] EMVCo. 3-d secure specification, v2.1, October 2017.

[57] Santiago Escobar, Catherine A. Meadows, and José Meseguer. Maude-npa: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, pages 1–50, 2007.

[58] Ariel J. Feldman, J. Alex Halderman, and Edward W. Felten. Security analysis of the diebold accuvote-ts voting machine. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, EVT'07, pages 2–2. USENIX Association, 2007.

[59] R. Fielding and J. Reschke. Rfc 7230 to 7237 - hypertext transfer protocol. Rfc standard, IETF, June 2014.

[60] Transport for London. TfL Oyster app, 2017.

[61] Frank Gangi. Wallet consolidator, 10 2000.

[62] Frank Gangi. Method and apparatus for combining data for multiple magnetic stripe cards or other sources, 06 2002.

[63] Frank Gangi. Wallet consolidator and related methods of processing a transaction using a wallet consolidator, 03 2008.

[64] GlobalPlatform. Trusted User Interface API, June 2013.

[65] GlobalPlatform. Card Specification v.2.3, December 2015.

[66] Google. Android Pay implementation guide.

[67] William Wang Graylin, Man Ho Li, and Jimmy Tai Kwan Tang. Mobile checkout systems and methods, 06 2017.

[68] GSMA. Remote Provisioning Architecture for Embedded UICC - SGP.02, October 2014.

[69] GSMA. Embedded UICC Protection Profile, Version 1.1/25.08.2015 , BSI-CC-PP-0089, August 2015.

[70] Enyang Huang. Lump sequences for multi-track magnetic stripe electronic data transmission, 10 2017.

[71] Enyang Huang, William Wang Graylin, and George Wallner. Terminal for magnetic secure transmission, 05 2016.

[72] Payment Card Industry. Payment Card Industry Data Security Standard v.3.2, April 2016.

[73] Information Sciences Institute. Rfc 793 - transmission control protocol. Rfc standard, IETF, September 1981.

[74] Fan Jiang, Aneto Pablo Okonkwo, and Erwin Aitenbichler. Secure offline payment system, 10 2015.

[75] Ahmer Khan. Method to send payment data through various air interfaces without compromising user data, 01 2014.

[76] Ahmer Khan, Timothy Hurley, Anton Diederich, George Dicker, Scott Herz, and Christopher Sharp. Online payments using a secure element of an electronic device, 04 2015.

[77] Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. In *Proceedings of the 35th IEEE Symposium on Security and Privacy*, SP '14. IEEE Computer Society, May 2014.

[78] Steve Kremer and Mark D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In Mooly Sagiv, editor, *Programming Languages and Systems — Proceedings of the 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200. Springer, April 2005.

[79] Stephen Kuhn and Angelie Zaslavsky. Display screen or portion thereof with an animated graphical user interface, 06 2016.

[80] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A Lightweight Verifiable Remote Voting System. In *IEEE 29th Computer Security Foundations Symposium (CSF 2016)*, pages 341–354. IEEE Computer Society, 2016.

[81] Ralf Kusters, Tomasz Truderung, and Andreas Vogt. Clash attacks on the verifiability of e-voting systems. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 395–409. IEEE Computer Society, 2012.

[82] Daryl Mun-Kid Low, Ronald Keryuan Huang, Puneet Mishra, Gaurav Jain, Jason Gosnell, and Jeff Bush. Group formation using anonymous broadcast information, 03 2010.

[83] Daryl Mun-Kid Low, Ronald Keryuan Huang, Puneet Mishra, Gaurav Jain, Jason Gosnell, and Jeff Bush. Group formation using anonymous broadcast information, 11 2011.

[84] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, TACAs '96, pages 147–166. Springer-Verlag, 1996.

[85] Simon Meier, David Basin, Benedikt Schmidt, Jannik Dreier, Ralf Sasse, and Cas Cremers. Tamarin prover for security protocol verification - Project Page.

[86] Simon Meier, Cas Cremers, and David Basin. Strong Invariants for the Efficient Construction of Machine-Checked Protocol Security Proofs. In *23rd IEEE Computer Security Foundations Symposium*, pages 231–245, Los Alamitos, USA, July 2010. IEEE Computer Society.

[87] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The tamarin prover for the symbolic analysis of security protocols. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV'13, pages 696–701, Berlin, Heidelberg, 2013. Springer-Verlag.

[88] Salvador Mendoza. Samsung Pay: tokenized numbers flaws and issues, 2016.

[89] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Information and Computation*, 100(1):1 – 40, 1992.

[90] P. Mockapetris. Rfc 1034 and 1035 - domain names. Rfc standard, IETF, November 1987.

[91] Sebastian Mödersheim and Luca Viganò. The open-source fixed-point model checker for symbolic analysis of security protocols, 2009.

[92] Steven J. Murdoch, Saar Drimer, Ross Anderson, and Mike Bond. Chip and PIN is broken. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, 2010.

[93] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21:993–999, December 1978.

[94] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. Rfc 4120 - the kerberos network authentication service (v5). Rfc standard, IETF, July 2005.

[95] H. Orman. Rfc 2412 - the oakley key determination protocol. Rfc standard, IETF, November 1998.

[96] John Osborne. Methods and apparatus for barcode reading and encoding, 06 2015.

[97] Vijayakrishnan Pasupathinathan, Josef Pieprzyk, Huaxiong Wang, and Joo Yeon Cho. Formal analysis of card-based payment systems in mobile devices. In *Proceedings of the 2006 Australasian Workshops on Grid Computing and e-Research - Volume 54*, ACSW Frontiers '06, pages 213–220. Australian Computer Society, Inc., 2006.

[98] J. Postel. Rfc 768 - user datagram protocol. Rfc standard, IETF, August 1980.

[99] E. Rescorla. Rfc 2818 - http over tls. Rfc standard, IETF, May 2000.

[100] Ronald L. Rivest and Warren D. Smith. Three voting protocols: Threeballot, vav, and twin. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, EVT'07, pages 16–16, Berkeley, CA, USA, 2007. USENIX Association.

[101] P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia. Prêt à Voter: a Voter-Verifiable Voting System. *IEEE Transactions on Information Forensics and Security*, 4(4):662–673, Dec 2009.

[102] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. *Selene: Voting with Transparent Verifiability and Coercion-Mitigation*, pages 176–192. Springer Berlin Heidelberg, 2016.

[103] Kazue Sako and Joe Kilian. *Receipt-Free Mix-Type Voting Scheme*, pages 393–403. Springer Berlin Heidelberg, 1995.

[104] Benedikt Schmidt, Ralf Sasse, Cas Cremers, and David Basin. Automated Verification of Group Key Agreement Protocols. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 179–194, Washington, DC, USA, 2014. IEEE Computer Society.

[105] C. P. Schnorr. Efficient identification and signatures for smart cards, 1990.

[106] SIMalliance. Open Mobile API Specification, August 2014.

[107] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. Security analysis of the estonian internet voting system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 703–715. ACM, 2014.

[108] International Standard. *Financial transaction card originated messages - Interchange message specificationsy*, 2003.

[109] Alwen Tiu, Nam Nguyen, and Ross Horne. Spec: An equivalence checker for security protocols, 2016.

[110] Mathieu Turuani. *The CL-Atse Protocol Analyser*, pages 277–286. Springer Berlin Heidelberg, 2006.

[111] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2. In *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*. ACM, 2017.

[112] George Wallner. Detachable electronic payment device, 08 2016.

[113] George Wallner. System and method for a baseband nearfield magnetic stripe data transmitter, 06 2016.

[114] George Wallner. Magnetic secure transmission device hardware, 05 2017.

Résumé

Les “smart-devices” tels les smartphones, tablettes et même les montres ont été largement démocratisés au cours de la dernière décennie. Dans nos sociétés occidentales, on ne garde plus seulement son ordinateur personnel chez soi, on le transporte dans la poche arrière de son pantalon ou bien autour de son poignet. Ces outils ne sont d'ailleurs plus limités, en termes d'utilisation, à de la simple communication par SMS ou bien téléphone, on se fie à eux pour stocker nos photos et données personnelles, ces dernières parfois aussi critiques que des données de paiement bancaires, on gère nos contacts et finances, se connecte à notre boite mail ou un site marchand depuis eux... Des exemples récents nous fournissent d'ailleurs un aperçu des tâches de plus en plus complexes que l'on confie à ces outils : l'Estonie autorise l'utilisation de smartphones pour participer aux scrutins nationaux et en 2017, la société Transport for London a lancé sa propre application autorisant l'émulation d'une Oyster card et son rechargement pour emprunter son réseau de transports publics. Plus les services se complexifient, plus la confiance qui leur est accordée par les groupes industriels et les utilisateurs grandit.

Nous nous intéressons ici aux protocoles cryptographiques qui définissent les échanges entre les outils et entités qui interviennent dans l'utilisation de tels services et aux garanties qu'ils proposent en termes de sécurité (authentification mutuelle des agent, intégrité des messages circulant, secret d'une valeur critique...). Moult exemples de la littérature et de la vie courante ont démontré que leur élaboration était hautement vulnérable à des erreurs de design. Heureusement, des années de recherches nous ont fournis des outils pour rendre cette tâche plus fiable, les méthodes formelles font partie de ceux-là. Il est possible de modeler un protocole cryptographique comme un processus abstrait qui manipule des données et primitives cryptographiques elles aussi modélisées comme des termes et fonctions abstraites. On met le protocole à l'épreuve face à un attaquant actif et on peut spécifier mathématiquement les propriétés de sécurité qu'il est censé garantir. Ces preuves de sécurité peuvent être automatisées grâce à des outils tels que ProVerif ou bien Tamarin.

L'une des grandes difficultés lorsque l'on cherche à concevoir et prouver formellement la sécurité d'un protocole de niveau industriel réside dans le fait que ce genre de protocole est généralement très long et doit satisfaire des propriétés de sécurité plus complexes que certains protocoles universitaires. Au cours de cette thèse, nous avons souhaité étudier deux cas d'usage : le vote électronique et le paiement mobile. Dans les deux cas, nous avons conçu et prouvé la sécurité d'un protocole répondant aux problématiques spécifiques à chacun des cas d'usage.

Dans le cadre du vote électronique, nous proposons le protocole Belenios VS, une variante de Belenios RF. Nous définissons l'écosystème dans lequel le protocole est exécuté et prouvons sa sécurité grâce à ProVerif. Belenios VS garantit la confidentialité du vote et le fait qu'un utilisateur puisse vérifier que son vote a bien fait parti du résultat final de l'élection, tout cela même si l'outil utilisé par le votant est sous le contrôle d'un attaquant.

Dans le cadre du paiement, nous avons proposé la première spécification ouverte de bout en bout d'une application de paiement mobile. Sa conception a pris en compte le fait qu'elle devait pouvoir s'adapter à l'écosystème de paiement déjà existant pour être largement déployable et que les coûts de gestion, de développement et de maintenance de la sécurité devait être optimisés.

Mots-clés: cryptographie, sécurité, méthodes formelles, systèmes mobiles et embarqués, vote, paiement

Abstract

The last decade has seen the massive democratization of smart devices such as phones, tablets, even watches. In the wealthiest societies of the world, not only do people have their personal computer at home, they now carry one in their pocket or around their wrist on a day to day basis. And those devices are no more used simply for communication through messaging or phone calls, they are now used to store personal photos or critical payment data, manage contacts and finances, connect to an e-mail box or a merchant website... Recent examples call for more complex tasks we ask to such devices: Estonia voting policy allows the use of smart ID cards and smartphones to participate to national elections. In 2017, Transport for London launched the TfL Oyster app to allow tube users to top up and manage their Oyster card from their smartphone. As services grow with more complexity, so do the trust users and businesses put in them. We focus our interest into cryptographic protocols which define the exchanges between devices and entities so that such interaction ensure some security guarantees such as authentication, integrity of messages, secrecy... Their design is known to be an error prone task. Thankfully, years of research gave us some tools to improve the design of security protocols, among them are the formal methods: we can model a cryptographic protocol as an abstract process that manipulates data and cryptographic function, also modeled as abstract terms and functions. The protocol is tested against an active adversary and the guarantees we would like a protocol to satisfy are modeled as security properties. The security of the protocol can then be mathematically proven. Such proofs can be automated with tools like ProVerif or Tamarin.

One of the big challenge when it comes to designing and formally proving the security an "industrial-level" protocol lies in the fact that such protocols are usually heavier than academic protocols and that they aim at more complex security properties than the classical ones. With this thesis, we wanted to focus on two use cases: electronic voting and mobile payment. We designed two protocols, one for each respective use case and proved their security using automated prover tools.

The first one, Belenios VS, is a variant of an existing voting scheme, Belenios RF. It specifies a voting ecosystem allowing a user to cast a ballot from a voting sheet by flashing a code. The protocol's security has been proven using the ProVerif tool. It guarantees that the vote confidentiality cannot be broken and that the user is capable of verifying their vote is part of the final result by performing a simple task that requires no technical skills all of this even if the user's device is compromised – by a malware for instance.

The second protocol is a payment one that has been conceived in order to be fully scalable with the existing payment ecosystem while improving the security management and cost on the smartphone. Its security has been proven using the Tamarin prover and holds even if the user's device is under an attacker's control.

Keywords: cryptography, security, formal methods, mobile and embedded systems, voting, payment

Résumé étendu de la thèse

La dernière décennie a vu se développer massivement l'usage des *smart devices* tels que les smartphones et tablettes en s'étendant jusqu'aux objets du quotidiens à l'instar des montres. Dans les sociétés occidentales, l'ordinateur n'est plus cantonné au domicile, il se porte désormais dans la poche arrière du pantalon ou autour du poignet et son usage va bien au delà des services de messagerie ou téléphonie ; dorénavant, ces outils gèrent jusqu'au plus petit détail de la vie de l'utilisateur : prise et stockage de photos personnelles, gestion des contacts et de la carte bleue, connexion à sa messagerie en ligne ou sa banque... Ils sont même utilisés pour des tâches plus complexes que la gestion et le stockage de valeurs critiques. En Estonie, il est possible de voter en utilisant son smartphone. A Londres, la société de transports permet de gérer ses titres de transport *via* une application. Autant de services dont les usages se complexifient au fil du temps.

Parallèlement à la complexification des différents service fournis par ses outils, le niveau de confiance requis envers ces derniers croit. Une application de paiement doit garantir qu'un utilisateur ne se fera pas voler des données critiques tout en permettant une intégration aux systèmes de paiement préexistants maximale. Un protocole de vote électronique doit convaincre chaque électeur que son vote fera partie du scrutin final tout en préservant le secret dudit vote. De plus, ce processus de vérification du vote ne doit pas requérir des capacités techniques avancées de la part du votant pour que l'élection reste accessible à tous.

On remarque alors que la confiance en ces services ne repose pas sur la simple confiance en son propre matériel. Les applications font partie d'un écosystème complexe faisant intervenir plus de deux entités dans la plupart des cas. Prenons le cas du paiement. Certes, les données de paiement doivent être stockées de manière suffisamment sécurisée par le smartphone du payeur. Cependant, il ne faudrait pas que le terminal du marchand qui traite ce paiement permette une fuite qui puisse mener à un vol ou une refacturation. Le marchand, de son côté, s'attendra à ce que ces données de paiement soient authentiques et que si la banque du client lui notifie le succès d'une transaction, il sera bien évidemment payé le montant adéquat. Au bout de la chaîne de transaction, les banques respectives des marchands et payeurs doivent minimiser le risque de fraude.

La conception de protocoles cryptographiques de sécurité

Les protocoles d'échange spécifient comment les appareils et entités d'un service communiquent entre eux et sont utilisés au quotidien sur Internet (on peut citer, parmi d'autres exemples, HTTP et sa version sécurisée, HTTPS, DNS, TCP, UDP...). Les *protocoles cryptographiques* sont, eux, conçus pour garantir certaines propriétés de sécurité attendues d'un service. Certaines de ces propriétés sont classiques et académiquement bien documentées et mises en pratiques telles que les diverses propriétés d'authentification d'entités entre elles, à l'instar de celles mises en œuvre dans le protocole Kerberos, ou bien le secret d'une certaine valeur comme le code PIN stocké dans la carte bleue. D'autres propriétés sont moins génériques, comme celles qui encadrent les protocoles de vote électronique par exemple : un

protocole de vote doit garantir le *secret du vote* de l'électeur, y compris si celui-ci souhaite le communiquer pour, par exemple, vendre son vote mais il faut aussi que l'électeur dispose d'un élément personnel lui permettant de *vérifier* que son vote a bien été dépouillé et comptabiliser pour le résultat final de l'élection. Il faut également que tout observateur de l'élection soit en mesure de vérifier que le scrutin reflète bien le contenu de l'ensemble des bulletins envoyés par des personnes autorisées à participer à l'élection.

La conception de tels protocoles est une tâche encline à l'erreur. On peut citer le célèbre exemple du protocole de Needham-Schroeder, un protocole académique d'authentification dont il a été démontré qu'il était sujet à une attaque de type "man in the middle" dix-sept ans après sa création. Plus récemment, on retiendra l'attaque sur WPA2, le protocole censé sécuriser la connexion à un réseau sans fil. Dans chacun de ces exemples, la cryptographie n'est pas à blâmer pour les failles, c'est la conception même des protocoles qui permet à l'attaquant de rejouer ou intercepter des valeurs critiques ou bien d'exploiter les relations cryptographiques entre lesdites valeurs, cassant par là-même la sécurité du protocole. Par ailleurs, les hypothèse de sécurité sur les entités de confiance sont également sujettes à des failles : fait-on confiance à l'état organisateur d'une élection ou au terminal d'un marchand potentiellement corrompu ? En effet, un attaquant est rarement une entité passive se contenter de jouer les passe-plats, il est crucial de définir deux point importants lors de la conception d'un protocole de sécurité :

- Que peut "raisonnablement" faire un attaquant ?

- Qui est l'attaquant ?

Ce qui se résume à la question plus globale de quel est notre modèle d'attaquant ?

Des années de recherches nous ont apportés des outils conceptuels et mathématiques permettant de prouver la sécurité des protocoles cryptographiques. dans ce domaine, deux mondes coexistent : le modèle calculatoire et le modèle logique. Le premier permet la recherche de défauts dans les algorithmes utilisés par les primitives cryptographiques, c'est celui qui est majoritairement utilisé par les cryptologues. Le second, en revanche, considère que les primitives cryptographiques sont parfaites et recherche plutôt les failles logiques au niveau du protocole global. Cette thèse met en œuvre des techniques du modèle symbolique, nous ne discuterons pas ici de la pertinence ou non d'un modèle plutôt que l'autre selon le contexte, ceci ayant déjà été largement documenté.

Dans le modèle symbolique, les primitives cryptographiques sont considérées de manière abstraite comme des symboles de fonctions et les opérations qu'elles exécutent - comme le couplage, le chiffrement, la signature... - comme des règles de réécriture ou des égalités modulo une théorie équationnelle. Par exemple, un schéma de chiffrement asymétrique fera intervenir trois symboles de fonctions (aenc pour l'opération de chiffrement, adec pour celle de déchiffrement et pk pour la génération d'une clef publique à partir d'une clef privée) et une règle de réécriture de la forme : pour tout message m et pour toute clef secrète k : $\mathsf{adec}(\mathsf{aenc}(m, \mathsf{pk}(k)), k) \rightarrow m$. En d'autres termes, la clef secrète k permet de déchiffrer tout message chiffré par la clef publique associée $\mathsf{pk}(k)$.

Le modèle symbolique permet également de modéliser un attaquant actif sur le réseau public, connu sous le nom de modèle de Dolev-Yao : il peut écouter le réseau public et interagir avec les entités y ayant recours que ce soit en recevant ou en envoyant des messages et est capable d'exécuter les mêmes primitives cryptographiques que celles utilisées dans le protocole (il pourra par exemple obtenir le haché d'une valeur ou en chiffrer une autre). Ce modèle d'attaquant, très puissant, permet de trouver de nombreuses failles logiques dans les protocoles cryptographiques.

Outre ces capacités de modélisation, le modèle symbolique est suffisamment expressif pour modéliser des propriétés de sécurité, souvent sous la forme de lemmes logiques, on peut d'ailleurs diviser les types de propriétés en deux catégories majeures : les propriétés de trace, qui permettent

d'exprimer le secret d'une valeur ou la correspondance entre plusieurs états d'un protocole, et les propriétés d'équivalence, qui permettent d'exprimer le fait que deux processus doivent être indistinguables au yeux d'un attaquant potentiel. Les premières sont utilisées principalement pour modéliser les propriétés de secret, d'authentification mutuelle ou d'accord sur une clef tandis que les autres interviennent plus au niveau de l'anonymat ou de l'intraçabilité.

Dernier point important concernant le modèle symbolique : les preuves peuvent être automatisées. Nous avons, au cours de cette thèse, pu travailler avec deux outils permettant la démonstration des deux types de propriétés : ProVerif et Tamarin.

Cadre et découpage de cette thèse

Les outils de preuve précédemment mentionnés ont été utilisés massivement pour étudier des protocoles de sécurité classiques, tels que les différents protocoles d'authentification, dans le monde académique. Il est plus rare de voir l'analyse de protocoles de niveau "industriel", tels que le paiement ou le vote électronique. Ceux-ci sont en effet plus lourds et manipulent un nombre de données généralement plus conséquent. Les propriétés de sécurité attendues de tels protocoles sont souvent plus complexes : Comment être convaincu du fait que les informations de paiement sont bien tenus secrètes au cours d'une transaction ? Quels sont les garanties en cas de corruption d'un smartphone ou d'un marchand dans de tels cas ? Comment être sûr que le gagnant d'une élection est bien celui élu par les citoyens, et ceux, même si l'on ne fait pas confiance à l'état organisateur de l'élection ?

Le but de cette thèse est de proposer deux protocoles pratiques d'utilisation dont la spécification est accompagnée de la preuve de sécurité, l'un concernant le vote électronique, le second le paiement mobile.

Vote électronique : le protocole de vote Belenios VS

Cela fait plus de deux siècles que l'automatisation du vote est un sujet de recherche. Depuis les premières machines à voter utilisées au dix-neuvième siècle par les chartistes anglais au système de vote électronique utilisé en Estonie ou au Brésil, nombre d'ingénieurs ont proposé des infrastructures et outils censés garantir la sécurité et l'accessibilité du vote à tout un chacun. On peut distinguer deux familles de votes automatisés à ce jour : le bureau de vote assisté par une machine, où l'électeur doit utiliser une machine présente dans le bureau pour enregistrer son choix, à l'instar des machines Diebold utilisées aux États-Unis et le vote par correspondance (ou vote par Internet) ou le vote est effectué par l'électeur depuis son propre outil, comme c'est une des possibilités en Estonie. Comme pour le vote physique traditionnel, l'une des principales préoccupations concernant le vote électronique réside dans le fait qu'une "bonne élection" doit satisfaire *a minima* deux propriétés qui, combinées, peuvent paraître assez paradoxales : le vote doit demeurer confidentiel tout en permettant à tout un chacun de vérifier sur la base d'une valeur personnelle que leur vote a bien été pris en compte lors du scrutin.

De nombreuses études ont été menées au cours des deux dernières décennies pour analyser des schémas existants, certaines dévoilant de sérieuses failles au sein des outils utilisés dans des démocraties comme le cas des machines du fournisseur Diebold aux États-Unis ou plus récemment les outils exploités par l'administration brésilienne pour ses élections. On peut également relever que l'analyse des protocoles de vote par Internet suppose le matériel du votant comme n'étant pas corrompu, une hypothèse que nous ne souhaitions pas faire.

Contributions

Nous apportons principalement deux contributions dans le domaine du vote par correspondance :

- **Proposer une amélioration du protocole de vote Belenios RF : le protocole de vote Belenios VS et sa preuve de sécurité en ProVerif :** nous avons conçu une amélioration du protocole de vote Belenios RF appelée *Belenios Voting Sheet* - ou Belenios VS. En effet, Belenios RF garantit à la fois la vérifiabilité et la confidentialité du vote mais sa sécurité suppose que la machine utilisée par l'électeur pour calculé et envoyer son bulletin de vote est sûre, une hypothèse de sécurité que nous considérons comme forte. Il suffirait que la machine de l'électeur soit corrompue pour briser la sécurité du vote, voire changer la valeur contenue dans le bulletin. Pour pallier à ce problème, nous nous reposons sur l'utilisation de bulletins pré-calculés et imprimés sur une feuille de vote par le greffier de l'élection. Chaque électeur reçoit une feuille de vote individuelle de la part du greffier et peut la scanner pour l'auditer et/ou envoyer son bulletin à l'urne. de cette manière, un outil corrompu ne sera pas en mesure de "voir" ou 'modifier" le bulletin du votant. L'audit de la feuille est une opération simple qu'il est possible de déléguer à une entité de confiance sans perdre en sécurité. L'utilisation d'une feuille de vote nous permet en plus de couper plus efficacement que dans les versions précédentes du protocole Belenios le lien entre un bulletin et son votant, puisque la feuille de vote n'est pas foncièrement liée à un votant en particulier, elle peut être distribuée ou récupérée aléatoirement par les membres de la liste électorale de l'élection. Outre la spécification du protocole, nous fournissons également une analyse extensive de la sécurité de celui-ci effectuée avec ProVerif. Nous considérons ainsi tous les cas de corruption et leur combinaisons possibles (par exemple, le cas où l'outil du votant et corrompu et l'urne électronique également, ou bien celui où le votant perd sa feuille de vote et où le greffier de l'élection est sous le contrôle de l'attaquant...) pour définir quels sont les cas de corruption maximaux sous lesquels notre protocole reste sécurisé et à quels types d'attaques il est vulnérable. L'analyse nous a permis d'établir précisément les cas où notre protocole garantissait à la fois la vérifiabilité te la confidentialité.

- **Deux théorèmes permettant l'automatisation des preuves de vérifiabilité avec ProVerif :** la vérifiabilité d'un schéma de vote peut s'établir selon plusieurs niveaux. Au niveau *individuel*, chacun des électeurs doit pouvoir, aisément, être convaincu du fait que son bulletin fait partie du scrutin final. Au niveau *collectif*, il doit être possible de contrôler que le scrutin prend bien en compte tous les bulletins, valides, soumis pour cette élection et que le résultat n'a pas été altéré. Enfin, il doit être établi que chacun des participants était autorisé à voter ou à défaut, que si des votants sont corrompus, le nombre de bulletins compromis faisant partie du décompte final n'excède pas le nombre de votants corrompus. L'ensemble de ces propriétés a déjà fait l'objet d'une formalisation qui, en revanche, n'est pas exprimable dans les outils automatiques de preuves. ProVerif ne peut pas "compter" et Tamarin ne peut pas gérer la théorie équationnelle derrière le chiffrement asymétrique sur lequel repose Belenios VS. Au vu du nombre important de cas de corruption que nous avions à étudier, il était inenvisageable de faire ces preuves manuellement. Des études préalables ont proposé une automatisation de la preuve d'une vérifiabilité plus faible que celle qui nous intéressait pour Belenios VS. Elles reposaient sur la preuve de certaines propriétés de trace qui impliquaient cette vérifiabilité plus faible. En nous inspirant de cette approche, nous avons été en mesure de fournir deux ensembles de propriétés de trace, aisément exprimables en ProVerif, qui, si elles sont vérifiées, impliquent que le protocole est vérifiable. Ce travail se matérialise par la preuve de deux théorèmes. Ces ensembles de propriétés sont suffisamment génériques pour pouvoir être appliqués à d'autres protocoles de vote électronique. Nous avons ainsi pu appliquer nos deux théorèmes à l'étude de sécurité de Belenios VS?

Paiement mobile : un protocole d'application de paiement fondé sur la tokenisation

Les téléphones portables ont été utilisés dès leur lancement pour effectuer ou confirmer des transactions. On pense notamment au paiement par SMS, très utilisé depuis plus d'une décennie dans les pays d'Afrique, au protocole 3D-Secure, qui permet de confirmer une transaction sur un canal autre que web, ou bien à la pléthore d'applications de paiement mobile lancées au cours des cinq dernières années. Ce sont ces applications, autorisant un paiement NFC au cours duquel le téléphone émule une carte bancaire sans contact, qui nous ont intéressées au cours de cette thèse.

En attestent les nombreuses application de paiement existantes (Apple Pay, Samsung Pay, Android Pay, Paylib, Orange Cash, pour en citer quelques unes), la compétition est rude dans le marché du paiement mobile basé sur l'émulation de carte bancaire. Et cela va de pair avec les failles de sécurité qui sont chaque jour relevées par la communauté. L'un des principaux freins à la sécurisation globale de ces applications bancaires réside dans le fait qu'il n'existe pas, à ce jour, de standard établissant clairement le cahier des charges technique et structurel de telles applications, tout juste quelques recommandations ou framework. D'autre part, la sécurisation dans le monde bancaire est traditionnellement portée par une logique de "sécurité par l'obscurité". en d'autres termes, les spécifications fonctionnelles des applications propriétaires ne sont pas publiques et il est très difficile, voire impossible, de les obtenir pour en juger la sécurité.

Nous avons donc proposé un protocole d'application de paiement mobile de bout en bout, décrivant toute la chaîne de transaction, dont la spécification est publique. Le protocole en question est entièrement compatible avec l'infrastructure existante qui repose sur le standard EMV, et ne nécessite donc pas de mise à jour particulière au niveau des terminaux marchands.

Contributions

Nous apportons principalement deux contributions dans le domaine du paiement mobile :

- **Fournir la spécification ouverte de bout en bout d'une application de paiement mobile :** à notre connaissance, nous avons proposé la toute première spécification publique de bout en bout pour une application de paiement mobile. Le protocole repose sur l'utilisation d'alias éphémères qui remplacent les données de paiement originales. Le consortium EMVCo a publié en 2014 un framework technique sur l'utilisation de ces alias, les tokens. Toutefois, ce papier définit simplement les entités et potentielles intégrations à l'écosystème de paiement d'une solution reposant sur l'utilisation de tokens sans faire état d'un protocole à proprement parler. Nous ne pouvons tirer de ce document des informations spécifiques concernant les différents échanges entre payeur, receveur, banques et applications, la définition de ces échanges étant laissée à l'implémenteur de la solution, d'où notre intérêt dans la proposition détaillée de notre protocole de paiement. Du point de vue utilisateur, l'expérience est très simple. Après avoir souscrit au service, le client peut recharger son téléphone en tokens auprès d'un serveur. Ceux-ci arrivent chiffrés et ne peuvent être déchiffrés, et donc utilisés, qu'après validation active de la part du client, la clef de chiffrement ne pouvant être générée que depuis une élément sécurisé (une carte SIM par exemple) à accès restreint. cette opération de rechargement se fait en ligne, avant même qu'une transaction ne se fasse. Au moment d'une transaction, l'utilisateur valide les paramètres de celle-ci (montant et marchand) depuis son téléphone et déclenche le déchiffrement d'un token ainsi que la génération d'une valeur unique de transaction liant un token spécifique à une montant et un marchand, réduisant de fait la fenêtre d'attaque contre l'application. Si un token est volé, il devra être utilisé pour le même montant et le même marchand avant que l'utilisateur ne valide une autre transaction.

- **Prouver la sécurité d'un protocole industriel avec l'outil de preuve Tamarin :** Le standard

EMV a fait l'objet de nombreuses études sur la sécurité des cartes bancaires qui le suivaient, toutefois, celles-ci présupposaient que lesdites cartes ne soient pas compromises, ce qui est une hypothèse raisonnable concernant les cartes à puce, mais quelque peu forte concernant les applications de paiement mobile reposant sur l'émulation de cartes bancaires. en effet, l'environnement n'est pas le même, la carte bleue est passive et non connectée tandis que les données de paiement sont stockées sur un téléphone connecté et en mesure d'interagir avec d'autres objets. Nous avons donc effectué la preuve de notre protocole en présupposant un modèle d'attaquant plus fort que celui traditionnellement utilisé dans le monde bancaire. Notre protocole utilisant des compteurs, il a été nécessaire d'effectuer la preuve formelle de sécurité en Tamarin, alimentant ainsi les exemples de preuves effectuées avec ce nouvel outil. Il a résulté de notre analyse que les garanties de sécurité étaient les même que celles détaillées dans le standard EMV des cartes bancaires, le tout avec une dimension supplémentaire concernant la vie privée de l'utilisateur. en effet, l'utilisation de tokens permet l'intraçabilité des transactions du côté du marchand.

Plan de la thèse

Après une premier chapitre introductif comprenant notamment une introduction à la modélisation de protocoles dans le modèle symbolique, ce manuscrit de thèse est divisé en deux parties indépendantes portant sur les deux cas d'usage qui ont fait l'objet de ces trois ans de travail : le vote électronique et le paiement mobile.

La partie sur le vote électronique s'articule autour de trois chapitres. Nous commençons par décrire le protocole de vote Belenios VS et ses apports par rapport à l'état de l'art. Le chapitre suivant détaille les deux théorèmes qui nous ont servis à automatiser les preuves de la vérifiabilité du protocole ainsi que leur démonstration. Le dernier chapitre fournit enfin une analyse de sécurité détaillée de Belenios VS explicitant les garanties de vérifiabilité et de confidentialité en fonction du modèle de corruption considéré.

La seconde partie commence par une description du paysage des applications de paiement mobiles et de la gestion de leur sécurité. Nous proposons ensuite une spécification d'une protocole d'application de paiement mobile pour enfin terminer sur l'analyse de sécurité en Tamarin.

L'ensemble de la thèse fait l'objet d'une conclusion finale résumant les contributions et proposant des pistes de recherche à venir découlant du travail effectué.