Verifying privacy-type properties of electronic voting protocols: a taster [⋆]

Stéphanie Delaune[1], Steve Kremer[1], and Mark Ryan[2]

[1] LSV, ENS Cachan & CNRS & INRIA, France
[2] School of Computer Science, University of Birmingham, UK

Abstract. While electronic elections promise the possibility of convenient, efficient and secure facilities for recording and tallying votes, recent studies have highlighted inadequacies in implemented systems. These inadequacies provide additional motivation for applying formal methods to the validation of electronic voting protocols.

In this paper we report on some of our recent efforts in using the applied pi calculus to model and analyse properties of electronic elections. We particularly focus on anonymity properties, namely vote-privacy and receipt-freeness. These properties are expressed using observational equivalence and we show in accordance with intuition that receipt-freeness implies vote-privacy.

We illustrate our definitions on two electronic voting protocols from the literature. Ideally, these properties should hold even if the election officials are corrupt. However, protocols that were designed to satisfy privacy or receipt-freeness 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.

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.

[⋆] This work has been partly supported by the EPSRC projects EP/E029833, *Verifying Properties in Electronic Voting Protocols* and EP/E040829/1, *Verifying anonymity and privacy properties of security protocols*, the ARA SESUR project AVOTÉ and the ARTIST2 NoE.

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. *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.

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 use *the applied pi calculus* as our basic modelling formalism [2], 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 [5]. 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.

Outline of the paper. In Section 2, we recall the basic ideas and concepts of the applied pi calculus. Next, in Section 3, we present the framework for formalising voting protocols from the literature, and in Section 4 we show how two of our three privacy-like properties are formalised, namely vote-privacy and receipt-freeness. (We omit the formalisation of coercion-resistance for reasons of space; see [9].) In Sections 5 and 6, we recall two voting protocols from the literature, and show how they can be formalised in our framework. We analyse which of the properties they satisfy. This paper summarises and provides a taster for a much longer paper currently being submitted to the *Journal of Computer Security* [9]. In this paper we intend to give the flavour of our work without going into great detail.

2 The applied pi calculus

The applied pi calculus [2] is a language for describing concurrent processes and their interactions. It is based on an earlier language called the pi calculus, which has enjoyed a lot of attention from computer scientists over the last decades because of its simplicity and mathematical elegance. The applied pi calculus is intended to be much richer than the pi calculus, while keeping its mathematical rigour, and is therefore more convenient to use in real applications. The applied pi calculus is similar to another pi calculus derivative called the spi calculus [3], but 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 a wide class of primitives by means of an equational theory. This is useful in electronic voting protocols, where the cryptography is often sophisticated and purpose-built. The applied pi calculus has been used to study a variety of security protocols, such as a private authentication protocol [11] or a key establishment protocol [1].

2.1 Syntax and informal semantics

Messages. To describe processes in the applied pi calculus, one starts with a infinite set of *names* (which are used to name communication channels or other atomic data), an infinite 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. When the set of variables occurring in a term T is empty, we say that T is *ground*.

Example 1. Let $\Sigma = \{\mathsf{enc}, \mathsf{dec}\}$, where enc and dec are each of arity 2. Suppose a, b, c are names (perhaps representing some bitstring constants or keys), and x, y, z are variables. Then $\mathsf{enc}(a, b)$ is a ground term (which represents the encryption of a using the key b). The term $\mathsf{dec}(\mathsf{enc}(a, b), y)$ is also a term (but not a ground term), representing the decryption by y of the result of encrypting a with b. The symbols enc and dec may be nested arbitrarily.

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. Two terms are related by $=_{\mathsf{E}}$ only if that fact can be derived from the equations in E.

Example 2. A typical example of an equational theory useful for cryptographic protocols is $\mathsf{dec}(\mathsf{enc}(x, y), y) = x$. In this equational theory, we have that the terms $T_1 := \mathsf{dec}(\mathsf{enc}(\mathsf{enc}(n, k), k'), k')$ and $T_2 := \mathsf{enc}(n, k)$ are equal, i.e. $T_1 =_{\mathsf{E}} T_2$, while obviously the syntactic equality $T_1 = T_2$ does not hold.

Equational theories are the means by which we represent cryptographic operations. We do not model the mechanisms (whether bitstring manipulation or numerical calculation) that constitute the cryptographic operations. Rather, we model the behaviour they are designed to exhibit. Thus, stipulating the equation $\mathsf{dec}(\mathsf{enc}(x, y), y) = x$ models symmetric encryption. In the model terms are unequal unless they can be proved equal by the equations. This means that the only way of recovering x from $\mathsf{enc}(x, y)$ is by the application of $\mathsf{dec}(\cdot, y)$ (and in particular, the agent that makes that application is required to know the key y).

If M and N are terms, then the pair (M, N) is a term, and from it may be extracted the components M and N. Formally, this requires us to introduce the binary "pairing" function $(\cdot, \cdot)$ and the projection functions proj_1 and proj_2, but usually we don't bother with that and keep the equational theory for pairs (and tuples of any finite length) implicit.

Processes. In order to model the dynamic part of protocols, we require processes. In applied pi, there are two kinds of processes, namely *plain processes*, denoted by P, Q, R and *extended processes*, denoted by A, B, C. 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.

$$
\begin{array}{ll}
P, Q, R := & \text{plain processes} \\
& 0 \\
& \mathsf{in}(u, x).P \\
& \mathsf{out}(u, N).P \\
& \text{if } M = N \text{ then } P \text{ else } Q \\
& P \mid Q \\
& !P \\
& \nu n.P
\end{array}
$$

$$
\begin{array}{ll}
A, B, C := & \text{extended processes} \\
& P \\
& A \mid B \\
& \nu n.A \\
& \nu x.A \\
& \{^{M}/_{x}\}
\end{array}
$$

The process 0 is the plain process that does nothing. The process $\mathsf{in}(u, x).P$ waits to receive a message on the channel u, and then continues as P but with x replaced by the received message. The process $\mathsf{out}(u, N).P$ outputs a term N on the channel u, and then continues as P. The process *if* $M = N$ *then* P *else* Q runs as P if the ground terms M and N are equal in the equational theory, and otherwise as Q. If there is no "else", it means "else 0". The process $P \mid Q$ runs P and Q in parallel. The process $!P$ executes P some finite number of times. The restriction νn is used to model the creation in a process of new random numbers (e.g., nonces or key material), or of new private channels. The process $\nu n.P$ is the process that invents a new name n and continues as P.

Extended processes add *active substitutions* (the process $\{^{M}/_{x}\}$), restriction on names νn, and restriction on variables νx. Active substitutions are the notation that is used to denote a process that has output a term. Consider the process $\mathsf{out}(c, N).P$, where c is a channel name, N is some term, and P is some continuation process. If $\mathsf{out}(c, N).P$ is allowed to run in an environment, it will become the process $P \mid \{^{N}/_{x}\}$, which means the process that can now run as P, and has output the term N. We do not retain the name of the channel name c, but we do give a handle name, here x, to the value that was output. The environment may now refer to the term N as x.

The handle x is important when the environment cannot itself describe the term that was output, except by referring to it as the term that was output (i.e., by the handle x). Consider the process $\nu k.\mathsf{out}(c, \mathsf{enc}(a, k)).P$ which creates a new key k and then outputs the name a encrypted with k. Here, a is a "free name" (modelling some well-known value) rather than a restricted name (like k) that was created by the process using the ν operator. The process $\nu k.\mathsf{out}(c, \mathsf{enc}(a, k)).P$ can output the term on the channel c, resulting in the process $\nu k.(P \mid \{^{\mathsf{enc}(a,k)}/_{x}\})$. In this process, the environment has the term $\mathsf{enc}(a, k)$, but it doesn't have k since the process hasn't output k. The environment can refer to the term $\mathsf{enc}(a, k)$ as x.

The syntax of extended processes also allows restriction νx on variables x. The combination of νx and active substitutions generalise the familiar "let" operator from many functional programming languages. We define "let $x = M$ in P" as an abbreviation of $\nu x.(\{^{M}/_{x}\} \mid P)$.

A process can perform an input and then test the value of the input for equality (modulo E) with some other term; for example, $\mathsf{in}(u, x)$. if $x = M$ then P. Suppose that after checking the input the process makes no further use it (i.e., x does not occur in P). This idiom is quite common, so we abbreviate it as $\mathsf{in}(u, =M).P$.

An *evaluation context* $C[\_]$ is an extended process with a hole $\_$ instead of an extended process; this is useful for describing part (e.g. the beginning) of a process, while leaving the hole to represent the other part that will be filled in later. Names and variables have scopes, which are delimited by restrictions νx and νn, and by inputs $\mathsf{in}(u, x)$. 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 stipulate 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.

2.2 Semantics

The operational semantics of processes in the applied pi calculus is defined by structural rules defining two relations: *structural equivalence*, noted $\equiv$, and *internal reduction*, noted $\rightarrow$.

Structural equivalence takes account of the fact that the syntax of processes necessarily makes distinctions that are not important. For example, $P \mid Q$ looks different from $Q \mid P$ but that difference is purely syntactic, and not important, so we say that $P \mid Q$ and $Q \mid P$ are structurally equivalent. Formally, structural equivalence is the smallest equivalence relation $\equiv$ on extended processes that is closed under α-conversion on names and variables (that is, renaming a bound name or variable), application of evaluation contexts, and some other standard rules such as associativity and commutativity of the parallel operator and commutativity of the bindings. In addition the following three rules are related to active substitutions and equational theories.

$$\begin{array}{r}
\nu x.\{^{M}/_{x}\} \equiv 0 \\
\{^{M}/_{x}\} \mid A \equiv \{^{M}/_{x}\} \mid (A\{^{M}/_{x}\}) \\
\{^{M}/_{x}\} \equiv \{^{N}/_{x}\} \quad \text{if } M =_{\mathsf{E}} N
\end{array}$$

where, in the second equivalence, $A\{^{M}/_{x}\}$ means A but with free occurrences of x replaced by M. Note the absence of the $|$. In $A\{^{M}/_{x}\}$, the substitution is not an active substitution, but a normal "metasyntactic" substitution; it tells the reader to perform the substitution.

Example 3. Consider the following process P:

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

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:

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

Internal reduction is the smallest relation on extended processes closed under structural equivalence and application of evaluation contexts such that

$$\begin{array}{lrcl}
(\textsc{Comm}) & \mathsf{out}(a,x).P \mid \mathsf{in}(a,x).Q & \rightarrow & P \mid Q \\
(\textsc{Then}) & \text{if } M = M \text{ then } P \text{ else } Q & \rightarrow & P \\
(\textsc{Else}) & \text{if } M = N \text{ then } P \text{ else } Q & \rightarrow & Q \\
 & \multicolumn{3}{r}{\text{for any ground terms } M \text{ and } N \text{ such that } M \neq_{\mathsf{E}} N.}
\end{array}$$

This definition looks more restrictive than it is, thanks to structural equivalence. It is straightforward to prove that $\mathsf{out}(a,M).P \mid \mathsf{in}(a,x).Q \rightarrow P \mid Q\{^M/_x\}$ and *if* $M = N$ *then* P *else* $Q \rightarrow P$ in the case that $M =_{\mathsf{E}} N$.

The applied pi calculus has another kind of transition operation, called *labelled reduction*, denoted $\xrightarrow{\alpha}$, where α is a label. We don't define that formally here, but refer the reader to our full paper [9] or the applied pi calculus paper [2].

2.3 Observational equivalence

Now we are able to define *observational equivalence.* This relation is important to understand how properties are defined in applied pi calculus. We write $A \Downarrow a$ when A can send a message on a, that is, when $A \rightarrow^* C[\mathsf{out}(a,M).P]$ for some evaluation context $C[\_]$ that does not bind a.

Definition 1. Observational equivalence *($\approx$) is the largest symmetric relation $\mathcal{R}$ between closed extended processes with the same domain such that $A \ \mathcal{R}\ B$ implies:*

1. *if $A \Downarrow a$, then $B \Downarrow a$;*
2. *if $A \rightarrow^* A'$, then $B \rightarrow^* B'$ and $A' \ \mathcal{R}\ B'$ for some B';*
3. *$C[A] \ \mathcal{R}\ C[B]$ for all closing evaluation contexts $C[\_]$.*

Intuitively, two processes are observationally equivalent if they cannot be distinguished by any active attacker represented by any context.

Example 4. Let E be the theory defined by the axiom $\mathsf{dec}(\mathsf{enc}(x,y),y) = x$. Consider the processes $P_0 = \mathsf{out}(c, \mathsf{enc}(s_0,k))$ and $Q_0 = \mathsf{out}(c, \mathsf{enc}(s_1,k))$. We have that $\nu k.P_0 \approx \nu k.Q_0$; intuitively, the attacker cannot distinguish between the encryption of two known values s_0 and s_1 where the encryption is by a secret key. Technically, there is no context C that, given these processes, can distinguish them, e.g., by taking some observable action in the case of P_0 but not in the case of Q_0. If the key k is available to the attacker, of course the situation changes. We have $P_0 \not\approx Q_0$, since the context

$$C[\_] = \mathsf{in}(c,x).\, \text{if } \mathsf{dec}(x,k) = s_0 \text{ then } \mathsf{out}(c, \text{“Found } s_0\text{!”}) \mid \_$$

distinguishes P_0 and Q_0.

Observational equivalence can be used to formalise many interesting security properties, in particular anonymity properties, such as those studied in this paper (see Section 4). However, proofs of observational equivalences are difficult because of the universal quantification over all contexts. In [9], our definitions and proofs rely on *labelled bisimulation* which has been shown to coincide with observational equivalence [2]. Labelled bisimulation has the advantage of being more convenient to manipulate in proofs. Its definition relies on two further notions: *static equivalence* and *labelled reduction*. To avoid additional definitions and ease the presentation we stick to observational equivalence in this paper.

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 2 (Voting process). *A voting process is a closed plain process*

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

V is the template voter process, and the $V\sigma_i$ are the actual voter processes (the substitution σ_i provides the voter's identity). The A_js are the election authorities that 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). 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.

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. Actually, 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 [15]. However, the definition of anonymity properties in the context of voting protocols is rather subtle.

4.1 Vote-privacy

The privacy property aims to guarantee that the link between a given voter V and his vote v remains hidden. 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 device 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 and 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 3. *A voting protocol respects* vote-privacy *(or just* privacy*) if*

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

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.

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 2 of VP (and hence S).

4.2 Receipt-Freeness

Similarly to privacy, receipt-freeness may be formalised as an 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: inputs and freshly generated names of base type (i.e. not channel 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 are forwarded but not the secret channel name on which the keys are received).

Definition 4. *Let P be a plain process and ch be a channel name. We define the process P^{ch} as follows:*

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

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 5. *Let A be an extended process.*

$$A^{\backslash out(ch,\cdot)} \mathrel{\widehat{=}} \nu ch.(A\ |!\mathsf{in}(ch, x)).$$

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

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

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 6.

Note that "receipt-freeness" does not preclude voting systems which give some kind of receipt to the voter that cannot be used for proving how she voted.

Intuition suggests an implication relation between receipt-freeness and vote-privacy, which indeed holds and is formally proved in [9]:

> *If a protocol is receipt free (for a given set of honest authorities), then it also respects vote-privacy (for the same set).*

5 Protocol due to Fujioka, Okamoto and Ohta

In this section we study a protocol due to Fujioka, Okamoto and Ohta [12].

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; most anonymiser mechanisms could be suitable depending on the precise context the protocol is used in. One can use MIX-nets introduced by Chaum [7] 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 [16].

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 [8] are used, i.e. the administrator does not learn the commitment of the vote. (Throughout, we assume signatures with message recovery.)

- The voter V selects a vote v, computes the commitment $x = \xi(v,r)$ using a random key r, computes the message $e = \chi(x, b)$ using a blinding function χ and a random blinding factor b, and finally digitally signs e and sends her signature $\sigma_V(e)$ to the administrator A together with her identity.
- The administrator 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 *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

We only give the interesting parts of the modelling. A complete formalisation can be found in [9].

```
(* 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

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 modelled 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 third). We enforce this in our model by the keyword synch. When a process encounters $\mathsf{synch}\ \mathsf{n}$, it waits until all the other process that could encounter $\mathsf{synch}\ \mathsf{n}$ arrive at that point too. Then all the processes are allowed to continue. If there are k processes that can encounter $\mathsf{synch}\ \mathsf{n}$, we can implement the synchronisation as follows. The command $\mathsf{synch}\ \mathsf{n}$ 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

```
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 2. Voter process

distribution. We only model the protocol for two voters and launch two copies of the administrator and collector process, one for each voter.

Voter (Process 2). 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.

Our model also includes a dedicated process for generating and distributing keying material modelling a PKI (processK), a process for the administrator and another one for the collector (those processes are not given here, see [9]).

5.3 Analysis

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

$$S[V_A\{^a/_v\} \mid V_B\{^b/_v\}] \approx S[V_A\{^b/_v\} \mid V_B\{^a/_v\}] \tag{1}$$

where $V_A = \mathsf{processV}\{^{skvaCh}/_{skvCh}\}$, $V_B = \mathsf{processV}\{^{skvbCh}/_{skvCh}\}$. 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. However, we have to ensure that both voters use the same public key for the administrator. Therefore, we send this public key on a private channel (pkaCh1), although the public key and its counterpart are known by the attacker. Actually, we show that

$$\begin{array}{c} \nu\mathsf{pkaCh1}.(V_A\{^a/_v\} \mid V_B\{^b/_v\} \mid \mathsf{processK}) \\ \approx \\ \nu\mathsf{pkaCh1}.(V_A\{^b/_v\} \mid V_B\{^a/_v\} \mid \mathsf{processK}) \end{array} \tag{2}$$

The proof, detailed in [9], uses the (equivalent) definition of labelled bisimulation instead of observational equivalence. We were able to automate parts of the proof (the static equivalence relations) using the ProVerif tool [5]. The remaining of the proof (the bisimulation part) is established manually by considering all cases. Although ProVerif provides observation equivalence checking, it was unable to perform the proof in this case: observation equivalence checking being undecidable, ProVerif aims at proving a finer relation which relies on easily matching up the execution paths of the two processes [6]. This relation happens to be too fine for proving equivalence 2. Our proof relies on matching $V_A\{^a/_v\}$ on the left-hand side with $V_A\{^b/_v\}$ on the right-hand side during the first stage (before $\mathsf{synch}\ 1$) and matching $V_A\{^a/_v\}$ on the left with $V_B\{^a/_v\}$ on the right in the following (after $\mathsf{synch}\ 1$).

As mentioned above, 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.

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:

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

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.

Note that 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 [13] which was designed to be incoercible. However, Okamoto himself shows a flaw [14]. 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 [4].

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* [10] 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;

```
(* 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 3. Main process

- T publishes the list of the votes 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 the fact that a commitment $\mathsf{tdcommit}(\mathsf{m_1},\mathsf{r},\mathsf{td})$ can be viewed as a commitment of any value $\mathsf{m_2}$. However, to open this commitment as $\mathsf{m_2}$ 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.

```
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 4. Voter process

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

Process 5. Timeliness process

Main (Process 3). Again, 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.

Voter (Process 4). 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.

Timeliness Member (Process 5). The timeliness member receives, through chT, messages of the form (vt, rt, 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.

```
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 6. V'- Receipt-freeness

We have also a dedicated process for generating and distributing keying material modelling a PKI, an administrator process and a collector. Those processes are not given here.

6.3 Analysis

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. Thus, our analysis is entirely manual. We only discuss receipt-freeness (since this implies vote privacy).

Receipt-freeness. To establish 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 6. To prove receipt-freeness, we need to show that

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

The context S we consider is $\nu\mathsf{pkaCh1}.\nu\mathsf{chT}.(\ \_\ \mid \mathsf{processK} \mid \mathsf{processT} \mid \mathsf{processT})$. 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, we have to consider all the executions of each side. As before, the details may be found in [9].

7 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 two protocols from the literature. Our results are summarised in Figure 1.

Property	*Fujioka et al.*	*Okamoto et al.*
Vote-privacy trusted authorities	✓ none	✓ timeliness mbr.
Receipt-freeness trusted authorities	× n/a	✓ timeliness mbr.

Fig. 1: Summary of protocols and properties

We have stated the intuitive relationships between the two properties: for a fixed set of trusted authorities, receipt-freeness implies vote-privacy. This is proved in our full version [9].

Some of our reasoning about bisimulation in applied pi has been informal. In the future, we hope to develop better techniques for formalising and automating this reasoning.

Acknowledgments Thanks to Michael Clarkson and Olivier Pereira for interesting questions and discussions. Thanks also to the editors of this volume for detailed comments about the presentation, which helped us improve the readability.

References

1. Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just fast keying in the pi calculus. In *Proc. 13th European Symposium on Programming (ESOP'04)*, volume 2986 of *LNCS*, pages 340–354. Springer, 2004.

2. 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.
3. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security (CCS'97)*, pages 36–47. ACM Press, 1997.
4. 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.
5. 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.
6. Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *Proc. 20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 331–340. IEEE Comp. Soc. Press, 2005.
7. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.
8. David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology – CRYPTO'82*, pages 199–203. Plenum Press, 1983.
9. Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols. Research report, Laboratoire Spécification et Vérification, ENS Cachan, France, January 2008.
10. Marc Fischlin. *Trapdoor Commitment Schemes and Their Applications.* PhD thesis, Fachbereich Mathematik Johann Wolfgang Goethe-Universität Frankfurt am Main, 2001.
11. 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.
12. 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.
13. Tatsuaki Okamoto. An electronic voting scheme. In *Proc. IFIP World Conference on IT Tools*, pages 21–30, 1996.
14. 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.
15. 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.
16. 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.