Verifying privacy-type properties of electronic voting protocols ⋆

Stéphanie Delaune [a,b], Steve Kremer [b], Mark Ryan [a]

[a] *School of Computer Science, University of Birmingham, UK*

[b] *LSV, CNRS & ENS Cachan & INRIA Futurs projet SECSI, France*

Abstract

Electronic voting promises the possibility of a convenient, efficient and secure facility for recording and tallying votes in an election. Recently highlighted inadequacies of implemented systems have demonstrated the importance of formally verifying the underlying voting protocols. We study three privacy-type properties of electronic voting protocols: in increasing order of strength, they are vote-privacy, receipt-freeness, and coercion-resistance.

We use the applied pi calculus, a formalism well adapted to modelling such protocols, which has the advantages of being based on well-understood concepts. The privacy-type properties are expressed using observational equivalence and we show in accordance with intuition that coercion-resistance implies receipt-freeness, which implies vote-privacy.

We illustrate our definitions on three electronic voting protocols from the literature. Ideally, these three properties should hold even if the election officials are corrupt. However, protocols that were designed to satisfy receipt-freeness or coercion-resistance may not do so in the presence of corrupt officials. Our model and definitions allow us to specify and easily change which authorities are supposed to be trustworthy.

Key words: voting protocol, applied pi calculus, formal methods, privacy and anonymity properties.

⋆ This work has been partly supported by the EPSRC projects EP/E029833 (*Verifying Properties in Electronic Voting Protocols*), EP/E040829/1 (*Verifying anonymity and privacy properties of security protocols*) and EP/D076625/2 (*UbiVal: Fundamental Approaches to Validation of Ubiquitous Computing Applications and Infrastructures*), and the ARA SESUR project AVOTÉ and the ARTIST2 NoE.

1 Introduction

Electronic voting protocols. Electronic voting promises the possibility of a convenient, efficient and secure facility for recording and tallying votes. It can be used for a variety of types of elections, from small committees or on-line communities through to full-scale national elections. Electronic voting protocols are formal protocols that specify the messages sent between the voters and administrators. Such protocols have been studied for several decades. They offer the possibility of abstract analysis of the voting system against formally-stated properties.

In this paper, we recall some existing protocols which have been developed over the last decades, and some of the security properties they are intended to satisfy. We focus on privacy-type properties. We present a framework for analysing those protocols and determining whether they satisfy the properties.

From the protocol point of view, the main challenge in designing an election system is to guarantee *vote-privacy*. We may distinguish three main kinds of protocols in the literature, classified according to the mechanism they employ to guarantee privacy. In *blind signature schemes* [16,25,31,36], the voter first obtains a token, which is a message blindly signed by the administrator and known only to the voter herself. The signature of the administrator confirms the voter's eligibility to vote. She later sends her vote anonymously, with this token as proof of eligibility. In schemes using *homomorphic encryption* [7,28], the voter cooperates with the administrator in order to construct an encryption of her vote. The administrator then exploits homomorphic properties of the encryption algorithm to compute the encrypted tally directly from the encrypted votes. A third kind of scheme uses randomisation (for example by mixnets) to mix up the votes so that the link between voter and vote is lost [17,18]. Our focus in this paper is on protocols of the first type, although our methods can probably be used for protocols of the second type. Because it involves mixes, which are probabilistic, the third type is hard to address with our methods that are purely non-deterministic.

Properties of electronic voting protocols. Some properties commonly sought for voting protocols are the following:

- Eligibility: only legitimate voters can vote, and only once.
- Fairness: no early results can be obtained which could influence the remaining voters.
- Individual verifiability: a voter can verify that her vote was really counted.
- Universal verifiability: the published outcome really is the sum of all the votes.
- Vote-privacy: the fact that a particular voter voted in a particular way is not revealed to anyone.

- Receipt-freeness: a voter does not gain any information (a *receipt*) which can be used to prove to a coercer that she voted in a certain way.
- Coercion-resistance: a voter cannot cooperate with a coercer to prove to him that she voted in a certain way.

The last three of these are broadly *privacy-type* properties since they guarantee that the link between the voter and her vote is not revealed by the protocol.

The weakest of the three, called *vote-privacy*, roughly states that the fact that a voter voted in a particular way is not revealed to anyone. When stated in this simple way, however, the property is in general false, because if all the voters vote unanimously then everyone will get to know how everyone else voted. The formalisation we give in this paper in fact says that no party receives information which would allow them to distinguish one situation from another one in which two voters swap their votes.

Receipt-freeness says that the voter does not obtain any artefact (a "receipt") which can be used later to prove to another party how she voted. Such a receipt may be intentional or unintentional on the part of the designer of the system. Unintentional receipts might include nonces or keys which the voter is given during the protocol. Receipt-freeness is a stronger property than privacy. Intuitively, privacy says that an attacker cannot discern how a voter votes from any information that the voter necessarily reveals during the course of the election. Receipt-freeness says the same thing even if the voter voluntarily reveals additional information.

Coercion-resistance is the third and strongest of the three privacy properties. Again, it says that the link between a voter and her vote cannot be established by an attacker, this time even if the voter cooperates with the attacker during the election process. Such cooperation can include giving to the attacker any data which she gets during the voting process, and using data which the attacker provides in return. When analysing coercion-resistance, we assume that the voter and the attacker can communicate and exchange data at any time during the election process. Coercion-resistance is intuitively stronger than receipt-freeness, since the attacker has more capabilities.

Of course, the voter can simply tell an attacker how she voted, but unless she provides convincing evidence the attacker has no reason to believe her. Receipt-freeness and coercion-resistance assert that she cannot provide convincing evidence.

Coercion-resistance cannot possibly hold if the coercer can physically vote on behalf of the voter. Some mechanism is necessary for isolating the voter from the coercer at the moment she casts her vote. This can be realised by a voting booth, which we model here as a private and anonymous channel between the voter and the election administrators.

Note that in literature the distinction between receipt-freeness and coercion-resistance is not very clear. The definitions are usually given in natural language and are insufficiently precise to allow comparison. The notion of receipt-freeness first appeared in the work of Benaloh and Tuinstra [8]. Since then, several schemes [8,40] were proposed in order to meet the condition of receipt-freeness, but later shown not to satisfy it. One of the reasons for such flaws is that no formal definition of receipt-freeness has been given. The situation for coercion-resistance is similar. Systems have been proposed aiming to satisfy it; for example, Okamoto [41] presents a system resistant to interactive coercers, thus aiming to satisfy what we call coercion-resistance, but this property is stated only in natural language. Recently, a rigorous definition in a computational model has been proposed by Juels *et al.* for coercion-resistance [32]. We present in this paper what we believe to be the first "formal methods" definition of receipt-freeness and coercion-resistance. It is difficult to compare our definition and the one proposed by Juels *et al.* [32] due to the inherently different models.

Verifying electronic voting protocols. Because security protocols in general are notoriously difficult to design and analyse, formal verification techniques are particularly important. In several cases, protocols which were thought to be correct for several years have, by means of formal verification techniques, been discovered to have major flaws. Our aim in this paper is to use and develop verification techniques, focusing on the three privacy-type properties mentioned above. We choose *the applied pi calculus* [1] as our basic modelling formalism, which has the advantages of being based on well-understood concepts. The applied pi calculus has a family of proof techniques which we can use, and it is partly supported by the ProVerif tool [9]. Moreover, the applied pi calculus allows us to reason about equational theories in order to model the wide variety of cryptographic primitives often used in voting protocols.

As it is often done in protocol analysis, we assume the Dolev-Yao abstraction: cryptographic primitives are assumed to work perfectly, and the attacker controls the public channels. The attacker can see, intercept and insert messages on public channels, but can only encrypt, decrypt, sign messages or perform other cryptographic operations if he has the relevant key. In general, we assume that the attacker also controls the election officials, since the protocols we investigate are supposed to be resistant even if the officials are corrupt. Some of the protocols explicitly require a trusted device, such as a smart card; we do not assume that the attacker controls those devices.

How the properties are formalised. As already mentioned, the vote-privacy property is formalised as the assertion that the attacker does not receive information which enables him to distinguish a situation from another one in which two voters swap their votes. In other words, the attacker cannot distinguish a situation in which Alice votes a and Bob votes b, from another one in which they vote the other way around. This is formalised as an observational equivalence property in applied pi.

Receipt-freeness is also formalised as an observational equivalence. Intuitively, a protocol is receipt-free if the attacker cannot detect a difference between Alice voting in the way he instructed, and her voting in some other way, provided Bob votes in the complementary way each time. As in the case of privacy, Bob's vote is required to prevent the observer seeing a different number of votes for each candidate. Alice cooperates with the attacker by sharing secrets, but the attacker cannot interact with Alice to give her some prepared messages.

Coercion-resistance is formalised as an observational equivalence too. In the case of coercion-resistance, the attacker (which we may also call the coercer) is assumed to communicate with Alice during the protocol, and can prepare messages which she should send during the election process. This gives the coercer much more power.

Ideally, these three properties should hold even if the election officials are corrupt. However, protocols that were designed to satisfy vote-privacy, receipt-freeness or coercion-resistance do not necessarily do so in the presence of corrupt officials. Our model and definitions allow us to specify and easily change which authorities are supposed to be trustworthy.

Related properties and formalisations. The idea of formalising privacy-type properties as some kind of observational equivalence in a process algebra or calculus goes back to the work of Schneider and Sidiropoulos [43]. Similar ideas have been used among others by Fournet and Abadi [24], Mauw *et al.* [37] as well as Kremer and Ryan [35]. Other formalizations of anonymity are based on epistemic logics, e.g. [27]. All of these definitions are mainly concerned with *possibilistic* definitions of anonymity. It is also possible to define *probabilistic* anonymity, such as in [42,45,27,12], which gives a more fine-grained characterisation of the level of anonymity which has been achieved. In [21,44,13], information-theoretic measures have been proposed to quantify the degree of anonymity. In this paper we only focus on *possibilistic* flavours of privacy-type properties and assume that channels are anonymous (without studying exactly how these channels are implemented).

Receipt-freeness and coercion-resistance are more subtle than simple privacy. They involve the idea that the voter cannot *prove* how she voted to the attacker. This is a special case of incoercible multi-party computation, which has been explored in the computational security setting [11]. Similarly to their definition, we define incoercibility as the ability to present the coercer with fake data which matches the public transcript as well as the real data. Our definition specialises the setting to electronic voting, and is designed for a Dolev-Yao-like model.

Independently of our work, Jonker and de Vink [29] give a logical characterisation of the notion of receipt in electronic voting processes. Jonker and Pieters [30] also define receipt-freeness in epistemic logic. However, while these formalisms may be appealing to reason about the property, they seem less suited for modelling the protocol and attacker capabilities. These logics are geared to expressing properties rather than operational steps of a protocol. Thus, modelling protocols using epistemic-logic-based approaches is tedious and requires a high degree of expertise. Baskar *et al.* [5] present a promising approach defining an epistemic logic for a protocol language.

The "inability to prove" character of coercion-resistance and receipt-freeness is also shared by the property called *abuse-freeness* in contract-signing protocols. A contract-signing protocol is abuse-free if signer Alice cannot prove to an observer that she is in a position to determine the outcome of the contract. Abuse-freeness has been formalised in a Dolev-Yao-like setting [33] as the ability to provide a message that allows the observer to test whether Alice is in such a position. This notion of test is inspired by static equivalence of the applied pi calculus. However, this notion of test is purely *offline*, which is suitable for abuse-freeness. In our formalization the voter may provide data that allows an active adversary to distinguish two processes which yields a more general notion of receipt (probably too general for abuse-freeness).

To the best of our knowledge, our definitions constitute the first observational equivalence formalisations of the notion of *not being able to prove* in the formal methods approach to security.

Electronic voting in the real world. Governments the world over are trialling and adopting electronic voting systems, and the security aspects have been controversial. For example, the electronic voting machines used in recent US elections have been fraught with security problems. Researchers [34] have analysed the source code of the Diebold machines used in 37 US states. This analysis has produced a catalogue of vulnerabilities and possible attacks. More recent work [22] has produced a security study of the Diebold AccuVote-TS voting machine, including both hardware and software. The results shows that it is vulnerable to very serious attacks. For example, an attacker who gets physical access to a machine or its removable memory card for as little as one minute could install malicious code, which could steal votes undetectably, modifying all records, logs, and counters to be consistent with the fraudulent vote count it creates. They also showed how an attacker could create malicious code that spreads automatically from machine to machine during normal election activities. In another study, a Dutch voting machine was reprogrammed to play chess, rather than count votes, which resulted in the machine being removed from use [26].

These real-world deployments do not rely on the kind of formal protocols studied in this paper, and therefore our work has no direct bearing on them. The protocols studied here are designed to ensure that vote stealing is cryptographically impossible, and the properties of individual and universal verifiability provide guarantee that voters can verify the outcome of the election themselves. It is hoped that work such as ours in proving the security properties of such protocols will promote their take-up by makers of electronic voting equipment. If deployed, these protocols would—at least to some extent—remove the requirement to trust the hardware and software used by election officials, and even to trust the officials themselves.

This paper. We recall the basic ideas and concepts of the applied pi calculus, in Section 2. Next, in Section 3, we present the framework for formalising voting protocols from the literature, and in Section 4 we show how the three privacy-like properties are formalised. Also in Section 4, we investigate the relationships between the properties and we show that the expected implications hold between them. In Sections 5, 6 and 7 we recall three voting protocols from the literature, and show how they can be formalised in our framework. We analyse which of the properties they satisfy.

Some of the results have been published in two previous papers [35,19]. This paper extends and clarifies our results, provides more examples, better explanations, additional case studies and includes proofs. In particular, our definition of coercion-resistance in this paper is much simpler than our previous definition [19], where we relied on a notion we called *adaptive simulation*. That notion turned out to have some counter-intuitive properties, and we have removed it.

2 The applied pi calculus

The applied pi calculus [1] is a language for describing concurrent processes and their interactions. It is based on the pi calculus, but is intended to be less pure and therefore more convenient to use. The applied pi calculus is, in some sense, similar to the spi calculus [2]. The key difference between the two formalisms concerns the way that cryptographic primitives are handled. The spi calculus has a fixed set of primitives built-in (symmetric and public-key encryption), while the applied pi calculus allows one to define less usual primitives (often used in electronic voting protocols) by means of an equational theory. The applied pi calculus has been used to study a variety of security protocols, such as a private authentication protocol [24] or a key establishment protocol [3].

To describe processes in the applied pi calculus, one starts with a set of *names* (which are used to name communication channels or other atomic data), a set of *variables*, and a *signature* Σ which consists of the *function symbols* which will be used to define *terms*. In the case of security protocols, typical function symbols will include enc for encryption, which takes plaintext and a key and returns the corresponding ciphertext, and dec for decryption, taking ciphertext and a key and returning the plaintext. Terms are defined as names, variables, and function symbols applied to other terms. Terms and function symbols are sorted, and of course function symbol application must respect sorts and arities. By the means of an equational theory E we describe the equations which hold on terms built from the signature. We denote $=_{\mathsf{E}}$ the equivalence relation induced by E. A typical example of an equational theory useful for cryptographic protocols is $\mathsf{dec}(\mathsf{enc}(x, k), k) = x$. In this theory, the terms $T_1 = \mathsf{dec}(\mathsf{enc}(\mathsf{enc}(n, k_1), k_2), k_2)$ and $T_2 = \mathsf{enc}(n, k_1)$ are equal, we have $T_1 =_{\mathsf{E}} T_2$ (while obviously the syntactic equality $T_1 = T_2$ does not hold). Two terms are related by $=_{\mathsf{E}}$ only if that fact can be derived from the equations in E. When the set of variables occurring in a term T is empty, we say that T is *ground*.

In the applied pi calculus, one has *plain processes* and *extended processes*. Plain processes are built up in a similar way to processes in the pi calculus, except that messages can contain terms (rather than just names). In the grammar described below, M and N are terms, n is a name, x a variable and u is a metavariable, standing either for a name or a variable.

$P, Q, R :=$	plain processes
$\quad 0$	null process
$\quad P \mid Q$	parallel composition
$\quad !P$	replication
$\quad \nu n.P$	name restriction
$\quad$ if $M = N$ then P else Q	conditional
$\quad \mathsf{in}(u, x).P$	message input
$\quad \mathsf{out}(u, N).P$	message output

We use the notation $\mathsf{in}(u, =M)$ to test whether the input on u is equal (modulo E) to the term M (if it doesn't, the process blocks). Moreover, we sometimes use tuples of terms, denoted by parentheses, while keeping the equational theory for these tuples implicit.

Extended processes add *active substitutions* and restriction on variables:

$A, B, C :=$	extended processes
$\quad P$	plain process

$A \mid B$	parallel composition
$\nu n.A$	name restriction
$\nu x.A$	variable restriction
$\{^M/_x\}$	active substitution

$\{^M/_x\}$ is the substitution that replaces the variable x with the term M. Active substitutions generalise “let”. The process $\nu x.(\{^M/_x\} \mid P)$ corresponds exactly to the process “let $x = M$ in P”. As usual, names and variables have scopes, which are delimited by restrictions and by inputs. We write $fv(A)$, $bv(A)$, $fn(A)$ and $bn(A)$ for the sets of free and bound variables and free and bound names of A, respectively. We also assume that, in an extended process, there is at most one substitution for each variable, and there is exactly one when the variable is restricted. We say that an extended process is *closed* if all its variables are either bound or defined by an active substitution.

Active substitutions are useful because they allow us to map an extended process A to its *frame* $\phi(A)$ by replacing every plain process in A with 0. A frame is an extended process built up from 0 and active substitutions by parallel composition and restriction. The frame $\phi(A)$ can be viewed as an approximation of A that accounts for the static knowledge A exposes to its environment, but not A's dynamic behaviour.

Example 1 *For instance, consider the extended processes $A_1 = \{^{M_1}/_{x_1}\} \mid \{^{M_2}/_{x_2}\} \mid P_1$ and $A_2 = \{^{M_1}/_{x_1}\} \mid \{^{M_2}/_{x_2}\} \mid P_2$. Even if these two processes are different from the point of view of their dynamic behaviour, the frames $\phi(A_1)$ and $\phi(A_2)$ are equal. This witnesses the fact that A_1 and A_2 have the same static knowledge.*

The domain of a frame φ, denoted by $\mathrm{dom}(\varphi)$, is the set of variables for which φ defines a substitution (those variables x for which φ contains a substitution $\{^M/_x\}$ not under a restriction on x).

An *evaluation context* $C[\_]$ is an extended process with a hole instead of an extended process. Structural equivalence, noted $\equiv$, is the smallest equivalence relation on extended processes that is closed under α-conversion on names and variables, by application of evaluation contexts, and such that

PAR-0	$A \mid 0 \equiv A$	REPL	$!P \equiv P \mid !P$
PAR-A	$A \mid (B \mid C) \equiv (A \mid B) \mid C$	REWRITE	$\{^M/_x\} \equiv \{^N/_x\}$
PAR-C	$A \mid B \equiv B \mid A$		if $M =_{\mathsf{E}} N$
NEW-0	$\nu n.0 \equiv 0$	ALIAS	$\nu x.\{^M/_x\} \equiv 0$
NEW-C	$\nu u.\nu v.A \equiv \nu v.\nu u.A$	SUBST	$\{^M/_x\} \mid A \equiv \{^M/_x\} \mid A\{^M/_x\}$

NEW-PAR $\quad A \mid \nu u.B \;\equiv\; \nu u.(A \mid B) \quad$ if $u \notin fn(A) \cup fv(A)$

Example 2 *Consider the following process P:*

$$\nu s.\nu k.(\mathsf{out}(c_1, \mathsf{enc}(s,k)) \mid \mathsf{in}(c_1, y).\mathsf{out}(c_2, \mathsf{dec}(y,k))).$$

The first component publishes the message $\mathsf{enc}(s,k)$ by sending it on c_1. The second receives a message on c_1, uses the secret key k to decrypt it, and forwards the resulting plaintext on c_2. The process P is structurally equivalent to the following extended process A:

$$A = \nu s, k, x_1.\Big(\mathsf{out}(c_1, x_1) \mid \mathsf{in}(c_1, y).\mathsf{out}(c_2, \mathsf{dec}(y,k)) \mid \{^{\mathsf{enc}(s,k)}/_{x_1}\}\Big)$$

We have $\phi(A) = \nu s, k, x_1.\{^{enc(s,k)}/_{x_1}\} \equiv 0$ (since x_1 is under a restriction).

The following lemma will be useful in the remainder of the paper.

Lemma 3 *Let $C_1 = \nu\tilde{u}_1.(\_ \mid B_1)$ and $C_2 = \nu\tilde{u}_2.(\_ \mid B_2)$ be two evaluation contexts such that $\tilde{u}_1 \cap (fv(B_2) \cup fn(B_2)) = \emptyset$ and $\tilde{u}_2 \cap (fv(B_1) \cup fn(B_1)) = \emptyset$. We have that $C_1[C_2[A]] \equiv C_2[C_1[A]]$ for any extended process A.*

PROOF. Let A be an extended process. We have that

$$\begin{aligned}
C_1[C_2[A]] &\equiv \nu\tilde{u}_1.(\nu\tilde{u}_2.(A \mid B_2) \mid B_1) & \\
&\equiv \nu\tilde{u}_2.\nu\tilde{u}_1.((A \mid B_1) \mid B_2) & \text{since } \tilde{u}_2 \notin fv(B_1) \cup fn(B_1) \\
&\equiv \nu\tilde{u}_2.(\nu\tilde{u}_1.(A \mid B_1) \mid B_2) & \text{since } \tilde{u}_1 \notin fv(B_2) \cup fn(B_2) \\
&\equiv C_2[C_1[A]] & \Box
\end{aligned}$$

2.2 Semantics

The operational semantics of processes in the applied pi calculus is defined by structural rules defining two relations: *structural equivalence* (briefly described in Section 2.1) and *internal reduction*, noted $\rightarrow$. Internal reduction $\rightarrow$ is the smallest relation on extended processes closed under structural equivalence and application of evaluation contexts such that

(COMM)	$\mathsf{out}(a,x).P \mid \mathsf{in}(a,x).Q \rightarrow P \mid Q$
(THEN)	if $M = M$ then P else $Q \rightarrow P$
(ELSE)	if $M = N$ then P else $Q \rightarrow Q$
	for any ground terms M and N such that $M \neq_{\mathsf{E}} N$.

The operational semantics is extended by a *labelled* operational semantics enabling us to reason about processes that interact with their environment. Labelled operational semantics defines the relation $\xrightarrow{\alpha}$ where α is either an input, or the output of a channel name or a variable of base type.

$$\text{(IN)} \qquad \mathsf{in}(a,x).P \xrightarrow{in(a,M)} P\{^M/_x\}$$

$$\text{(OUT-ATOM)} \qquad \mathsf{out}(a,u).P \xrightarrow{out(a,u)} P$$

$$\text{(OPEN-ATOM)} \qquad \frac{A \xrightarrow{out(a,u)} A' \qquad u \neq a}{\nu u.A \xrightarrow{\nu u.out(a,u)} A'}$$

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

$$\text{(PAR)} \qquad \frac{A \xrightarrow{\alpha} A' \quad bv(\alpha) \cap fv(B) = bn(\alpha) \cap fn(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

$$\text{(STRUCT)} \qquad \frac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad A' \equiv B'}{A \xrightarrow{\alpha} A'}$$

Note that the labelled transition is not closed under application of evaluation contexts. Moreover the output of a term M needs to be made "by reference" using a restricted variable and an active substitution.

Example 4 *Consider the process P defined in Example 2. We have*

$$\begin{array}{cl}
P \quad \equiv & \nu s, k, x_1.(\mathsf{out}(c_1, x_1) \mid \mathsf{in}(c_1, y).\mathsf{out}(c_2, \mathsf{dec}(y,k)) \mid \{^{\mathsf{enc}(s,k)}/_{x_1}\}) \\
\xrightarrow{\nu x_1.out(c_1,x_1)} & \nu s, k.(\mathsf{in}(c_1, y).\mathsf{out}(c_2, \mathsf{dec}(y,k)) \mid \{^{\mathsf{enc}(s,k)}/_{x_1}\}) \\
\xrightarrow{in(c_1,x_1)} & \nu s, k.(\mathsf{out}(c_2, \mathsf{dec}(x_1,k)) \mid \{^{\mathsf{enc}(s,k)}/_{x_1}\}) \\
\equiv & \nu s, k, x_2.(\mathsf{out}(c_2, x_2) \mid \{^{\mathsf{enc}(s,k)}/_{x_1}\} \mid \{^{\mathsf{dec}(x_1,k)}/_{x_2}\}) \\
\xrightarrow{\nu x_2.out(c_1,x_2)} & \nu s, k.(\{^{\mathsf{enc}(s,k)}/_{x_1}\} \mid \{^{\mathsf{dec}(x_1,k)}/_{x_2}\})
\end{array}$$

Let A be the extended process obtained after this sequence of reduction steps. We have that $\phi(A) \equiv \nu s.\nu k.\{^{enc(s,k)}/_{x_1},\ ^{s}/_{x_2}\}$.

2.3 Equivalences

We can now define what it means for two frames to be *statically equivalent* [1].

Definition 5 (Static equivalence ($\approx_s$)) *Two terms M and N are* equal in the frame *ϕ, written $(M =_{\mathsf{E}} N)\phi$, if, and only if there exists $\tilde{n}$ and a substitution σ such that $\phi \equiv \nu\tilde{n}.\sigma$, $M\sigma =_{\mathsf{E}} N\sigma$, and $\tilde{n} \cap (fn(M) \cup fn(N)) = \emptyset$.*

Two frames ϕ_1 and ϕ_2 are statically equivalent, *$\phi_1 \approx_{\mathsf{E}} \phi_2$, when:*

- $\mathrm{dom}(\phi_1) = \mathrm{dom}(\phi_2)$, *and*
- *for all terms M, N we have that $(M =_{\mathsf{E}} N)\phi_1$ if and only if $(M =_{\mathsf{E}} N)\phi_2$.*

Two extended processes A and B are said to be statically equivalent, denoted by $A \approx_s B$, if we have that $\phi(A) \approx_s \phi(B)$.

Example 6 *Let $\varphi_0 = \nu k.\sigma_0$ and $\varphi_1 = \nu k.\sigma_1$ where $\sigma_0 = \{^{\mathsf{enc}(s_0,k)}/_{x_1}, ^{k}/_{x_2}\}$, $\sigma_1 = \{^{\mathsf{enc}(s_1,k)}/_{x_1}, ^{k}/_{x_2}\}$ and s_0, s_1 and k are names. Let E be the theory defined by the axiom $\mathsf{dec}(\mathsf{enc}(x,k),k) = x$. We have $\mathsf{dec}(x_1,x_2)\sigma_0 =_{\mathsf{E}} s_0$ but not $\mathsf{dec}(x_1,x_2)\sigma_1 =_{\mathsf{E}} s_0$. Therefore we have $\varphi_0 \not\approx_s \varphi_1$. However, note that we have $\nu k.\{^{\mathsf{enc}(s_0,k)}/_{x_1}\} \approx_s \nu k.\{^{\mathsf{enc}(s_1,k)}/_{x_1}\}$.*

Definition 7 (Labelled bisimilarity ($\approx_\ell$)) Labelled bisimilarity *is the largest symmetric relation $\mathcal{R}$ on closed extended processes, such that $A \mathrel{\mathcal{R}} B$ implies*

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

The definition of labelled bisimilarity is like the usual definition of bisimilarity, except that at each step one additionally requires that the processes are statically equivalent. It has been shown that labelled bisimilarity coincides with observational equivalence [1]. We prefer to work with labelled bisimilarity, rather than observational equivalence, because proofs for labelled bisimilarity are generally easier. Labelled bisimilarity can be used to formalise many security properties, in particular anonymity properties, such as those studied in this paper.

When we model protocols in applied pi calculus, we model the honest parties as processes. The dishonest parties are considered to be under the control of the attacker, and are not modelled explicitly. The attacker (together with any parties it controls) form the environment in which the honest processes run. This arrangement implies that we consider only one attacker; to put in another way, we consider that all dishonest parties and attackers share information and trust each other, thus forming a single coalition. This arrangement does not allow us to consider attackers that do not share information with each other.

3 Formalising voting protocols

Before formalising security properties, we need to define what is an electronic voting protocol in applied pi calculus. Different voting protocols often have substantial differences. However, we believe that a large class of voting protocols can be represented by processes corresponding to the following structure.

Definition 8 (Voting process) *A voting process is a closed plain process*

$$VP \equiv \nu\tilde{n}.(V\sigma_1 \mid \cdots \mid V\sigma_n \mid A_1 \mid \cdots \mid A_m).$$

The $V\sigma_i$ are the voter processes, the A_js the election authorities which are required to be honest and the $\tilde{n}$ are channel names. We also suppose that $v \in \mathrm{dom}(\sigma_i)$ is a variable which refers to the value of the vote. We define an evaluation context S which is as VP, but has a hole instead of two of the $V\sigma_i$.

In order to prove a given property, we may require some of the authorities to be honest, while other authorities may be assumed to be corrupted by the attacker. The processes $A_1, \ldots, A_m$ represent the authorities which are required to be honest. The authorities under control of the attacker need not be modelled, since we consider any possible behaviour for the attacker (and therefore any possible behaviour for corrupt authorities). In this case the communications channels are available to the environment.

We have chosen to illustrate our definition with three classical electronic voting protocols of the literature: a protocol due to Fujioka *et al.* [25], a protocol due to Okamoto [40] and one due to Lee *et al.* [36]. After a brief and informal description of those protocols, we formalise them in the applied pi calculus framework in Sections 5, 6 and 7.

4 Formalising privacy-type properties

In this section, we show how the anonymity properties, informally described in the introduction, can be formalised in our setting and we show, in accordance with intuition, that coercion-resistance implies receipt-freeness, which implies privacy. It is rather classical to formalise anonymity properties as some kind of observational equivalence in a process algebra or calculus, going back to the work of Schneider and Sidiropoulos [43]. However, the definition of anonymity properties in the context of voting protocols is rather subtle.

The privacy property aims to guarantee that the link between a given voter V and his vote v remains hidden. Anonymity and privacy properties have been successfully studied using equivalences. However, the definition of privacy in the context of voting protocols is rather subtle. While generally most security properties should hold against an arbitrary number of dishonest participants, arbitrary coalitions do not make sense here. Consider for instance the case where all but one voter are dishonest: as the results of the vote are published at the end, the dishonest voter can collude and determine the vote of the honest voter. A classical trick for modelling anonymity is to ask whether two processes, one in which V_A votes and one in which V_B votes, are equivalent. However, such an equivalence does not hold here as the voters' identities are revealed (and they need to be revealed at least to the administrator to verify eligibility). In a similar way, an equivalence of two processes where only the vote is changed does not hold, because the votes are published at the end of the protocol. To ensure privacy we need to hide the *link* between the voter and the vote and not the voter or the vote itself.

In order to give a reasonable definition of privacy, we need to suppose that at least two voters are honest. We denote the voters V_A and V_B and their votes a, respectively b. We say that a voting protocol respects privacy whenever a process where V_A votes a and V_B votes b is observationally equivalent to a process where V_A votes b and V_B votes a. Formally, privacy is defined as follows.

Definition 9 (Vote-privacy) *A voting protocol respects* vote-privacy *(or just* privacy*) if*

$$S[V_A\{^a/_v\} \mid V_B\{^b/_v\}] \approx_\ell S[V_A\{^b/_v\} \mid V_B\{^a/_v\}]$$

for all possible votes a and b.

The intuition is that if an intruder cannot detect if arbitrary honest voters V_A and V_B swap their votes, then in general he cannot know anything about how V_A (or V_B) voted. Note that this definition is robust even in situations where the result of the election is such that the votes of V_A and V_B are necessarily revealed. For example, if the vote is unanimous, or if all other voters reveal how they voted and thus allow the votes of V_A and V_B to be deduced.

A protocol satisfying privacy also allows arbitrary permutations of votes between voters. For example, we may prove that

$$S[V_A\{^a/_v\} \mid V_B\{^b/_v\} \mid V_C\{^c/_v\}] \approx_\ell S[V_A\{^b/_v\} \mid V_B\{^c/_v\} \mid V_C\{^a/_v\}]$$

as follows:

$$
\begin{aligned}
&S[V_A\{^a/_v\} \mid V_B\{^b/_v\} \mid V_C\{^c/_v\}] \\
&\quad \approx_\ell S[V_A\{^b/_v\} \mid V_B\{^a/_v\} \mid V_C\{^c/_v\}] \text{ using privacy, with } S' = S[\_ \mid V_c\{^c/_v\}] \\
&\quad \approx_\ell S[V_A\{^b/_v\} \mid V_B\{^c/_v\} \mid V_C\{^a/_v\}] \text{ using privacy, with } S'' = S[V_a\{^b/_v\} \mid \_]
\end{aligned}
$$

As already noted, in some protocols the vote-privacy property may hold even if authorities are corrupt, while other protocols may require the authorities to be honest. When proving privacy, we choose which authorities we want to model as honest, by including them in Definition 8 of VP (and hence S).

4.2 Receipt-Freeness

Similarly to privacy, receipt-freeness may be formalised as an observational equivalence. We also formalise receipt-freeness using observational equivalence. However, we need to model the fact that V_A is willing to provide secret information, i.e., the receipt, to the coercer. We assume that the coercer is in fact the attacker who, as usual in the Dolev-Yao model, controls the public channels. To model V_A's communication with the coercer, we consider that V_A executes a voting process which has been modified: any input of base type and any freshly generated names of base type are forwarded to the coercer. We do not forward restricted channel names, as these are used for modelling purposes, such as physically secure channels, e.g. the voting booth, or the existence of a PKI which securely distributes keys (the keys themself are forwarded but not the secret channel name on which the keys are received).

Definition 10 (Process P^{ch}) *Let P be a plain process and ch a channel name. We define P^{ch} as follows:*

- $0^{ch} \mathrel{\hat=} 0$,
- $(P \mid Q)^{ch} \mathrel{\hat=} P^{ch} \mid Q^{ch}$,
- $(\nu n.P)^{ch} \mathrel{\hat=} \nu n.\mathsf{out}(ch, n).P^{ch}$ *when n is name of base type,*
- $(\nu n.P)^{ch} \mathrel{\hat=} \nu n.P^{ch}$ *otherwise,*
- $(\mathsf{in}(u, x).P)^{ch} \mathrel{\hat=} \mathsf{in}(u, x).\mathsf{out}(ch, x).P^{ch}$ *when x is a variable of base type,*
- $(\mathsf{in}(u, x).P)^{ch} \mathrel{\hat=} \mathsf{in}(u, x).P^{ch}$ *otherwise,*
- $(\mathsf{out}(u, M).P)^{ch} \mathrel{\hat=} \mathsf{out}(u, M).P^{ch}$,
- $(!P)^{ch} \mathrel{\hat=} !P^{ch}$,
- *(if $M = N$ then P else Q)^{ch} $\mathrel{\hat=}$ if $M = N$ then P^{ch} else Q^{ch}.*

In the remainder, we assume that $ch \notin fn(P) \cup bn(P)$ before applying the transformation.

Given an extended process A and a channel name ch, we need to define the extended process $A^{\backslash out(ch,\cdot)}$. Intuitively, such a process is as the process A, but hiding the outputs on the channel ch.

Definition 11 (Process $A^{\backslash out(ch,\cdot)}$) *Let A be an extended process. We define the process $A^{\backslash out(ch,\cdot)}$ as $\nu ch.(A \,|!\mathsf{in}(ch, x))$.*

We are now ready to define receipt-freeness. Intuitively, a protocol is receipt-free if, for all voters V_A, the process in which V_A votes according to the intruder's wishes is indistinguishable from the one in which she votes something else. As in the case of privacy, we express this as an observational equivalence to a process in which V_A swaps her vote with V_B, in order to avoid the case in which the intruder can distinguish the situations merely by counting the votes at the end. Suppose the coercer's desired vote is c. Then we define receipt-freeness as follows.

Definition 12 (Receipt-freeness) *A voting protocol is* receipt-free *if there exists a closed plain process V' such that*

- $V'^{\backslash out(chc,\cdot)} \approx_\ell V_A\{^a/_v\}$,
- $S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}] \approx_\ell S[V' \mid V_B\{^c/_v\}]$,

for all possible votes a and c.

As before, the context S in the second equivalence includes those authorities that are assumed to be honest. V' is a process in which voter V_A votes a but communicates with the coercer C in order to feign cooperation with him. Thus, the second equivalence says that the coercer cannot tell the difference between a situation in which V_A genuinely cooperates with him in order to cast the vote c and one in which she pretends to cooperate but actually casts the vote a, provided there is some counterbalancing voter that votes the other way around. The first equivalence of the definition says that if one ignores the outputs V' makes on the coercer channel chc, then V' looks like a voter process V_A voting a.

The first equivalence of the definition may be considered too strong; informally, one might consider that the equivalence should be required only in a particular S context rather than requiring it in any context (with access to all the private channels of the protocol). This would result in a weaker definition, although one which is more difficult to work with. In fact, the variant definition would be only slightly weaker; it is hard to construct a natural example which distinguishes the two possibilities, and in particular it makes no difference to the case studies of later sections. Therefore, we prefer to stick to Definition 12.

According to intuition, if a protocol is receipt-free (for a given set of honest authorities), then it also respects privacy (for the same set):

Proposition 13 *If a voting protocol is receipt-free then it also respects privacy.*

Before we prove this proposition we need to introduce a lemma.

Lemma 14 *Let P be a closed plain process and ch a channel name such that $ch \notin fn(P) \cup bn(P)$. We have $(P^{ch})^{\backslash out(ch,\cdot)} \approx_\ell P$.*

PROOF. (sketch, see Appendix A for details)
We show by induction on the size of P that for any channel name ch such that $ch \notin fn(P) \cup bn(P)$, the equivalence $P^{ch \backslash out(ch,\cdot)} \approx_\ell P$ holds. The base case where $P = 0$ is trivial. Then, we consider the different possibilities for building P. □

PROOF. (of Proposition 13)
By hypothesis, there exists a closed plain process V', such that

- $V'^{\backslash out(chc,\cdot)} \approx_\ell V_A\{^a/_v\}$, and
- $S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}] \approx_\ell S[V' \mid V_B\{^c/_v\}]$.

By applying the evaluation context $\nu chc.(\_ \mid !\mathsf{in}(chc, x))$ on both sides we obtain

$$S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}]^{\backslash out(chc,\cdot)} \approx_\ell S[V' \mid V_B\{^c/_v\}]^{\backslash out(chc,\cdot)}.$$

By using Lemma 3, we obtain that:

- $S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}]^{\backslash out(chc,\cdot)} \equiv S[(V_A\{^c/_v\}^{chc})^{\backslash out(chc,\cdot)} \mid V_B\{^a/_v\}]$,
- $S[V' \mid V_B\{^c/_v\}]^{\backslash out(chc,\cdot)} \equiv S[V'^{\backslash out(chc,\cdot)} \mid V_B\{^c/_v\}]$.

Lastly, thanks to Lemma 14 and the fact that labelled bisimilarity is closed under structural equivalence, we deduce that

$$S[V_A\{^c/_v\} \mid V_B\{^a/_v\}] \approx_\ell S[V'^{\backslash out(chc,\cdot)} \mid V_B\{^c/_v\}].$$

Since we have $V'^{\backslash out(chc,\cdot)} \approx_\ell V_A\{^a/_v\}$, we easily conclude. □

4.3 Coercion-Resistance

Coercion-resistance is a stronger property as we give the coercer the ability to communicate *interactively* with the voter and not only receive information. In this model, the coercer can prepare the messages he wants the voter to send. As for receipt-freeness, we modify the voter process. In the case of coercion-resistance, we give the coercer the possibility to provide the messages the voter should send. The coercer can also decide how the voter branches on *if*-statements.

Definition 15 (Process P^{c_1,c_2}) *Let P be a plain process and c_1, c_2 be channel names. We define P^{c_1,c_2} inductively as follows:*

- $0^{c_1,c_2} \hat{=} 0$,
- $(P \mid Q)^{c_1,c_2} \hat{=} P^{c_1,c_2} \mid Q^{c_1,c_2}$,
- $(\nu n.P)^{c_1,c_2} \hat{=} \nu n.\mathsf{out}(c_1, n).P^{c_1,c_2}$ *when n is a name of base type,*
- $(\nu n.P)^{c_1,c_2} \hat{=} \nu n.P^{c_1,c_2}$ *otherwise,*
- $(\mathsf{in}(u, x).P)^{c_1,c_2} \hat{=} \mathsf{in}(u, x).\mathsf{out}(c_1, x).P^{c_1,c_2}$ *when x is a variable of base type,*
- $(\mathsf{in}(u, x).P)^{c_1,c_2} \hat{=} \mathsf{in}(u, x).P^{c_1,c_2}$ *otherwise,*
- $(\mathsf{out}(u, M).P)^{c_1,c_2} \hat{=} \mathsf{in}(c_2, x).\mathsf{out}(u, x).P^{c_1,c_2}$ *when M is a term of base type and x is a fresh variable,*
- $(\mathsf{out}(u, M).P)^{c_1,c_2} \hat{=} \mathsf{out}(u, M).P^{c_1,c_2}$ *otherwise,*
- $(!P)^{c_1,c_2} \hat{=} !P^{c_1,c_2}$,
- *(if $M = N$ then P else Q)$^{c_1,c_2} \hat{=}$ $\mathsf{in}(c_2, x)$. if x = true then P^{c_1,c_2} else Q^{c_1,c_2} where x is a fresh variable and true is a constant.*

As a first approximation, we could try to define coercion-resistance in the following way: a protocol is coercion-resistant if there is a V' such that

$$S[V_A\{^?/_v\}^{c_1,c_2} \mid V_B\{^a/_v\}] \approx_\ell S[V' \mid V_B\{^c/_v\}]. \tag{1}$$

On the left, we have the coerced voter $V_A\{^?/_v\}^{c_1,c_2}$; no matter what she intends to vote (the "?"), the idea is that the coercer will force her to vote c. On the right, the process V' resists coercion, and manages to vote a. Unfortunately, this characterisation has the problem that the coercer could oblige $V_A\{^?/_v\}^{c_1,c_2}$ to vote $c' \neq c$. In that case, the process $V_B\{^c/_v\}$ would not counterbalance the outcome to avoid a trivial way of distinguishing the two sides.

To enable us to reason about the coercer's choice of vote, we model the coercer's behaviour as a context C that defines the interface c_1, c_2 for the voting process. The context C coerces a voter to vote c. Thus, we can characterise coercion-resistance as follows: a protocol is coercion-resistant if there is a V' such that

$$S[C[V_A\{^?/_v\}^{c_1,c_2}] \mid V_B\{^a/_v\}] \approx_\ell S[C[V'] \mid V_B\{^c/_v\}], \tag{2}$$

where C is a context ensuring that the coerced voter $V_A\{^?/_v\}^{c_1,c_2}$ votes c. The context C models the coercer's behaviour, while the environment models the coercer's powers to observe whether the coerced voter behaves as instructed. We additionally require that the context C does not directly use the channel names $\tilde{n}$ restricted by S. Formally one can ensure that $V_A\{^?/_v\}^{c_1,c_2}$ votes c by requiring that $C[V_A\{^?/_v\}^{c_1,c_2}] \approx_\ell V_A\{^c/_v\}^{chc}$. We actually require a slightly weaker condition, $S[C[V_A\{^?/_v\}^{c1,c2}] \mid V_B\{^a/_v\}] \approx_\ell S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}]$, which results in a stronger property.

Putting these ideas together, we arrive at the following definition:

Definition 16 (Coercion-resistance) *A voting protocol is* coercion-resistant *if there exists a closed plain process V' such that for any $C = \nu c_1.\nu c_2.(\_ \mid P)$ satisfying $\tilde{n} \cap fn(C) = \emptyset$ and $S[C[V_A\{^?/_v\}^{c1,c2}] \mid V_B\{^a/_v\}] \approx_\ell S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}]$, we* *have*

- $C[V']^{\backslash out(chc,\cdot)} \approx_\ell V_A\{^a/_v\}$,
- $S[C[V_A\{^?/_v\}^{c_1,c_2}] \mid V_B\{^a/_v\}] \approx_\ell S[C[V'] \mid V_B\{^c/_v\}]$.

Note that $V_A\{^?/_v\}^{c_1,c_2}$ does not depend on what we put for "?".

The condition that $S[C[V_A\{^?/_v\}^{c1,c2}] \mid V_B\{^a/_v\}] \approx_\ell S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}]$ means that the context C outputs the secrets generated during its computation; this is required so that the environment can make distinctions on the basis of those secrets, as in receipt-freeness. The first bullet point expresses that V' is a voting process for A which fakes the inputs/outputs with C and succeeds in voting a in spite of the coercer. The second bullet point says that the coercer cannot distinguish between V' and the really coerced voter, provided another voter V_B counterbalances.

As in the case of receipt-freeness, the first equivalence of the definition could be made weaker by requiring it only in a particular S context. But we chose not to adopt this extra complication, for the same reasons as given in the case of receipt-freeness.

Remark 17 *The context C models the coercer's behaviour; we can see its role in equivalence (2) as imposing a restriction on the distinguishing power of the environment in equivalence (1). Since the coercer's behaviour is modelled by C while its distinguishing powers are modelled by the environment, it would be useful to write (2) as*

$$C[S[V_A\{^?/_v\}^{c_1,c_2}] \mid V_B\{^a/_v\}]] \approx_\ell C[S[V' \mid V_B\{^c/_v\}]]. \tag{3}$$

Equivalences (2) and (3) are the same (Lemma 3).

According to intuition, if a protocol is coercion-resistant then it respects receipt-freeness too (as before, we keep constant the set of honest authorities):

Proposition 18 *If a voting protocol is coercion-resistant then it also respects receipt-freeness.*

PROOF. Let C be an evaluation context such that $C = \nu c_1.\nu c_2.(\_ \mid P)$ for some plain process P and $S[C[V_A\{^?/_v\}^{c1,c2}] \mid V_B\{^a/_v\}] \approx_\ell S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}]$. Note that such a C can be constructed directly from the vote process V. By hypothesis, we know that there exists a closed plain process V' such that

- $C[V']^{\backslash out(chc,\cdot)} \approx_\ell V_A\{^a/_v\}$,
- $S[C[V_A\{^?/_v\}^{c_1,c_2}] \mid V_B\{^a/_v\}] \approx_\ell S[C[V'] \mid V_B\{^c/_v\}]$.

We need to show that there exists V'' such that

- $V''^{\setminus out(chc,\cdot)} \approx_\ell V_A\{^a/_v\}$,
- $S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}] \approx_\ell S[V'' \mid V_B\{^c/_v\}]$.

Let $V'' = C[V']$. We directly obtain the first requirement. For the second one, we take the hypotheses

- $S[C[V_A\{^?/_v\}^{c_1,c_2}] \mid V_B\{^a/_v\}] \approx_\ell S[C[V'] \mid V_B\{^c/_v\}]$, and
- $S[C[V_A\{^?/_v\}^{c1,c2}] \mid V_B\{^a/_v\}] \approx_\ell S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}]$.

By transitivity of $\approx_\ell$, we obtain $S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}] \approx_\ell S[C[V'] \mid V_B\{^c/_v\}]$. Lastly, we replace $C[V']$ on the right by V''. $\square$

Using the definition of coercion-resistance. To show that a voting protocol satisfies coercion-resistance, it is necessary to give a process V', and it is necessary to show the two bullet points in the definition for all contexts C which satisfy the requirement stated in the definition. In case studies, it is difficult to reason about all possible contexts C, and our analysis is rather informal. In future work, we hope to provide better methods for doing that.

To show that a voting protocol does not satisfy coercion-resistance, it is necessary to show that for all V', there exists a context C for which the bullet points fail. In practice, one may try to give a single C which works for all V'. Since this is a stronger condition, it is sufficient.

5 Protocol due to Fujioka, Okamoto and Ohta

In this section we study a protocol due to Fujioka, Okamoto and Ohta [25]. We first give an informal description of the protocol (see Section 5.1). Then, we show in Section 5.2 how this protocol can be modelled in the applied pi calculus. Lastly, in Section 5.3, we show that the protocol respects privacy. However, the protocol is not receipt-free [40]. The Fujioka, Okamoto and Ohta protocol was also analysed by Nielsen *et al.* [39], but their focus is on properties such as verifiability, eligibility, and fairness, rather than the privacy-type properties of this paper.

5.1 Description

The protocol involves voters, an administrator, verifying that only eligible voters can cast votes, and a collector, collecting and publishing the votes. In comparison with authentication protocols, the protocol also uses some unusual cryptographic primitives such as secure bit-commitment and blind signatures. Moreover, it relies on anonymous channels. We deliberately do not specify the way these channels are handled as any anonymiser mechanism could be suitable depending on the precise context the protocol is used in. One can use MIX-nets introduced by Chaum [14] whose main idea is to permute and modify (by using decryption or re-encryption) some sequence of objects in order to hide the correspondence between elements of the original and the final sequences. Some other implementations may also be possible, e.g. onion routing [46].

A bit-commitment scheme allows an agent A to commit a value v to another agent B without revealing it immediately. Moreover, B is ensured that A cannot change her mind afterwards and that the value she later reveals will be the same as she thinks at the beginning. For this, A encrypts the value v in some way and sends the encryption to B. The agent B is not able to recover v until A sends him the key.

A blind signature scheme allows a requester to obtain a signature of a message m without revealing the message m to anyone, including the signer. Hence, the signer is requested to sign a message blindly without knowing what he signs. This mechanism is very useful in electronic voting protocol. It allows the voter to obtain a signature of her vote by an authority who checks that she has right to vote without revealing it to the authority.

In a first phase, the voter gets a signature on a commitment to his vote from the administrator. To ensure privacy, blind signatures [15] are used, i.e. the administrator does not learn the commitment of the vote.

- Voter V selects a vote v and computes the commitment $x = \xi(v,r)$ using the commitment scheme ξ and a random key r;
- V computes the message $e = \chi(x, b)$ using a blinding function χ and a random blinding factor b;
- V digitally signs e and sends her signature $\sigma_V(e)$ to the administrator A together with her identity;
- A verifies that V has the right to vote, has not voted yet and that the signature is valid; if all these tests hold, A digitally signs e and sends his signature $\sigma_A(e)$ to V;
- V now *unblinds* $\sigma_A(e)$ and obtains $y = \sigma_A(x)$, i.e. a signed commitment to V's vote.

The second phase of the protocol is the actual voting phase.

- V sends y, A's signature on the commitment to V's vote, to the collector C using an anonymous channel;
- C checks correctness of the signature y and, if the test succeeds, enters (ℓ, x, y) into a list as an ℓ-th item.

The last phase of the voting protocol starts, once the collector decides that he received all votes, e.g. after a fixed deadline. In this phase the voters reveal the random key r which allows C to open the votes and publish them.

- C publishes the list (ℓ_i, x_i, y_i) of commitments he obtained;
- V verifies that her commitment is in the list and sends ℓ, r to C via an anonymous channel;
- C opens the ℓ-th ballot using the random r and publishes the vote v.

Note that we need to separate the voting phase into a commitment phase and an opening phase to avoid releasing partial results of the election and to ensure privacy. This is ensured by requiring synchronisation between the different agents involved in the election.

5.2 The model in applied pi

Cryptographic primitives as an equational theory. We model cryptography in a Dolev-Yao style as being perfect. The equations are given below.

$$
\begin{aligned}
\mathsf{open}(\mathsf{commit}(\mathsf{m}, \mathsf{r}), \mathsf{r}) &= \mathsf{m} \\
\mathsf{checksign}(\mathsf{sign}(\mathsf{m}, \mathsf{sk}), \mathsf{pk}(\mathsf{sk})) &= \mathsf{m} \\
\mathsf{unblind}(\mathsf{blind}(\mathsf{m}, \mathsf{r}), \mathsf{r}) &= \mathsf{m} \\
\mathsf{unblind}(\mathsf{sign}(\mathsf{blind}(\mathsf{m}, \mathsf{r}), \mathsf{sk}), \mathsf{r}) &= \mathsf{sign}(\mathsf{m}, \mathsf{sk})
\end{aligned}
$$

In this model we can note that bit commitment (modelled by the functions commit and open) is identical to classical symmetric-key encryption. For simplicity, we identify host names and public keys. Our model of cryptographic primitives is an abstraction; for example, bit commitment gives us perfect binding and hiding. Digital signatures are modeled as being signatures with message recovery, i.e. the signature itself contains the signed message which can be extracted using the $\mathsf{checksign}$ function. To model blind signatures we add a pair of functions blind and $\mathsf{unblind}$. These functions are again similar to perfect symmetric key encryption and bit commitment. However, we add a second equation which permits us to extract a signature out of a blind signature, when the blinding factor is known. Note that the equation modelling commitment cannot be applied on the term $\mathsf{open}(\mathsf{commit}(\mathsf{m}, \mathsf{r}_1), \mathsf{r}_2)$ when $\mathsf{r}_1 \neq \mathsf{r}_2$.

Process synchronisation. As mentioned, the protocol is divided into three phases, and it is important that every voter has completed the first phase before going onto the second one (and then has completed the second one before continuing to the

```
(* private channels *)
ν privCh.ν pkaCh1.ν pkaCh2.ν skaCh.ν skvaCh.ν skvbCh.
(* administrators *)
(processK | processA | processA | processC | processC |
(* voters *)
(let skvCh = skvaCh in let v = a in processV) |
(let skvCh = skvbCh in let v = b in processV) )
```

Process 1. Main process

```
processK=
    (* private keys *)
    ν ska. ν skva. ν skvb.
    (* corresponding public keys *)
    let (pka, pkva, pkvb)=(pk(ska), pk(skva), pk(skvb)) in
    (* public keys disclosure *)
    out(ch,pka). out(ch,pkva). out(ch,pkvb).
    (* register legitimate voters *)
    (out(privCh,pkva)| out(privCh,pkvb) |
    (* keys disclosure on private channels *)
    out(pkaCh1,pka) | out(pkaCh1,pka) | out(pkaCh2,pka) |
    out(pkaCh2,pka) | out(skaCh,ska)  | out(skaCh,ska) |
    out(skvaCh,skva) | out(skvbCh,skvb))
```

Process 2. Administrator for keying material

third). We enforce this in our model by the keyword synch. When a process encounters synch n, it waits until all the other process that could encounter synch n arrive at that point too. Then all the processes are allowed to continue.

If there are k processes that can encounter synch n, we can implement the synchronisation as follows. The command synch n is replaced by $\mathsf{out}(\mathsf{n}, 0); \mathsf{in}(\mathsf{n}, =1)$ where n is a globally declared private channel. Moreover we assume an additional process $(\mathsf{in}(\mathsf{n}, =0); \ldots; \mathsf{in}(\mathsf{n}, =0); \mathsf{out}(\mathsf{n}, 1); \ldots; \mathsf{out}(\mathsf{n}, 1)$ that has k ins and k outs. This simple encoding is fine for our purpose since the value of k can be inferred by inspecting the code; it would not work if new processes were created, e.g. with "!".

Main (Process 1). The main process sets up private channels and specifies how the processes are combined in parallel. Most of the private channels are for key distribution. We only model the protocol for two voters and launch two copies of the administrator and collector process, one for each voter.

Keying material (Process 2). Our model includes a dedicated process for generating and distributing keying material modelling a PKI. Additionally, this process

```
processV =                          (* parameters: skvCh, v *)
    (* her private key *)
    in(skvCh, skv).
    (* public keys of the administrator *)
    in(pkaCh1, pubka).
    ν blinder. ν r.
    let committedvote = commit(v,r) in
    let blindedcommittedvote=blind(committedvote,blinder) in
    out(ch1,(pk(skv),sign(blindedcommittedvote,skv))).
    in(ch2,m2).
    let result = checksign(m2,pubka) in
    if result = blindedcommittedvote then
    let signedcommittedvote=unblind(m2,blinder) in
    synch 1.
    out(ch3,(committedvote,signedcommittedvote)).
    synch 2.
    in(ch4,(l,=committedvote,=signedcommittedvote)).
    out(ch5,(l,r))
```

Process 3. Voter process

```
processA =
    (* administrator's private key *)
    in(skaCh, skadm).
    (* register legimitate voters *)
    in(privCh, pubkv).
    in(ch1,m1).
    let (pubkeyv, sig) = m1 in
    if pubkeyv = pubkv then
    out(ch2,sign(checksign(sig,pubkeyv),skadm))
```

Process 4. Administrator process

registers legitimate voters and also distributes the public keys of the election authorities to legitimate voters: this is modelled using restricted channels so that the attacker cannot provide false public keys.

Voter (Process 3). First, each voter obtains her secret key from the PKI as well as the public keys of the administrator. The remainder of the specification follows directly the informal description given in Section 5.1.

Administrator (Process 4). The administrator first receives through a private channel his own public key as well as the public key of a legitimate voter. Legitimate voters have been registered on this private channel in Process 2 described above. The received public key has to match the voter who is trying to get a signed

```
processC =
    (* administrator's public key *)
    in(pkaCh2, pkadmin).
    synch 1. in(ch3,(m3,m4)).
    if checksign(m4,pkadmin) = m3 then
    synch 2.
    ν l.
    out(ch4,(l,m3,m4)).
    in (ch5,(=l,rand)).
    let voteV=open(m4,rand) in
    out(ch,voteV)
```

Process 5. Collector process

ballot from the administrator. If the public key indeed matches, then the administrator signs the received message which he supposes to be a blinded ballot.

Collector (Process 5). When the collector receives a committed vote, he associates a fresh label 'l' with this vote. Publishing the list of votes and labels is modelled by sending those values on a public channel. Then the voter can send back the random number which served as a key in the commitment scheme together with the label. The collector receives the key matching the label and opens the vote which he then publishes.

5.3 Analysis

Vote-privacy. According to our definition, to show that the protocol respects privacy, we need to show that

$$S[V_A\{^a/_v\} \mid V_B\{^b/_v\}] \approx_\ell S[V_A\{^b/_v\} \mid V_B\{^a/_v\}]. \tag{4}$$

where $V_A = \mathsf{processV}\{^{skvaCh}/_{skvCh}\}$, $V_B = \mathsf{processV}\{^{skvbCh}/_{skvCh}\}$ and S is defined as the parallel composition of the voter processes, but with a hole instead of the two voter processes. We do not require that any of the authorities are honest, so they are not modelled in S, but rather left as part of the attacker context. To establish this equivalence, we show that

$$\begin{gathered}\nu\mathsf{pkaCh1}.(V_A\{^a/_v\} \mid V_B\{^b/_v\} \mid \mathsf{processK}) \\ \approx_\ell \\ \nu\mathsf{pkaCh1}.(V_A\{^b/_v\} \mid V_B\{^a/_v\} \mid \mathsf{processK})\end{gathered} \tag{5}$$

Note that this implies privacy (equivalence 4) only in the case of precisely two voters (i.e., S doesn't contain any additional voters). To deduce equivalence 4 for an arbitrary context S, one would like to use the fact that labelled bisimilarity is closed under application of evaluation contexts. Unfortunately, the context $\nu\mathsf{pkaCh1}.\_$ prevents us from easily making this inference (recall that pkaCh1 is the channel on which the voters receive the public key of the administrator). Our proof is formally valid only for two voters, although a similar proof can easily be made for other numbers.

Note that to ensure privacy we do not need to require any of the keys to be secret. However, we need to ensure that both voters use the same public key for the administrator. Therefore, we send this public key on a private channel, although the secret key itself is a free name. We α-rename the bounded variables and names in the two voter processes in a straightforward way. Although ProVerif is not able to prove this observational equivalence directly, we were able to check all of the static equivalences on the frames below using ProVerif (see Lemmas 19 and 20).

We denote the left-hand process as P and the right-hand process as Q. We have that both processK start with the output of all the keys. None of these transitions depend on the value of the vote, and so they commute in the same way for P and Q. For the sake of readability, we do not detail this part. The only important point is that the output of the administrator's public key is sent on a private channel yielding an internal reduction. We have that

$$
\begin{array}{lll}
P & \xrightarrow{in(skvaCh,skva)} P_1 \xrightarrow{in(skvbCh,skvb)} P_2 \to^* & \\
& \xrightarrow{\nu x_1.out(ch,x_1)} & \nu b_A.\nu r_A.\nu b_B.\nu r_B.(P_3 \mid \{^{(pk(skva),sign(blind(commit(a,r_A),b_A),skva))}/_{x_1}\} \\
& \xrightarrow{\nu x_2.out(ch,x_2)} & \nu b_A.\nu r_A.\nu b_B.\nu r_B.(P_4 \mid \{^{(pk(skva),sign(blind(commit(a,r_A),b_A),skva)}/_{x_1}\} \\
& & \qquad \mid \{^{(pk(skvb),sign(blind(commit(b,r_B),b_B),skvb)}/_{x_2}\})
\end{array}
$$

Similarly,

$$
\begin{array}{lll}
Q & \xrightarrow{in(skvaCh,skva)} Q_1 \xrightarrow{in(skvbCh,skvb)} Q_2 \to^* & \\
& \xrightarrow{\nu x_1.out(ch,x_1)} & \nu b_A.\nu r_A.\nu b_B.\nu r_B.(Q_3 \mid \{^{(pk(skva),sign(blind(commit(b,r_A),b_A),skva))}/_{x_1}\} \\
& \xrightarrow{\nu x_2.out(ch,x_2)} & \nu b_A.\nu r_A.\nu b_B.\nu r_B.(Q_4 \mid \{^{(pk(skva),sign(blind(commit(b,r_A),b_A),skva)}/_{x_1}\} \\
& & \qquad \mid \{^{(pk(skvb),sign(blind(commit(a,r_B),b_B),skvb)}/_{x_2}\})
\end{array}
$$

We could have considered any permutation of these transitions which respects the partial order dictated by the processes. Note that for the above inputs we may consider any public term, i.e. term that does not use bounded names of the processes. For the next input of both voters, we need to consider two cases: either the input of both voters corresponds to the expected messages from the administrator or at least one input does not correspond to the correct administrator's signature. In the second case, one of the voters will block, as testing correctness of the message fails and hence they cannot synchronise. In the first case, both voters synchronise at phase 1. Until that point any move of voter $V_A\{^a/_v\}$ on the left-hand side has been imitated by voter $V_A\{^b/_v\}$ on the right-hand side and equally for the second voter. However, from now on, any move of voter $V_A\{^a/_v\}$ on the left-hand side will be matched with the corresponding move of $V_B\{^a/_v\}$ on the right-hand side and similarly for the second voter. The voters will now output the committed votes signed by the administrator. The corresponding frames are described below and are statically equivalent.

$$\begin{aligned}\phi_{P'} \equiv \nu b_A.\nu r_A.\nu b_B.\nu r_B.\ & \{^{(pk(skva),sign(blind(commit(a,r_A),b_A),skva))}/_{x_1}\} \mid \\ & \{^{(pk(skvb),sign(blind(commit(b,r_B),b_B),skvb))}/_{x_2}\} \mid \\ & \{^{(commit(a,r_A),sign(commit(a,r_A),ska))}/_{x_3}\} \mid \\ & \{^{(commit(b,r_B),sign(commit(b,r_B),ska))}/_{x_4}\}\end{aligned}$$

$$\begin{aligned}\phi_{Q'} \equiv \nu b_A.\nu r_A.\nu b_B.\nu r_B.\ & \{^{(pk(skva),sign(blind(commit(b,r_A),b_A),skva))}/_{x_1}\} \mid \\ & \{^{(pk(skvb),sign(blind(commit(a,r_B),b_B),skvb))}/_{x_2}\} \mid \\ & \{^{(commit(a,r_B),sign(commit(a,r_B),ska))}/_{x_3}\} \mid \\ & \{^{(commit(b,r_A),sign(commit(b,r_A),ska))}/_{x_4}\}\end{aligned}$$

The following result can be establish using ProVerif.

Lemma 19 *The frames $\phi_{P'}$ and $\phi_{Q'}$ are statically equivalent.*

For the following input, we again consider two cases: either the input of both voters corresponds to the expected messages or at least one input does not succeed the tests. In the second case, one of the voters will block, as testing correctness of the message fails and hence they cannot synchronise. In the first case, we obtain at the end the two frames below which are again statically equivalent.

$$\begin{aligned}\phi_{P''} \equiv \nu b_A.\nu r_A.\nu b_B.\nu r_B.\ & \{^{(pk(skva),sign(blind(commit(a,r_A),b_A),skva))}/_{x_1}\} \\ & \{^{(pk(skvb),sign(blind(commit(b,r_B),b_B),skvb))}/_{x_2}\} \mid \\ & \{^{(commit(a,r_A),sign(commit(a,r_A),ska))}/_{x_3}\} \mid \\ & \{^{(commit(b,r_B),sign(commit(b,r_B),ska))}/_{x_4}\} \mid \\ & \{^{(l_A,r_A)}/_{x_5}\} \mid \{^{(l_B,r_B)}/_{x_6}\}\end{aligned}$$

$$\begin{aligned}\phi_{Q''} \equiv \nu b_A.\nu r_A.\nu b_B.\nu r_B.\ & \left\{{}^{(pk(skva),sign(blind(commit(b,r_A),b_A),skva))}/_{x_1}\right\} \mid \\ & \left\{{}^{(pk(skvb),sign(blind(commit(a,r_B),b_B),skvb))}/_{x_2}\right\} \mid \\ & \left\{{}^{(commit(a,r_B),sign(commit(a,r_B),ska))}/_{x_3}\right\} \mid \\ & \left\{{}^{(commit(b,r_A),sign(commit(b,r_A),ska))}/_{x_4}\right\} \mid \\ & \left\{{}^{(l_A,r_B)}/_{x_5}\right\} \mid \left\{{}^{(l_B,r_A)}/_{x_6}\right\}\end{aligned}$$

Again, ProVerif is able to establish the following result.

Lemma 20 *The frames $\phi_{P''}$ and $\phi_{Q''}$ are statically equivalent.*

Note that it is sufficient to prove static equivalences for all reachable final states. Thus, Lemma 19 is actually a consequence of Lemma 20.

Note that the use of phases is crucial for privacy to be respected. When we omit the synchronisation after the registration phase with the administrator, privacy is violated. Indeed, consider the following scenario. Voter V_A contacts the administrator. As no synchronisation is considered, voter V_A can send his committed vote to the collector before voter V_B contacts the administrator. As voter V_B could not have submitted the committed vote, the attacker can link this commitment to the first voter's identity. This problem was found during a first attempt to prove the protocol where the phase instructions were omitted. The original paper divides the protocol into three phases but does not explain the crucial importance of the synchronisation after the first phase. Our analysis emphasises this need and we believe that it increases the understanding of some subtle details of the privacy property in this protocol. We may also note that we do not make any assumptions about the correctness of the administrator or the collector, who may be corrupt, However, we need to assume that both voters use the same value for the administrator's public key. Otherwise, privacy does not hold.

Receipt-freeness. This scheme is not receipt-free and was in fact not designed with receipt-freeness in mind. Indeed, if the voter gives away the random numbers for blinding and commitment, i.e. b_A and r_A, the coercer can verify that the committed vote corresponds to the coercer's wish and by unblinding the first message, the coercer can trace which vote corresponds to this particular voter. Moreover, the voter cannot lie about these values as this will immediately be detected by the coercer.

In our framework, this corresponds to the fact that there exists no V' such that:

- $V'^{\backslash out(chc,\cdot)} \approx_\ell V_A\{{}^a/_v\}$,
- $S[V_A\{{}^c/_v\}^{chc} \mid V_B\{{}^a/_v\}] \approx_\ell S[V' \mid V_B\{{}^c/_v\}]$.

We show that there is no V' by proving that the requirements on V' are not satisfiable. We have that $V_A\{^c/_v\}^{chc}$ outputs the values r_A and b_A on the channel chc. This will generate entries in the frame. Hence, V' needs to generate similar entries in the frame. The coercer can now verify that the values r_A and b_A are used to encode the vote c in the message sent to the administrator. Thus V' is not able to commit to a value different from c, in order to satisfy the second equivalence. But then V' will not satisfy the first equivalence, since he will be unable to change his vote afterwards as the commitment to c has been signed by the administrator. Thus, the requirements on V' are not satisfiable.

The failure of receipt-freeness is not due to the possible dishonesty of the administrator or collector; even if we include them as honest parties, the protocol still doesn't guarantee receipt-freeness. It follows that coercion-resistance doesn't hold either.

6 Protocol due to Okamoto

In this section we study a protocol due to Okamoto [40] which was designed to be incoercible. However, Okamoto himself shows a flaw [41]. According to him, one of the reasons why the voting scheme he proposed had such a flaw is that no formal definition and proof of receipt-freeness and coercion-resistance have been given when the concept of receipt-freeness has been introduced by Benaloh and Tuinstra [8].

6.1 Description

The authorities managing the election are an administrator for registration, a collector for collecting the tokens and a timeliness member (denoted by T) for publishing the final tally. The main difference with the protocol due to Fujioka *et al.* is the use of a trap-door bit commitment scheme [23] in order to retrieve receipt-freeness. Such a commitment scheme allows the agent who has performed the commitment to open it in many ways. Hence, trap-door bit commitment does not bind the voter to the vote v. Now, to be sure that the voter does not change her mind at the end (during the opening stage) she has to say how she wants to open her commitment during the voting stage. This is done by sending the required information to T through an untappable anonymous channel, i.e. a physical apparatus by which only voter V can send a message to a party, and the message is perfectly secret to all other parties.

The first phase is similar to the one of the protocol due to Fujioka *et al.*. The only change is that ξ is a trap-door bit commitment scheme.

The second phase of the protocol is the actual voting phase. Now, the voter has to say how she wants to open her commitment to the timeliness member T.

- V sends y, A's signature on the trap-door commitment to V's vote, to the collector C using an anonymous channel;
- C checks correctness of the signature y and, if the test succeeds, enters (x, y) into a list.
- V sends (v, r, x) to the timeliness member T through an untappable anonymous channel.

The last phase of the voting protocol starts, once the collector decides that he received all votes, e.g. after a fixed deadline.

- C publishes the list (x_i, y_i) of trap-door commitments he obtained;
- V verifies that her commitment is in the list;
- T publishes the list of the vote v_i in random order and also proves that he knows the permutation π and the r_i's such that $x_{\pi(i)} = \xi(v_i, r_i)$ without revealing π or the r_i's.

We have chosen to not entirely model this last phase. In particular, we do not model the zero-knowledge proof performed by the timeliness member T, as it is not relevant for illustrating our definitions of privacy, receipt-freeness and coercion-resistance. This proof of zero-knowledge is very useful to ensure that T outputs the correct vote chosen by the voter. This is important in order to ensure correctness, even in the case that T is dishonest. However, the proof of knowledge is unimportant for anonymity properties. In particular, if T is the coercer himself, then he can enforce the voter to vote as he wants as in the protocol due to Fujioka *et al.* Indeed, the timeliness member T can force the voter to give him the trap-door she has used to forge her commitment and then he can not only check if the voter has vote as he wanted, but he can also open her vote as he wants.

6.2 The model in applied pi

Cryptographic primitives as an equational theory. The equations modelling public keys and blind signatures are the same as in Section 5.2. To model trap-door bit commitment, we consider the two following equations:

$$\begin{aligned}\mathsf{open}(\mathsf{tdcommit}(\mathsf{m}, \mathsf{r}, \mathsf{td}), \mathsf{r}) &= \mathsf{m}\\ \mathsf{tdcommit}(\mathsf{m}_1, \mathsf{r}, \mathsf{td}) &= \mathsf{tdcommit}(\mathsf{m}_2, \mathsf{f}(\mathsf{m}_1, \mathsf{r}, \mathsf{td}, \mathsf{m}_2), \mathsf{td})\end{aligned}$$

Firstly, the term $\mathsf{tdcommit}(\mathsf{m}, \mathsf{r}, \mathsf{td})$ models the commitment of the message m under the key r by using the trap-door td. The second equation is used to model

```
    (* private channels *)
    ν privCh. ν pkaCh1. ν pkaCh2.
    ν skaCh. ν skvaCh. ν skvbCh. ν chT.
    (* administrators *)
    (processK | processA | processA | processC | processC |
     processT | processT |
    (* voters *)
    (let skvCh=skvaCh in let v=a in processV) |
    (let skvCh=skvbCh in let v=b in processV) )
```

Process 6. Main process

```
processV =                    (* parameters: skvCh, v *)
    (* her private key *)
    in(skvCh, skv).
    (* public keys of the administrator *)
    in(pkaCh1, pubka).
    ν blinder. ν r. ν td.
    let committedvote = tdcommit(v,r,td) in
    let blindedcommittedvote=blind(committedvote, blinder) in
    out(ch1,(pk(skv), sign(blindedcommittedvote, skv))).
    in(ch2,m2).
    let result = checksign(m2,pubka) in
    if result = blindedcommittedvote then
    let signedcommittedvote=unblind(m2,blinder) in
    synch 1.
    out(ch3,(committedvote, signedcommittedvote)).
    out(chT,(v,r,committedvote))
```

Process 7. Voter process

the fact that a commitment $\mathsf{tdcommit}(\mathsf{m}_1, \mathsf{r}, \mathsf{td})$ can be viewed as a commitment of any value m_2. However, to open this commitment as m_2 one has to know the key $\mathsf{f}(\mathsf{m}_1, \mathsf{r}, \mathsf{td}, \mathsf{m}_2)$. Note that this is possible only if one knows the key r used to forge the commitment $\mathsf{tdcommit}(\mathsf{m}_1, \mathsf{r}, \mathsf{td})$ and the trap-door td.

Main (Process 6). The main process sets up private channels and specifies how the processes are combined in parallel. Most of the private channels are for key distribution. The channel chT is the untappable anonymous channel on which voters send to T how they want to open their commitment.

We have also a dedicated process for generating and distributing keying material modelling a PKI. This process is the same as the one we have given for the protocol due to Fujioka *et al.* (see Section 5).

```
processC =
    (* administrator's public key *)
    in(pkaCh2, pkadmin).
    synch 1.
    in(ch3,(m3,m4)).
    if checksign(m4,pkadmin) = m3 then
    synch 2.
    out(chBB,(m3,m4))
```

Process 8. Collector process

```
processT =
    synch 1.
    (* reception du commitment *)
    in(chT,(vt, rt, xt)).
    synch 2.
    if open(xt, rt) = vt then
    out(board, vt)
```

Process 9. Timeliness process

Voter (Process 7). This process is very similar to the one given in the previous section. We use the primitive tdcommit instead of commit and at the end, the voter sends, through the channel chT, how she wants to open her commitment.

Administrator. The administrator process is exactly the same as the one given in Section 5 to model the protocol due to Fujioka *et al.*

Collector (Process 8). When C receives a commitment, he checks the correctness of the signature and if he succeeds, he enters this pair into a list. This list is published in a second phase by sending the values contained in the list on the public channel chBB.

Timeliness Member (Process 9). The timeliness member receives, through chT, messages of the form $(\mathsf{vt}, \mathsf{rt}, \mathsf{xt})$ where vt is the value of the vote, xt the trap-door bit commitment and rt the key he has to use to open the commitment. In a second phase, he checks that he can obtain vt by opening the commitment xt with rt. Then, he publishes the vote vt on the board. This is modelled by sending vt on a public channel.

Unfortunately, the equational theory which is required to model this protocol is beyond the scope of ProVerif and we cannot rely on automated verification, even for the static equivalence parts.

Vote-privacy. Privacy can be established as in the protocol due to Fujioka *et al.* Note that the equivalence proved there does not hold here. We have to hide the outputs on the channel chT. Hence, we establish the following equivalence

$$\nu\mathsf{pkaCh1}.\nu\mathsf{chT}.(V_A\{^a/_v\} \mid V_B\{^b/_v\} \mid \mathsf{processK} \mid \mathsf{processT} \mid \mathsf{processT})$$
$$\approx_\ell$$
$$\nu\mathsf{pkaCh1}.\nu\mathsf{chT}.(V_A\{^b/_v\} \mid V_B\{^a/_v\} \mid \mathsf{processK} \mid \mathsf{processT} \mid \mathsf{processT})$$

Below we show that the protocol respects receipt-freeness and hence privacy also holds.

Receipt-freeness. To show receipt-freeness one needs to construct a process V' which successfully fakes all secrets to a coercer. The idea is for V' to vote a, but when outputting secrets to the coercer, V' lies and gives him fake secrets to pretend to cast the vote c. The crucial part is that, using trap-door commitment and thanks to the fact that the key used to open the commitment is sent through an untappable anonymous channel, the value given by the voter to the timeliness member T can be different from the one she provides to the coercer. Hence, the voter who forged the commitment, provides to the coercer the one allowing the coercer to retrieve the vote c, whereas she sends to T the one allowing her to cast the vote a.

We describe such a process V' in Process 10. To prove receipt-freeness, we need to show that

- $V'^{\setminus out(chc,\cdot)} \approx_\ell V_A\{^a/_v\}$, and
- $S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}] \approx_\ell S[V' \mid V_B\{^c/_v\}]$.

The context S we consider here is the same we have used to establish privacy, i.e. $\nu\mathsf{pkaCh1}.\nu\mathsf{chT}.(\_ \mid \mathsf{processK} \mid \mathsf{processT} \mid \mathsf{processT})$; thus, as for Fujioka *et al.*, the proof is valid for two voters. The first equivalence may be seen informally by considering V' without the instructions "$\mathsf{out}(\mathsf{chc}, \ldots)$", and comparing it visually with $V_A\{^a/_v\}$. The two processes are the same.

To see the second labelled bisimulation, one can informally consider all the executions of each side. We denote the left-hand process as P and the right-hand as Q.

```
processV =
    (* her private key *)
    in(skvCh, skv). out(chc, skv).
    (* public keys of the administrator *)
    in(pkaCh1, pubka).  out(chc, pubka).
    ν blinder. ν r.  ν td.
    out(chc, blinder). out(chc, f(a, r, td, c)). out(chc, td).
    let committedvote = tdcommit(a, r, td) in
    let blindedcommittedvote=blind(committedvote, blinder) in
    out(ch1, (pk(skv), sign(blindedcommittedvote, skv))).
    out(chc, (pk(skv), sign(blindedcommittedvote, skv))).
    in(ch2, m2).
    let result = checksign(m2, pubka) in
    if result = blindedcommittedvote then
    let signedcommittedvote=unblind(m2, blinder) in
    synch 1.
    out(ch3, (committedvote, signedcommittedvote)).
    out(chc, (committedvote, signedcommittedvote)).
    out(chT, (a, r, committedvote)).
    out(chc, (c, f(a, r, td, c), committedvote))
```

Process 10. V'- Receipt-freeness

Both processK start with the output of all the keys. For sake of readability, we ignore these outputs which are not really important for what we wish to show. We denote by $\tilde{n}$ the sequence of names $b_A, r_A, td_A, b_B, r_B, td_B$. After distribution of keying material which can be done in the same way on both sides, we observe that the instructions of $V_A\{^c/_v\}^{chc}$ can be matched with those of V'. Similarly, execution steps performed by $V_B\{^a/_v\}$ on the left are matched with $V_B\{^c/_v\}$ on the right. We need, of course, to consider all the possible executions of the two processes. However, to argue that the processes are bisimilar, we consider below a particular execution and we describe the interesting part of the two frames we obtained after execution of the first phase by the two processes.

$$
\begin{aligned}
P &\xrightarrow{in(skvaCh,skva)} \xrightarrow{\nu x_1.out(chc,x_1)} P_1 \mid \{^{skva}/_{x_1}\} \xrightarrow{in(skvbCh,skvb)}\rightarrow^* \; P_2 \mid \{^{skva}/_{x_1}\} \\
&\xrightarrow{\nu x_2.out(chc,x_2)} \xrightarrow{\nu x_3.out(chc,x_3)} \xrightarrow{\nu x_4.out(chc,x_4)} \\
&\xrightarrow{\nu x_5.out(chc,x_5)} \nu\tilde{n}.\ (P_3 \mid \{^{skva}/_{x_1}\} \mid \{^{pk(ska)}/_{x_2}\} \mid \{^{b_A}/_{x_3}\} \mid \{^{r_A}/_{x_4}\} \mid \{^{td_A}/_{x_5}\}) \\
&\xrightarrow{\nu x_6.out(ch,x_6)} \nu\tilde{n}.\ (P_4 \mid \{^{skva}/_{x_1}\} \mid \{^{pk(ska)}/_{x_2}\} \mid \{^{b_A}/_{x_3}\} \mid \{^{r_A}/_{x_4}\} \mid \{^{td_A}/_{x_5}\} \\
&\qquad\qquad \mid \{^{(pk(skva),sign(blind(tdcommit(c,r_A,td_A),b_A),skva))}/_{x_6}\}) \\
&\xrightarrow{\nu x_7.out(chc,x_7)} \nu\tilde{n}\ .(P_5 \mid \{^{skva}/_{x_1}\} \mid \{^{pk(ska)}/_{x_2}\} \mid \{^{b_A}/_{x_3}\} \mid \{^{r_A}/_{x_4}\} \mid \{^{td_A}/_{x_5}\} \\
&\qquad\qquad \mid \{^{(pk(skva),sign(blind(tdcommit(c,r_A,td_A),b_A),skva))}/_{x_6}\} \mid \{^{x_6}/_{x_7}\}) \\
&\xrightarrow{\nu x_8.out(ch,x_8)} \nu\tilde{n}\ .(P_6 \mid \{^{skva}/_{x_1}\} \mid \{^{pk(ska)}/_{x_2}\} \mid \{^{b_A}/_{x_3}\} \mid \{^{r_A}/_{x_4}\} \mid \{^{td_A}/_{x_5}\} \\
&\qquad\qquad \mid \{^{(pk(skva),sign(blind(tdcommit(c,r_A,td_A),b_A),skva))}/_{x_6}\} \mid \{^{x_6}/_{x_7}\}) \\
&\qquad\qquad \mid \{^{(pk(skvb),sign(blind(tdcommit(a,r_B,td_B),b_B),skvb))}/_{x_8}\}).
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
Q &\xrightarrow{in(skvaCh,skva)} \xrightarrow{\nu x_1.out(chc,x_1)} Q_1 \mid \{^{skva}/_{x_1}\} \xrightarrow{in(skvbCh,skvb)}\rightarrow^* \; Q_2 \mid \{^{skva}/_{x_1}\} \\
&\xrightarrow{\nu x_2.out(chc,x_2)} \xrightarrow{\nu x_3.out(chc,x_3)} \xrightarrow{\nu x_4.out(chc,x_4)} \\
&\xrightarrow{\nu x_5.out(chc,x_5)} \nu\tilde{n}.\ (Q_3 \mid \{^{skva}/_{x_1}\} \mid \{^{pk(ska)}/_{x_2}\} \mid \{^{b_A}/_{x_3}\} \mid \{^{f(a,r_A,td_A,c)}/_{x_4}\} \mid \{^{td_A}/_{x_5}\}) \\
&\xrightarrow{\nu x_6.out(ch,x_6)} \nu\tilde{n}.\ (Q_4 \mid \{^{skva}/_{x_1}\} \mid \{^{pk(ska)}/_{x_2}\} \mid \{^{b_A}/_{x_3}\} \mid \{^{f(a,r_A,td_A,c)}/_{x_4}\} \mid \{^{td_A}/_{x_5}\} \\
&\qquad\qquad \mid \{^{(pk(skva),sign(blind(tdcommit(a,r_A,td_A),b_A),skva))}/_{x_6}\}) \\
&\xrightarrow{\nu x_7.out(chc,x_7)} \nu\tilde{n}\ .(Q_5 \mid \{^{skva}/_{x_1}\} \mid \{^{pk(ska)}/_{x_2}\} \mid \{^{b_A}/_{x_3}\} \mid \{^{f(a,r_A,td_A,c)}/_{x_4}\} \mid \{^{td_A}/_{x_5}\} \\
&\qquad\qquad \mid \{^{(pk(skva),sign(blind(tdcommit(a,r_A,td_A),b_A),skva))}/_{x_6}\} \mid \{^{x_6}/_{x_7}\}) \\
&\xrightarrow{\nu x_8.out(ch,x_8)} \nu\tilde{n}\ .(Q_6 \mid \{^{skva}/_{x_1}\} \mid \{^{pk(ska)}/_{x_2}\} \mid \{^{b_A}/_{x_3}\} \mid \{^{f(a,r_A,td_A,c)}/_{x_4}\} \mid \{^{td_A}/_{x_5}\} \\
&\qquad\qquad \mid \{^{(pk(skva),sign(blind(tdcommit(a,r_A,td_A),b_A),skva))}/_{x_6}\} \mid \{^{x_6}/_{x_7}\} \\
&\qquad\qquad \mid \{^{(pk(skvb),sign(blind(tdcommit(c,r_B,td_B),b_B),skvb))}/_{x_8}\}).
\end{aligned}
$$

We argue informally that the frames obtained at the end of this first phase are statically equivalent. In particular, note that the test

$$\mathsf{open}(\mathsf{unblind}(\mathsf{checksign}(\mathsf{proj}_2(\mathsf{x}_6), \mathsf{pk}(\mathsf{x}_1)), \mathsf{x}_3), \mathsf{x}_4) = \mathsf{c}$$

is true in both frames. Indeed, if we denote B' the process obtained on the left hand-side after this first phase, we have that

$$\begin{aligned}
&\mathsf{open}(\mathsf{unblind}(\mathsf{checksign}(\mathsf{proj_2}(\mathsf{x_6}), \mathsf{pk}(\mathsf{x_1})), \mathsf{x_3}), \mathsf{x_4})\sigma \\
&= \mathsf{open}(\mathsf{tdcommit}(\mathsf{a}, \mathsf{r_A}, \mathsf{td_A}), \mathsf{f}(\mathsf{a}, \mathsf{r_A}, \mathsf{td_A}, \mathsf{c})) \\
&= \mathsf{open}(\mathsf{tdcommit}(\mathsf{c}, \mathsf{f}(\mathsf{a}, \mathsf{r_A}, \mathsf{td_A}, \mathsf{c}), \mathsf{td_A}), \mathsf{f}(\mathsf{a}, \mathsf{r_A}, \mathsf{td_A}, \mathsf{c})) \\
&= \mathsf{c}
\end{aligned}$$

where $\phi(B') = \nu\tilde{n}.\sigma$.

For the "first input", of both voters, we need to consider two cases: either the input of both voters corresponds to the expected messages from the administrator or at least one input does not correspond to the correct administrator's signature. In the second case, one of the voters will block, as testing correctness of the message fails and hence the voters cannot synchronise. In the first case, we obtain at the end the two frames below.

$$\begin{aligned}
\phi_{P''} \equiv \nu\tilde{n}.\ &\{^{skva}/_{x_1}\} \mid \{^{pk(ska)}/_{x_2}\} \mid \{^{b_A}/_{x_3}\} \mid \{^{r_A}/_{x_4}\} \mid \{^{td_A}/_{x_5}\} \mid \\
&\{^{(pk(skva),sign(blind(tdcommit(c,r_A,td_A),b_A),skva))}/_{x_6}\} \mid \{^{x_6}/_{x_7}\} \mid \\
&\{^{(pk(skvb),sign(blind(tdcommit(a,r_B,td_B),b_B),skvb))}/_{x_8}\} \mid \\
&\{^{(tdcommit(c,r_A,td_A),sign(tdcommit(c,r_A,td_A),ska))}/_{x_9}\} \mid \{^{x_9}/_{x_{10}}\} \mid \\
&\{^{(tdcommit(a,r_B,td_B),sign(tdcommit(a,r_B,td_B),ska))}/_{x_{11}}\} \mid \\
&\{^{(c,r_A,tdcommit(c,r_A,td_A))}/_{x_{12}}\} \mid \{^{a}/_{x_{13}}\} \mid \{^{c}/_{x_{14}}\}
\end{aligned}$$

$$\begin{aligned}
\phi_{Q''} \equiv \nu\tilde{n}.\ &\{^{skva}/_{x_1}\} \mid \{^{pk(ska)}/_{x_2}\} \mid \{^{b_A}/_{x_3}\} \mid \{^{f(a,r_A,td_A,c)}/_{x_4}\} \mid \{^{td_A}/_{x_5}\} \mid \\
&\{^{(pk(skva),sign(blind(tdcommit(a,r_A,td_A),b_A),skva))}/_{x_6}\} \mid \{^{x_6}/_{x_7}\} \mid \\
&\{^{(pk(skvb),sign(blind(tdcommit(c,r_B,td_B),b_B),skvb))}/_{x_8}\} \mid \\
&\{^{(tdcommit(a,r_A,td_A),sign(tdcommit(a,r_A,td_A),ska))}/_{x_9}\} \mid \{^{x_9}/_{x_{10}}\} \mid \\
&\{^{(tdcommit(c,r_B,td_B),sign(tdcommit(c,r_B,td_B),ska))}/_{x_{11}}\} \mid \\
&\{^{(c,f(a,r_A,td_A,c),tdcommit(a,r_A,td_A))}/_{x_{12}}\} \mid \{^{a}/_{x_{13}}\} \mid \{^{c}/_{x_{14}}\}
\end{aligned}$$

We observe that the frames are statically equivalent. In particular, note that the test $\mathsf{tdcommit}(\mathsf{c}, \mathsf{x_4}, \mathsf{x_5}) = \mathsf{proj_1}(\mathsf{x_9})$ is true in both frames and the attacker cannot distinguish the terms $\mathsf{tdcommit}(\mathsf{a}, \mathsf{r_B}, \mathsf{td_B})$ and $\mathsf{tdcommit}(\mathsf{c}, \mathsf{r_B}, \mathsf{td_B})$ since he is not able to open this commitment. As the goal of this section is to illustrate our definitions and as tool support is not provided for this equational theory we do not give a formal proof of this static equivalence.

```
processC[] =
    ν c1. ν c2. ( _ |
    (* private key of V *)
    in(c1,x1). out(chc,x1).
    (* public keys of the administrator *)
    in(c1,x2).  out(chc,x2).
    ν blinder. ν r.  ν td.
    (* nonces of V - blinder, r, td *)
    in(c1,x3). out(chc,blinder).
    in(c1,x4). out(chc,r).
    in(c1,x5). out(chc,td).

    let committedvote = tdcommit(c,r,td) in
    let blindedcommittedvote=blind(committedvote,blinder) in
    out(c2,(pk(x1),sign(blindedcommittedvote,x1))).

    (* signature of the administrator *)
    in(c1,x6). out(chc,x6).
    let result = checksign(x6,x2) in
    if result = blindedcommittedvote then
    out(c2,true).
    let signedcommittedvote=unblind(x6,blinder) in
    synch 1.
    out(c2,(committedvote,signedcommittedvote)).
    out(c2,(c,r,committedvote))
```

Process 11. Context C - coercion-resistance

Coercion-resistance. This scheme is not coercion-resistant [41]. If the coercer provides the coerced voter with the commitment that he has to use but without revealing the trap-door, the voter cannot cast her own vote a since the voter cannot produce fake outputs as she did for receipt-freeness. In terms of our definition, we need to show that there is no V' such that for all coercer C satisfying $\tilde{n} \cap fn(C) = \emptyset$ and $S[C[V_A\{^?/_v\}^{c_1,c_2}] \mid V_B\{^a/_v\} \approx_\ell S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}]$, we have the two bullet points of the definition of coercion-resistance. Suppose V' was such a process. Let C be the context given as Process 11 (note that it is, in fact, independent of V'). In order to satisfy the second bullet point, V' has to use the commitment provided by the coercer, for otherwise this would yield an observable. But then it cannot give to the timeliness member the key to open the commitment to obtain the voter's desired vote, in order to satisfy the first bullet, since V' does not know the trap-door. Hence, for the given C, the requirements on V' are not satisfiable simultaneously.

7 Protocol due to Lee *et al.*

In this section we study a protocol based on the Lee *et al.* protocol [36]. One of the main advantages of this protocol is that it is *vote and go*: voters need to participate in the election only once, in contrast with [25] and [40] (see Sections 5 and 6), where all voters have to finish a first phase before any of them can participate in the second phase. We simplified the protocol in order to concentrate on the aspects that are important with respect to privacy, receipt-freeness and coercion-resistance. In particular we do not consider distributed authorities.

7.1 Description

The protocol relies on re-encryption and on a less usual cryptographic primitive: designated verifier proofs (DVP) of re-encryption. We start by explaining these primitives.

A re-encryption of a ciphertext (obtained using a randomised encryption scheme) changes the random coins, without changing or revealing the plaintext. In the ElGamal scheme for instance, if (x, y) is the ciphertext, this is simply done by computing (xg^r, yh^r), where r is a random number, and g and h are the subgroup generator and the public key respectively. Note that neither the creator of the original ciphertext nor the person re-encrypting knows the random coins used in the re-encrypted ciphertext, for they are a function of the coins chosen by both parties. In particular, a voter cannot reveal the coins to a potential coercer who could use this information to verify the value of the vote, by ciphering his expected vote with these coins.

A DVP of the re-encryption proves that the two ciphertexts contain indeed the same plaintext. However, a designated verifier proof only convinces one intended person, e.g., the voter, that the re-encrypted ciphertext contains the original plaintext. In particular this proof cannot be used to convince the coercer. Technically, this is achieved by giving the designated verifier the ability to simulate the transcripts of the proof. A more abstract description is the following. A DVP for a designated verifier A of a statement φ is a proof of the statement "$\varphi \vee$ I know A's private key". As A is the only one to know his own private key a proof that has not been generated by himself must be a proof of the statement φ while A himself can generate a proof of the second part of the disjunction.

Our simplified protocol can be described in three steps.

- Firstly, the voter encrypts his vote with the collector's public key (using the ElGamal scheme), signs the encrypted vote and sends it to an administrator on a private channel. The administrator checks whether the voter is a legitimate voter and has not voted yet. Then the administrator *re-encrypts* the given ciphertext, signs it and sends it back to the voter. The administrator also provides a DVP that the two ciphertexts contain indeed the same plaintext. In practice, this first stage of the protocol can be done using a voting booth where eligibility of the voter is tested at the entrance of the booth. The booth contains a tamper-proof device which performs re-encryptions, signatures and DVP proofs.

- Then, the voter sends (via an anonymous channel) the re-encrypted vote, which has been signed by the administrator to the public board.
- Finally, the collector checks the administrator's signature on each of the votes and, if valid, decrypts the votes and publishes the final results.

7.2 *The model in applied pi*

Cryptographic primitives as an equational theory. The functions and equations that handle public keys and digital signature are as usual (see Section 5 for instance). To model re-encryption we add a function $\mathsf{rencrypt}$, that permits us to obtain a different encryption of the same message with another random coin which is a function of the original one and the one used during the re-encryption. We also add a pair of functions dvp and $\mathsf{checkdvp}$: dvp permits us to build a *designated verifier proof* of the fact that a message is a re-encryption of another one and $\mathsf{checkdvp}$ allows the designated verifier to check that the proof is valid. Note that $\mathsf{checkdvp}$ also succeeds for a *fake dvp* created using the designated verifier's private key. We have the following equations:

$$\mathsf{decrypt}(\mathsf{penc}(\mathsf{m}, \mathsf{pk}(\mathsf{sk}), \mathsf{r}), \mathsf{sk}) = \mathsf{m}$$

$$\mathsf{rencrypt}(\mathsf{penc}(\mathsf{m}, \mathsf{pk}(\mathsf{sk}), \mathsf{r1}), \mathsf{r2}) = \mathsf{penc}(\mathsf{m}, \mathsf{pk}(\mathsf{sk}), \mathsf{f}(\mathsf{r1}, \mathsf{r2}))$$

$$\mathsf{checkdvp}(\mathsf{dvp}(\mathsf{x}, \mathsf{rencrypt}(\mathsf{x}, \mathsf{r}), \mathsf{r}, \mathsf{pk}(\mathsf{sk})), \mathsf{x}, \mathsf{rencrypt}(\mathsf{x}, \mathsf{r}), \mathsf{pk}(\mathsf{sk})) = \mathsf{ok}$$

$$\mathsf{checkdvp}(\mathsf{dvp}(\mathsf{x}, \mathsf{y}, \mathsf{z}, \mathsf{skv}), \mathsf{x}, \mathsf{y}, \mathsf{pk}(\mathsf{skv})) = \mathsf{ok}$$

Main (Process 12). The main process sets up private channels and specifies how the processes are combined in parallel. Most of the private channels are for key distribution. The private channel chA_1 (resp. chA_2) is a private channel between the voter and her administrator. This is motivated by the fact that the administrator corresponds to a tamper-proof hardware device in this protocol. We only model the protocol for two voters and launch two copies of the administrator and collector process, one for each voter.

Keying material (Process 13). Our model includes a dedicated process for generating and distributing keying material modelling a PKI. Additionally, this process

```
    (* private channels *)
    ν privCh. ν pkaCh1. ν pkaCh2. ν pkcCh. ν skaCh. ν skcCh.
    ν skvaCh. ν skvbCh. ν chA1. ν chA2.
    (* administrators *)
    (processK | processC | processC |
    (* voters *)
    (let chA = chA1 in processA |
    (let skvCh = skvaCh in let v = a in processV)) |
    (let chA = chA2 in processA |
    (let skvCh = skvbCh in let v = b in processV)))
```

Process 12. Main process

```
processK =
    (* private key *)
    ν ska. ν skc. ν skva. ν skvb.
    (* corresponding public keys *)
    let (pka, pkc) = (pk(ska), pk(skc)) in
    let (pkva, pkvb) = (pk(skva), pk(skvb)) in
    (* publik keys disclosure *)
    out(ch,pka). out(ch,pkc). out(ch,pkva). out(ch,pkvb).
    (* register legitimate voters *)
    (out(privCh,pkva)| out(privCh,pkvb) |
    (* keys disclosure on private channels *)
    out(pkaCh,pka) | out(pkaCh,pka) | out(pkaCh,pka) |
    out(pkaCh,pka) | out(skaCh,ska) | out(skaCh,ska) |
    out(pkcCh,pkc) | out(pkcCh,pkc) | out(skcCh,skc) |
    out(skcCh,skc) | out(skvaCh,skva) | out(skvbCh,skvb))
```

Process 13. Administrator for keying material

registers legitimate voters and also distributes the public keys of the election authorities to legitimate voters: this is modelled using restricted channels so that the attacker cannot provide false public keys.

Voter (Process 14). First, each voter obtains her secret key from the PKI as well as the public keys of the election authorities. Then, a fresh random number is generated to encrypt her vote with the public key of the collector. Next, she signs the result and sends it on a private channel to the administrator. If the voter has been correctly registered, she obtains from the administrator, a re-encryption of her vote signed by the administrator together with a designated verifier proof of the fact that this re-encryption has been done correctly. If this proof is correct, then the voter sends her re-encrypted vote signed by the administrator to the collector.

Note that we used the synchronisation command to model this process. This command is crucial for privacy to hold in presence of a corrupted collector. This ensures

```
processV =                    (* parameters: skvCh, v *)
    (* her private key *)
    in(skvCh, skv).
    (* public keys of the administrators *)
    in(pkaCh1, pubka). in(pkcCh, pubkc).
    synch 1. ν r.
    let e = penc(v, pubkc, r) in
    out(chA, (pk(skv), e, sign(e, skv))).
    in(chA, m2).
    let (re, sa, dvpV) = m2 in
    if checkdvp(dvpV, e, re, pk(skv)) = ok
    then if checksign(sa, pubka) = re
    then out(ch, (re, sa))
```

Process 14. Voter process

```
processA =
    (* administrator's private key *)
    in(skaCh, skadm).
    (* register a legimitate voter *)
    in(privCh, pubkv).
    synch 1.
    in(chA, m1).
    let (pubv, enc, sig)=m1 in
    if pubv=pubkv then
    if checksign(sig, pubv)= enc
    then ν r1.
    let reAd=rencrypt(enc, r1) in
    let signAd=sign(reAd, skadm) in
    let dvpAd=dvp(enc, reAd, r1, pubv) in
    out(chA, (reAd, signAd, dvpAd))
```

Process 15. Administrator process

that key distribution is finished before any of the two voter proceeds. Otherwise an attack on privacy can be mounted since the attacker can prevent one of the voters from obtaining her keys. One may also note that this protocol is *vote and go*: even if synchronisation is used the voters participate actively only during one of the synchronised phases.

Administrator (Process 15). The administrator first receives through a private channel his own private key as well as the public key of a legitimate voter. The received public key has to match the voter who is trying to get a re-encryption of her vote signed by the administrator. The administrator has also to prove to the voter that he has done the re-encryption properly. For this, he builds a designated verifier proof which will be only convincing for the voter.

```
processC =
    (* collector's private key *)
    in(skcCh,privc).
    (* administrator's public key *)
    in(pkaCh2,pkadmin).
    synch 1.
    in(ch,m3).
    let (ev,sev) = m3 in
    if checksign(sev,pkadmin) = ev
    then let voteV = decrypt(ev,privc) in
    synch 2.
    out(ch,voteV)
```

Process 16. Collector process

Collector (Process 16). First, the collector receives all the signed ballots. He checks the signature and decrypts the result with his private key to obtain the value of the vote in order to publish the results. Although it is not mentioned in the description of the protocol [36], it seems reasonable to think that the collector does not accept the same ballot twice. For sake of readability, we do not model this feature in Process 16; however, we will model it when we come to receipt-freeness, since it is crucial there. Finally, when all votes have been submitted to the collector (synchronisation is achieved using the synchronisation instruction), they are published.

7.3 Analysis

Let $V_A = V\{^{skvaCh}/_{skvCh}\}\{^{chA1}/_{chA}\}$ and $V_B = V\{^{skvbCh}/_{skvCh}\}\{^{chA2}/_{chA}\}$. Note that again we have to establish all the static equivalences manually: ProVerif is not able to deal with equational theories such as this one.

Vote privacy. We show that the protocol respects privacy. For this, we establish the following equivalence

$$S[V_A\{^a/_v\} \mid V_B\{^b/_v\}] \approx_\ell S[V_A\{^b/_v\} \mid V_B\{^a/_v\}]$$

$$\begin{aligned}\text{where } S = \nu \mathsf{pkaCh1}, \mathsf{pkcCh}, \mathsf{skaCh}, \mathsf{chA1}, \mathsf{chA2}.(\ \_\ &\mid \mathsf{processK}\\ &\mid \mathsf{processA}\{^{\mathsf{chA1}}/_{\mathsf{chA}}\}\\ &\mid \mathsf{processA}\{^{\mathsf{chA2}}/_{\mathsf{chA}}\})\end{aligned}$$

As for the other case studies, we prove privacy only for the case of two voters.

Privacy does not require any of the keys to be secret. However, we need to ensure that both voters use the same public key for the administrator and for the collector. Therefore, we send public keys on a private channel, although the corresponding private keys can be considered as free names. We assume that both administrators have the same private key and that both voters have the right to vote. If any of these conditions is not satisfied, privacy does not hold.

We denote the left-hand process as P and the right-hand process as Q. The processK starts with the output of all the keys. For the sake of readability, we ignore some of these outputs which are not important for our analysis and we write $\nu\tilde{r}$ instead of the sequence $\nu r_A.\nu r_B.\nu r_1.\nu r_2$.

$$
\begin{aligned}
P & \xrightarrow{in(skvaCh,skva)} \rightarrow^* \xrightarrow{in(skvbCh,skvb)} \rightarrow^* P_1 \\
& \xrightarrow{\nu x_1.out(ch,x_1)} \nu\tilde{r}.(P_2 \mid \{^{(penc(a,pkc,f(r_A,r_1)),sign(penc(a,pkc,f(r_A,r_1)),ska)}/_{x_1}\} \\
& \xrightarrow{\nu x_2.out(ch,x_2)} \nu\tilde{r}.(P_3 \mid \{^{(penc(a,pkc,f(r_A,r_1)),sign(penc(a,pkc,f(r_A,r_1)),ska)}/_{x_1}\} \\
& \qquad\qquad \mid \{^{(penc(b,pkc,f(r_B,r_2)),sign(penc(b,pkc,f(r_B,r_2)),ska)}/_{x_2}\})
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
Q & \xrightarrow{in(skvaCh,skva)} \rightarrow^* \xrightarrow{in(skvbCh,skvb)} \rightarrow^* Q_1 \\
& \xrightarrow{\nu x_1.out(ch,x_1)} \nu\tilde{r}.(Q_2 \mid \{^{(penc(a,pkc,f(r_B,r_2)),sign(penc(a,pkc,f(r_B,r_2)),ska)}/_{x_1}\} \\
& \xrightarrow{\nu x_2.out(ch,x_2)} \nu\tilde{r}.(Q_3 \mid \{^{(penc(a,pkc,f(r_B,r_2)),sign(penc(a,pkc,f(r_B,r_2)),ska)}/_{x_1}\} \\
& \qquad\qquad \mid \{^{(penc(b,pkc,f(r_A,r_1)),sign(penc(b,pkc,f(r_A,r_1)),ska)}/_{x_2}\})
\end{aligned}
$$

The resulting frames are statically equivalent. Note that, during key distribution, the process $V_A\{^a/_v\}$ is matched with $V_A\{^b/_v\}$, while afterwards $V_A\{^a/_v\}$ is matched with $V_B\{^a/_v\}$. Therefore, we require a phase after the keying distribution.

Receipt-freeness. To show receipt-freeness one needs to construct a process V' which can successfully fake all secrets to a coercer. The idea is that V' votes a, but when outputting secrets to the coercer V' prepares all outputs as if she was voting c. The crucial part is that, using her private key, she provides a fake DVP stating that the actual re-encryption of the encryption of vote a is a re-encryption of the encryption of vote c. Given our equational theory, the two resulting frames are statically equivalent because for both the real and the fake DVP, checkdvp returns ok.

To establish receipt-freeness, we have to assume that the collector is trusted. Indeed, it is important to be sure that its private key remains secret. Otherwise, an attack against receipt-freeness can be mounted: if the coercer knows the collector's

```
processV'=
    (* her private key *)
    in(skvaCh,skv). out(chc,skv).
    (* public keys of administrators *)
    in(pkaCh,pubka). out(chc,pubka).
    in(pkcCh,pubkc). out(chc,pubkc).
    synch 1.
    ν r. out(chc,r).
    let e = penc(a,pubkc,r) in
    out(chA1,(pk(skv),e,sign(e,skv))).

    (* message from the administrator *)
    in(chA1,m2).
    let (re,sa,dvpV) = m2 in
    if checkdvp(dvpV,e,re,pk(skv))=ok then
    ν r'.
    let fk=dvp(penc(c,pubkc,r), re,r',skv) in
    out(chc,(re,sa,fk)).
    if checksign(sa,pubka) = re then
    out(ch,(re,sa))
```

Process 17. Process V' - Receipt-Freeness

private key he can directly decrypt the re-encryption and check whether the vote is c rather than relying on the designated verifier proof. Note that, in reality [36], a threshold encryption scheme is used and decryption has to be performed by multiple collectors. Hence, their scheme can deal with some corrupt collectors. It is also important that the private key of the administrator remains secret. Otherwise an attacker can forge any vote and submit it to the collector.

Process 17 shows a possible V'. To prove receipt-freeness, we need to show

- $V'^{\backslash out(chc,\cdot)} \approx_\ell V_A\{^a/_v\}$, and
- $S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}] \approx_\ell S[V' \mid V_B\{^c/_v\}]$.

where S represents all of the remaining process.

The first labelled bisimulation may be seen informally by considering V' with the "$\mathsf{out}(chc, ...)$" commands removed, and comparing it visually with V_A. To see the second labelled bisimulation, one can informally consider all the executions of each side. S consists of the Main process, and therefore includes processK, the two processA's, and the two processC's, but it has a hole for the two voter processes. As shown above, the hole is filled by $V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}$ on the left and by $V' \mid V_B\{^c/_v\}$ on the right. Executions of $V_A\{^c/_v\}^{chc}$ are matched with those of V'; similarly, $V_B\{^a/_v\}$ on the left is matched with $V_B\{^c/_v\}$ on the right. To illustrate this, we consider a particular execution on the left, and we give the corresponding execution on the right. Here the process P_1 is the one obtained after key distribution. The sequence of names $\tilde{n}$ denotes r_A, r_1, r_B, r_2, r' and also $skvb, skc$ and ska but not $skva$ (coerced voter). We write $pkva$ instead of $pk(skva)$ and assume that public keys are in the frame. We denote by $p_A = penc(c, pkc, f(r_A, r_1))$ and by $p_B = penc(a, pkc, f(r_B, r_2))$.

$$
\begin{aligned}
P_1 &\xrightarrow{\nu x_1.out(ch,x_1)} \nu\tilde{n}.(P_2 \mid \{^{r_A}/_{x_1}\}) \\
&\xrightarrow{\nu x_2.out(ch,x_2)} \nu\tilde{n}.(P_3 \mid \{^{r_A}/_{x_1}\} \mid \{^{(p_A,sign(p_A,ska),dvp(penc(c,pkc,r_A),p_A,r_1,pkva))}/_{x_2}\}) \\
&\xrightarrow{\nu x_3.out(ch,x_3)} \nu\tilde{r}.(P_4 \mid \{^{r_A}/_{x_1}\} \mid \{^{(p_A,sign(p_A,ska),dvp(penc(c,pkc,r_A),p_A,r_1,pkva))}/_{x_2}\} \\
&\qquad\qquad \mid \{^{(p_A,sign(p_A,ska)}/_{x_3}\}) \\
&\xrightarrow{\nu x_4.out(ch,x_4)} \nu\tilde{n}.(P_5 \mid \{^{r_A}/_{x_1}\} \mid \{^{(p_A,sign(p_A,ska),dvp(penc(c,pkc,r_A),p_A,r_1,pkva))}/_{x_2}\} \\
&\qquad\qquad \mid \{^{(p_A,sign(p_A,ska)}/_{x_3}\} \mid \{^{(p_B,sign(p_B,ska))}/_{x_4}\})
\end{aligned}
$$

Similarly, we have that

$$
\begin{aligned}
Q_1 &\xrightarrow{\nu x_1.out(ch,x_1)} \nu\tilde{n}.(Q_2 \mid \{^{r_A}/_{x_1}\}) \\
&\xrightarrow{\nu x_2.out(ch,x_2)} \nu\tilde{n}.(Q_3 \mid \{^{r_A}/_{x_1}\} \mid \{^{(q_A,sign(q_A,ska),dvp(penc(c,pkc,r_A),q_A,r',skva))}/_{x_2}\}) \\
&\xrightarrow{\nu x_3.out(ch,x_3)} \nu\tilde{n}.(Q_4 \mid \{^{r_A}/_{x_1}\} \mid \{^{(q_A,sign(q_A,ska),dvp(penc(c,pkc,r_A),q_A,r',skva))}/_{x_2}\} \\
&\qquad\qquad \mid \{^{(q_A,sign(q_A,ska)}/_{x_3}\}) \\
&\xrightarrow{\nu x_4.out(ch,x_4)} \nu\tilde{n}.(Q_5 \mid \{^{r_A}/_{x_1}\} \mid \{^{(q_A,sign(q_A,ska),dvp(penc(c,pkc,r_A),q_A,r',skva))}/_{x_2}\} \\
&\qquad\qquad \mid \{^{(q_A,sign(q_A,ska)}/_{x_3}\} \mid \{^{(q_B,sign(q_B,ska))}/_{x_4}\})
\end{aligned}
$$

where $q_A = penc(a, pkc, f(r_A, r_1))$ and $q_B = penc(c, pkc, f(r_B, r_2))$.

Note that, the test $\mathsf{checkdvp}(\mathsf{proj}_3(\mathsf{x}_2), \mathsf{penc}(\mathsf{c}, \mathsf{pkc}, \mathsf{x}_1), \mathsf{proj}_1(\mathsf{x}_2), \mathsf{pk}(\mathsf{skva})) = \mathsf{ok}$ is true in both frames. Now, for the input of the collector, we have to consider any public terms. There are essentially two cases. Either the input of both collectors corresponds to the votes submitted by both voters or at least one of the inputs does not. In the last case, since the attacker is not able to provide fake inputs of the expected form, i.e. the input needs to be signed by the administrator, this means that either the collector will block or that both inputs are exactly the same. To prevent the last case, we have to ensure that the collector does not accept a same vote twice. This can be modelled by adding a process in charge of checking double votes and by slightly modifying the processC. The additional process is described in Process 18. In the collector process we add the following instructions just before "synch 2": $\mathsf{out}(\mathsf{privDblChk}, \mathsf{ballot}).\mathsf{in}(\mathsf{privDblChk}, \mathsf{x})$. if $\mathsf{x} = \mathsf{ok}$ then $[\ldots]$ where privDblChk is a restricted channel.

```
doubleCheck =
    in(privDblChk, ballot1). out(privDblChk, ok).
    in(privDblChk, ballot2).
    if ballot1=ballot2 then 0 else out(privDblChk, ok)
```

Process 18. Process to prevent double ballot

We know that if the tests succeeded, both collectors synchronise at phase 2. Up to that point any move of the collector that received the vote of $V_A\{^c/_v\}^{chc}$ on the left-hand side has been imitated on the right-hand side by the collector that received the vote of the voter $V_B\{^c/_v\}$, and similarly for the second collector. The interesting part of the frames obtained after a complete execution is described below.

$$\phi_{P'} \equiv \nu\tilde{n}.\ (\{^{r_A}/_{x_1}\} \mid \{^{(p_A, sign(p_A, ska), dvp(penc(c, pkc, r_A), p_A, r_1, pkva))}/_{x_2}\}$$
$$\mid \{^{(p_A, sign(p_A, ska)}/_{x_3}\} \mid \{^{(p_B, sign(p_B, ska))}/_{x_4}\} \mid \{^a/_{x_5}\} \mid \{^c/_{x_6}\})$$

$$\phi_{Q'} \equiv \nu\tilde{n}.\ (\{^{r_A}/_{x_1}\} \mid \{^{(q_A, sign(q_A, ska), dvp(penc(c, pkc, r_A), q_A, r', skva))}/_{x_2}\}$$
$$\mid \{^{(q_A, sign(q_A, ska)}/_{x_3}\} \mid \{^{(q_B, sign(q_B, ska))}/_{x_4}\} \mid \{^a/_{x_5}\} \mid \{^c/_{x_6}\})$$

Coercion-resistance. We prove coercion resistance by constructing V', which is similar to the one for receipt-freeness. However, for coercion-resistance the coercer also provides the inputs for the messages to send out. Thanks to the fact that

$$S[C[V_A\{^?/_v\}^{c_1, c_2}] \mid V_B\{^a/_v\}] \approx_\ell S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}],$$

we know that the coercer prepares messages corresponding to the given vote c. Hence,

- V' fakes the outputs as in the case of receipt-freeness; the non-coerced voter will counter-balance the outcome, by choosing the vote c;
- V' simply ignores the inputs provided by the coercer.

Such a process V' is shown in Process 19. Similar reasoning to the one used above (for receipt freeness) can be used here, to establish that the conditions

- $C[V']^{\backslash out(chc, \cdot)} \approx_\ell V_A\{^a/_v\}$
- $S[C[V_A\{^?/_v\}^{c_1, c_2} \mid V_B\{^a/_v\}] \approx_\ell S[C[V'] \mid V_B\{^c/_v\}]$,

hold, thus establishing coercion resistance. It is a bit more difficult to perform this reasoning since we have to consider any context $C = \nu c_1.\nu c_2.(\_ \mid P)$ such that $\tilde{n} \cap fn(C) = \emptyset$ and $S[C[V_A\{^?/_v\}^{c_1, c_2}] \mid V_B\{^a/_v\}] \approx_\ell S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}]$.

For the first condition, we can see that if the process $C[V']^{\backslash out(chc,\cdot)}$ does not block then it has the same behaviour as $V_A\{^a/_v\}$ since V' completely ignores the inputs provided by C. The only point is to ensure that V' can fake the outputs to C as in the case of receipt-freeness. This is indeed possible to do so since the voter does not have to know any private data used by the coercer to prepare the messages. (For instance, the voter does not have to know the nonce used by the coercer when he encrypts the vote c.)

To obtain the second condition, it is sufficient to show that the equivalence

$$S[V'' \mid V_B\{^c/_v\}] \approx_\ell S[C[V'] \mid V_B\{^c/_v\}]$$

holds, where V'' is the process provided for receipt-freeness (Process 17). Note that the processes $C[V']$ and V'' are not bisimilar by themselves, because some tests involving messages outputted on $\mathsf{chA1}$ allows us to distinguish them. Indeed, it may be possible that the coercer (i.e. the context C) chooses to generate his own nonce r_c to encrypt his vote c and does not use the one provided by the voter. In such a case, the coercer has to output r_c on the channel chc, and does not forward the nonce provided by the voter, in order to ensure that $S[C[V_A\{^?/_v\}^{c_1,c_2}] \mid V_B\{^a/_v\}] \approx_\ell S[V_A\{^c/_v\}^{chc} \mid V_B\{^a/_v\}]$. This means that the outputs performed on chc by V'' on the left hand-side and by the coercer C on the right hand-side are not quite the same. However, those tests cannot be performed when these processes are put inside the context S, because $\mathsf{chA1}$ is restricted.

8 Conclusion

We have defined a framework for modelling cryptographic voting protocols in the applied pi calculus, and shown how to express in it the properties of vote-privacy, receipt-freeness and coercion-resistance. Within the framework, we can stipulate which parties are assumed to be trustworthy in order to obtain the desired property. We investigated three protocols from the literature. Our results are summarised in Figure 1.

We have proved the intuitive relationships between the three properties: for a fixed set of trusted authorities, coercion-resistance implies receipt-freeness, and receipt-freeness implies vote-privacy.

Our definition of coercion-resistance does not attempt to handle "fault attacks", in which the coercer supplies material which forces the voter to vote randomly, or to vote incorrectly resulting in an abstention (these attacks are respectively called *randomisation* and *forced abstention* attacks in the work of Juels *et al.* [32]). A protocol which succumbs to such attacks could still be considered coercion-resistant according to our definition. In our model, the coercer can count the votes for each candidate, so it seems to be in fact impossible to resist fault attacks fully.

```
processV'=
    (* her private key *)
    in(skvaCh, skv). out(c1, skv).
    (* public keys of administrators *)
    in(pkaCh, pubka). out(c1, pubka).
    in(pkcCh, pubkc). out(c1, pubkc).
    synch 1.
    ν r. out(c1, r).
    let e = penc(a, pubkc, r) in
    (* instruction from the coercer *)
    in(c2, x1).
    let (pi, ei, si) = x1 in
    out(chA1, (pk(skv), e, sign(e, skv))).

    (* message from the administrator *)
    in(chA1, m2).
    let (re, sa, dvpV) = m2 in
    if checkdvp(dvpV, e, re, pk(skv)) = ok then
    ν r'.
    let fk = dvp(ei, re, r', skv) in
    out(c1, (re, sa, fk)).
    if checksign(sa, pubka) = re then
    in(c2, x2). out(ch, (re, sa))
```

Process 19. Process V' - coercion-resistance

Property	*Fujioka et al.*	*Okamoto et al.*	*Lee et al.*
Vote-privacy	✓	✓	✓
trusted authorities	none	timeliness mbr.	administrator
Receipt-freeness	×	✓	✓
trusted authorities	n/a	timeliness mbr.	admin. & collector
Coercion-resistance	×	×	✓
trusted authorities	n/a	n/a	admin. & collector

Fig. 1: Summary of protocols and properties

Our reasoning about bisimulation in applied pi is rather informal. In the future, we hope to develop better techniques for formalising and automating this reasoning. The ProVerif tool goes some way in this direction, but the technique it uses is focused on process which have the same structure and differ only in the choice of terms [10]. The sort of reasoning we need in this paper often involves a bisimulation relation which does not follow the structure of the processes. For example, in proving vote-privacy for Fujioka *et al.*, early on we match $V_A\{^a/_v\}$ on the left-hand side with $V_A\{^b/_v\}$ on the right-hand side, while later we match $V_A\{^a/_v\}$ on the left with $V_B\{^a/_v\}$ on the right. It would be useful to automate this kind of reasoning, or to investigate more general and more powerful methods for establishing bisimulation. Symbolic reasoning has proved successful for reachability properties [38,6], in which terms input from the environment are represented as symbolic variables, together with some constraints. One direction we are investigating is the development of symbolic bisimulation and corresponding decision procedures for the finite applied pi calculus. This work has been initiated in [20].

Our definition of coercion-resistance involves quantification over all possible contexts which satisfy a certain condition, and this makes it hard to work with in practice. Coercion-resistance may thus be seen as a kind of observational equivalence but with a restriction on the powers of the observer. Our earlier paper [19] included a notion which we called *adaptive simulation*, a variant of bisimulation which attempts to model the coerced voter's ability to adapt her vote according to the instructions of the coercer. Unfortunately, we have found this notion to have some undesirable properties, and we have not used it in this paper. In the future, we hope to find a corresponding restriction of labelled bisimilarity, which will help us to reason with coercion-resistance more effectively. In this vein, some new work has appeared [4] that builds on our definitions and avoids the universal quantification over contexts.

Acknowledgments Michael Clarkson read our CSFW paper [19] and asked us several challenging questions, which were instrumental in helping us prepare this paper. Anonymous reviewers of this journal article provided many detailed comments which were very useful in helping us to improve its quality.

References

[1] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proc. 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, London, UK, 2001. ACM.

[2] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[3] Martn Abadi, Bruno Blanchet, and Cdric Fournet. Just fast keying in the pi calculus. *ACM Transactions on Information and System Security (TISSEC)*, 10(3):1–59, 2007.

[4] Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Proc. 21st IEEE Computer Security Foundations Symposium, (CSF'08)*, pages 195–209. IEEE Comp. Soc. Press, 2008.

[5] A. Baskar, R. Ramanujam, and S.P. Suresh. Knowledge-based modelling of voting protocols. In *Proc. 11th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'07)*, pages 62–71, 2007.

[6] Mathieu Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 16–25, Alexandria, Virginia, USA, 2005. ACM Press.

[7] Josh Benaloh. *Verifiable Secret Ballot Elections*. PhD thesis, Yale University, 1987.

[8] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proc. 26th Symposium on Theory of Computing (STOC'94)*, pages 544–553. ACM Press, 1994.

[9] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Comp. Soc. Press, 2001.

[10] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[11] Ran Canetti and Rosario Gennaro. Incoercible multiparty computation. In *Proc. 37th Symposium on Foundations of Computer Science (FOCS'96)*, pages 504–513. IEEE Comp. Soc. Press, 1996.

[12] Konstantinos Chatzikokolakis and Catuscia Palamidessi. Probable innocence revisited. In *Proc. 3rd Formal Aspects in Security and Trust (FAST'05)*, volume 3866 of *LNCS*, pages 142–157. Springer, 2006.

[13] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and P. Panangaden. Anonymity protocols as noisy channels. In *Proc. 2nd Symposium on Trustworthy Global Computing (TGC'06)*, volume 4661 of *LNCS*, pages 281–300. Springer, 2007.

[14] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[15] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology – CRYPTO'82*, pages 199–203. Plenum Press, 1983.

[16] David Chaum. Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA. In *Advances in Cryptology – Eurocrypt'88*, volume 330 of *LNCS*, pages 177–182. Springer, 1988.

[17] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security and Privacy*, 2(1):38–47, 2004.

[18] David Chaum, Peter Y. A. Ryan, and Steve Schneider. A practical, voter-verifiable election scheme. In *Proc. 10th European Symposium On Research In Computer Security (ESORICS'05)*, volume 3679 of *LNCS*, pages 118–139. Springer, 2005.

[19] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Proc. 19th Computer Security Foundations Workshop (CSFW'06)*, pages 28–39. IEEE Comp. Soc. Press, 2006.

[20] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Symbolic bisimulation for the applied pi-calculus. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *LNCS*, pages 133–145. Springer, 2007.

[21] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In *Proc. 2nd International Workshop on Privacy Enhancing Technologies (PET'02)*, volume 2482 of *LNCS*, pages 54–68. Springer, 2002.

[22] Ariel J. Feldman, J. Alex Halderman, and Edward W. Felten. Security analysis of the Diebold accuvote-ts voting machine. `http://itpolicy.princeton.edu/voting/`, 2006.

[23] Marc Fischlin. *Trapdoor Commitment Schemes and Their Applications*. PhD thesis, Fachbereich Mathematik Johann Wolfgang Goethe-Universität Frankfurt am Main, 2001.

[24] Cédric Fournet and Martín Abadi. Hiding names: Private authentication in the applied pi calculus. In *Proc. International Symposium on Software Security (ISSS'02)*, volume 2609 of *LNCS*, pages 317–338. Springer, 2003.

[25] Atsushi Fujioka, Tatsuaki Okamoto, and Kazui Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology – AUSCRYPT '92*, volume 718 of *LNCS*, pages 244–251. Springer, 1992.

[26] Rop Gonggrijp, Willem-Jan Hengeveld, Andreas Bogk, Dirk Engling, Hannes Mehnert, Frank Rieger, Pascal Scheffers, and Barry Wels. Nedap/Groenendaal ES3B voting computer: a security analysis. `www.wijvertrouwenstemcomputersniet.nl/other/es3b-en.pdf`. Retrieved 24 October 2007.

[27] Joseph Y. Halpern and Kevin R. O'Neill. Anonymity and information hiding in multiagent systems. *Journal of Computer Security*, 13(3):483–512, 2005.

[28] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptography – Eurocrypt'00*, volume 1807 of *LNCS*, pages 539–556. Springer, 2000.

[29] Hugo L. Jonker and Erik P. de Vink. Formalising Receipt-Freeness. In *Proc. Information Security (ISC'06)*, volume 4176 of *LNCS*, pages 476–488. Springer, 2006.

[30] Hugo L. Jonker and Wolter Pieters. Receipt-freeness as a special case of anonymity in epistemic logic. In *Proc. AVoSS Workshop On Trustworthy Elections (WOTE'06)*, 2006.

[31] Wen-Shenq Juang and Chin-Laung Lei. A secure and practical electronic voting scheme for real world environments. *IEICE Transaction on Fundamentals of Electronics, Communications and Computer Science, E80A*, 1:64–71, January 1997.

[32] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proc. Workshop on Privacy in the Electronic Society (WPES'05)*. ACM Press, 2005.

[33] Detlef Kähler, Ralf Küsters, and Thomas Wilke. A Dolev-Yao-based Definition of Abuse-free Protocols. In *Proc. 33rd International Colloqium on Automata, Languages, and Programming (ICALP'06)*, volume 4052 of *LNCS*, pages 95–106. Springer, 2006.

[34] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *Proc. 25th IEEE Symposium on Security and Privacy (SSP'04)*, pages 27–28. IEEE Comp. Soc. Press, 2004.

[35] Steve Kremer and Mark D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In *Proc. 14th European Symposium On Programming (ESOP'05)*, volume 3444 of *LNCS*, pages 186–200. Springer, 2005.

[36] Byoungcheon Lee, Colin Boyd, Ed Dawson, Kwangjo Kim, Jeongmo Yang, and Seungjae Yoo. Providing receipt-freeness in mixnet-based voting protocols. In *Proc. Information Security and Cryptology (ICISC'03)*, volume 2971 of *LNCS*, pages 245–258. Springer, 2004.

[37] Sjouke Mauw, Jan H.S. Verschuren, and Erik P. de Vink. A formalization of anonymity and onion routing. In *Proc. 9th European Symposium on Research Computer Security (ESORICS'04)*, volume 3193 of *LNCS*, pages 109–124. Springer, 2004.

[38] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*, pages 166–175. ACM Press, 2001.

[39] Christoffer Rosenkilde Nielsen, Esben Heltoft Andersen, and Hanne Riis Nielson. Static analysis of a voting protocol. In *Proc. 2nd Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA'05)*, 2005.

[40] Tatsuaki Okamoto. An electronic voting scheme. In *Proc. IFIP World Conference on IT Tools*, pages 21–30, 1996.

[41] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Proc. 5th Int. Security Protocols Workshop*, volume 1361 of *LNCS*, pages 25–35. Springer, 1997.

[42] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.

[43] Steve Schneider and Abraham Sidiropoulos. CSP and anonymity. In *Proc. 4th European Symposium On Research In Computer Security (ESORICS'96)*, volume 1146 of *LNCS*, pages 198–218. Springer, 1996.

[44] Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In *Proc. 2nd International Workshop on Privacy Enhancing Technologies (PET'02)*, volume 2482 of *LNCS*, pages 41–53. Springer, 2002.

[45] Vitaly Shmatikov. Probabilistic analysis of anonymity. In *Proc. 15th Computer Security Foundations Workshop (CSFW'02)*, pages 119–128. IEEE Comp. Soc. Press, 2002.

[46] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *Proc. 18th IEEE Symposium on Security and Privacy (SSP'97)*, pages 44–54. IEEE Comp. Soc. Press, 1997.

Appendix A Proof of Lemma 14

Lemma 14 *Let P be a closed plain process and ch a channel name such that $ch \notin fn(P) \cup bn(P)$. We have $(P^{ch})^{\backslash out(ch,\cdot)} \approx_\ell P$.*

PROOF. Let P be a closed plain process. We show by induction on the size of P that for any channel name ch such that $ch \notin fn(P) \cup bn(P)$ we have $P^{ch\backslash out(ch,\cdot)} \approx_\ell P$. The size of the null process is defined to be 0. Prefixing the process P by a restriction, an input or an output or putting it under a replication adds 1 to its size. The size of the process $P \mid Q$ (resp. if $M = N$ then P else Q) is the sum of the size of P and Q plus 1.

The base case where $P = 0$ is trivial. Let ch be a channel name such that $ch \notin fn(P) \cup bn(P)$. The possibilities for building P are the following:

- $P = P_1 \mid P_2$. In such a case, we have:

$$\begin{array}{rll}
P^{ch\backslash out(ch,.)} & \hat{=} & ({P_1}^{ch} \mid {P_2}^{ch})^{\backslash out(ch,.)} \\
 & \hat{=} & \nu ch.({P_1}^{ch} \mid {P_2}^{ch} \mid !\mathsf{in}(ch,x)) \\
 & \approx_\ell & \nu ch.({P_1}^{ch} \mid !\mathsf{in}(ch,x)) \mid \nu ch.((P_2)^{ch} \mid !\mathsf{in}(ch,x)) \\
 & & \qquad \text{since } \mathsf{in}(ch,.) \text{ occurs neither in } P_1^{ch} \text{ nor in } P_2^{ch} \\
 & \approx_\ell & P_1^{ch\backslash out(ch,.)} \mid P_2^{ch\backslash out(ch,.)} \\
 & \approx_\ell & P_1 \mid P_2 \qquad \text{by induction hypothesis} \\
 & = & P
\end{array}$$

- $P = \nu n.P_1$. We have:

$$\begin{array}{rll}
P^{ch\backslash out(ch,.)} & = & (\nu n.P_1)^{ch\backslash out(ch,.)} \\
 & \hat{=} & \nu ch.(\nu n.\mathsf{out}(ch,n).{P_1}^{ch} \mid !\mathsf{in}(ch,x)) \\
 & \approx_\ell & \nu ch.(\nu n.{P_1}^{ch} \mid !\mathsf{in}(ch,x)) \\
 & \equiv & \nu n.\nu ch.({P_1}^{ch} \mid !\mathsf{in}(ch,x)) \qquad \text{since } n \neq ch \\
 & \hat{=} & \nu n.{P_1}^{ch\backslash out(ch,.)} \\
 & \approx_\ell & \nu n.P_1 \qquad \text{by induction hypothesis} \\
 & = & P
\end{array}$$

- $P = \mathsf{in}(c, y).P_1$. Note that $c \neq ch$. We have:

$$
\begin{array}{rcl}
P^{ch\backslash out(ch,.)} & = & (\mathsf{in}(c, y).P_1)^{ch\backslash out(ch,.)} \\
& \widehat{=} & \nu ch.(\mathsf{in}(c, y).\mathsf{out}(ch, y).P_1^{ch} \mid !\mathsf{in}(ch, x)) \\
& \approx_\ell & \mathsf{in}(c, y).\nu ch.(\mathsf{out}(ch, y).P_1^{ch} \mid !\mathsf{in}(ch, x)) \\
& \approx_\ell & \mathsf{in}(c, y).\nu ch.(P_1^{ch} \mid !\mathsf{in}(ch, x)) \\
& \widehat{=} & \mathsf{in}(c, y).P_1^{ch\backslash out(ch,.)} \\
& \approx_\ell & \mathsf{in}(c, y).P_1
\end{array}
$$

  To establish the last step, we can see that for any ground term M, the processes Q_1 and Q_2 such that $\mathsf{in}(c, y).P_1^{ch\backslash out(ch,.)} \xrightarrow{in(c,M)} Q_1$ and $\mathsf{in}(c, y).P_1 \xrightarrow{in(c,M)} Q_2$ are such that $Q_1 \equiv P_1\{^M/_y\}^{ch\backslash out(ch,.)}$ and $Q_2 \equiv P_1\{^M/_y\}$. By induction hypothesis, we have that Q_1 and Q_2 are bisimilar. Note that for this step we assume that w.l.o.g. $ch \notin fv(M)$. This can always be obtained by α-renaming ch. Lastly, we conclude thanks to the fact that $\mathsf{in}(c, y).P_1 = P$.
- $P = \mathsf{out}(c, M).P_1$. Note that $c \neq ch$. We have:

$$
\begin{array}{rcll}
P^{ch\backslash out(ch,.)} & = & (\mathsf{out}(c, M).P_1)^{ch\backslash out(ch,.)} & \\
& \widehat{=} & \nu ch.(\mathsf{out}(c, M).P_1^{ch} \mid !\mathsf{in}(ch, x)) & \\
& \approx_\ell & \mathsf{out}(c, M).\nu ch.(P_1^{ch} \mid !\mathsf{in}(ch, x)) & \\
& \widehat{=} & \mathsf{out}(c, M).P_1^{ch\backslash out(ch,.)} & \\
& \approx_\ell & \mathsf{out}(c, M).P_1 & \text{by induction hypothesis} \\
& = & P &
\end{array}
$$

- $P = !P_1$. In such a case, we have:

$$
\begin{array}{rcll}
P^{ch\backslash out(ch,.)} & \widehat{=} & (!P_1)^{ch\backslash out(ch,.)} & \\
& \widehat{=} & \nu ch.(!P_1^{ch} \mid !\mathsf{in}(ch, x)) & \\
& \approx_\ell & \nu ch.!(P_1^{ch} \mid !\mathsf{in}(ch, x)) & \\
& \approx_\ell & !(\nu ch.(P_1^{ch} \mid !\mathsf{in}(ch, x))) & \text{since } \mathsf{in}(ch, .) \text{ does not occur in } P_1^{ch} \\
& \widehat{=} & !P_1^{ch\backslash out(ch,.)} & \\
& \approx_\ell & !P_1 & \text{by induction hypothesis} \\
& = & P &
\end{array}
$$

- $P = \text{if } M = N \text{ then } P_1 \text{ else } P_2$. Hence, we have:

$$
\begin{aligned}
P^{ch\setminus out(ch,.)} &= (\text{if } M = N \text{ then } P_1 \text{ else } P_2)^{ch\setminus out(ch,.)}\\
&\hat{=}\ \nu ch.(\text{if } M = N \text{ then } P_1^{ch} \text{ else } P_2^{ch} \mid !\mathsf{in}(ch,x))\\
&\approx_\ell \nu ch.(\text{if } M = N \text{ then } (P_1^{ch} \mid !\mathsf{in}(ch,x) \text{ else } (P_2^{ch} \mid !\mathsf{in}(ch,x)))\\
&\approx_\ell \nu ch.(\text{if } M = N \text{ then } (P_1^{ch} \mid !\mathsf{in}(ch,x) \text{ else } (P_2^{ch} \mid !\mathsf{in}(ch,x)))\\
&\approx_\ell \text{if } M = N \text{ then } \nu ch.(P_1^{ch} \mid !\mathsf{in}(ch,x)) \text{ else } \nu ch.(P_2^{ch} \mid !\mathsf{in}(ch,x))\\
&\qquad\qquad \text{since } \mathsf{in}(ch,.) \text{ occurs neither in } P_1^{ch} \text{ nor in } P_2^{ch}\\
&\hat{=}\ \text{if } M = N \text{ then } P_1^{ch\setminus out(ch,.)} \text{ else } P_2^{ch\setminus out(ch,.)}\\
&\approx_\ell \text{if } M = N \text{ then } P_1 \text{ else } P_2\\
&= P
\end{aligned}
$$

This last case conludes the proof. □