Formalising security properties in electronic voting protocols

Stéphanie Delaune and Steve Kremer

LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France

The results presented in this report are based on joint work with Mark Ryan and Ben Smyth

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 security properties of electronic elections. We particularly focus on privacy and verifiability properties. Our definitions allow us to specify and easily change which authorities are supposed to be trustworthy and are compatible with a large class of electronic voting schemes, including those based on blind signatures, homomorphic encryptions, and mixnets.
We distinguish three notions of privacy: vote-privacy, receipt-freeness and coercion-resistance. These properties are expressed using observational equivalence and we show in accordance with intuition that coercion-resistance implies receipt-freeness which implies vote-privacy.
Concerning verifiability, we distinguish three aspects of verifiability, which we call individual verifiability, universal verifiability, and eligibility verifiability.

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. 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.

- *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.
- *Individual verifiability:* a voter can check that her own ballot is included in the election's bulletin board.
- *Universal verifiability:* anyone can check that the election outcome corresponds to the ballots published on the bulletin board.

We identify another aspect that is sometimes included in universal verifiability.

- *Eligibility verifiability:* anyone can check that each vote in the election outcome was cast by a registered voter and there is at most one vote per voter.

We explicitly distinguish eligibility verifiability as a distinct property.

Privacy properties. Vote-privacy, receipt-freeness, and coercion-resistance 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.

Verifiability properties. A major difference with traditional paper based elections is the lack of transparency. In paper elections it is often possible to observe the whole process from ballot casting to tallying, and to rely on robustness characteristics of the physical world (such as the impossibility of altering the markings on a paper ballot sealed inside a locked ballot box). By comparison, it is not possible to observe the electronic operations performed on data. Computer systems may alter voting records in a way that cannot be detected by either voters or election observers. A voting terminal's software might be infected by malware which could change the entered vote, or even execute a completely

different protocol than the one expected. The situation can be described as *voting on Satan's computer*, analogously with [6].

The concept of *election* or *end-to-end verifiability* that has emerged in the academic literature, *e.g.*, [11, 12, 8, 5, 14, 4], aims to address this problem. It should allow voters and election observers to verify, independently of the hardware and software running the election, that votes have been recorded, tallied and declared correctly.

Verifying electronic 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.

Our contributions. In this paper we report on some of our recent efforts in using the applied pi calculus to model security properties of electronic elections. 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 [7]. 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. We recall the basic ideas and concepts of the applied pi calculus in Section 2. We propose definitions for privacy-type properties and verifiability properties. Our definitions allow us to specify and easily change which authorities are supposed to be trustworthy and are compatible with a large class of electronic voting schemes, including those based on blind signatures, homomorphic encryptions, and mixnets. Section 3 is devoted to privacy-type properties. We omit the formalisation of coercion-resistance. Our goal is to give the flavour of our work without going into great details. We present our symbolic definition of election verifiability in Section 4.

This paper summarises a paper that has been recently published [9] and another one currently being submitted [13]. In this report, 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 [10] 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.

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

$A, B, C :=$	extended processes
	P
	$A \mid B$
	$\nu n.A$
	$\nu x.A$
	$\{^M/_x\}$

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.

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

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

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

This definition looks more restrictive than it is, thanks to structural equivalence. It is straightforward to prove that $\mathsf{out}(a, M).P \mid \mathsf{in}(a, x).Q \to P \mid Q\{^M/_x\}$ and *if* $M = N$ *then* P *else* $Q \to P$ 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]. We write $\Longrightarrow$ for $(\to^* \xrightarrow{\alpha} \to^*)^*$, that is, the reflexive transitive closure of the labelled reduction.

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 $A \to^* C[\mathsf{out}(a, M).P]$ 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 $A \to^* A'$, then $B \to^* B'$ 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 3).

3 Formalising privacy-type properties

Before formalising security properties, we need to define what is an electronic voting protocol in applied pi calculus. Then, 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.

3.1 Formalising voting protocols

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.

3.2 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).

3.3 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,*
- $(\mathsf{out}(u, M).P)^{ch} = \mathsf{out}(u, M).P^{ch}$,
- $(!P)^{ch} = !P^{ch}$,
- $(\textit{if } M = N \textit{ then } P \textit{ else } Q)^{ch} = \textit{if } M = N \textit{ then } P^{ch} \textit{ else } Q^{ch}$.

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 $A^{\setminus out(ch,\cdot)}$. 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.*

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

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*

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

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).*

4 Formalising verifiability properties

We present a definition of election verifiability which captures the three desirable aspects: individual, universal and eligibility verifiability. We model voting protocols in the applied pi calculus and formalise verifiability as a triple of boolean tests $\Phi^{IV}, \Phi^{UV}, \Phi^{EV}$ which are required to satisfy several conditions on all possible executions of the protocol. Φ^{IV} is intended to be checked by the individual voter who instantiates the test with her private information (*e.g.*, her vote and data derived during the execution of the protocol) and the public information available on the bulletin board. Φ^{UV} and Φ^{EV} can be checked by any external observer and only rely on public information, *i.e.*, the contents of the bulletin board.

The consideration of eligibility verifiability is particularly interesting as it provides an assurance that the election outcome corresponds to votes legitimately cast and hence provides a mechanism to detect ballot stuffing. We note that this property has been largely neglected in previous work and our earlier work [16] only provided limited scope for.

A further interesting aspect of our work is the clear identification of which parts of the voting system need to be trusted to achieve verifiability. As it is not reasonable to assume voting systems behave correctly we only model the parts of the protocol that we need to trust for the purpose of verifiability; all the remaining parts of the system will be controlled by the adversarial environment. Ideally, such a process would only model the interaction between a *voter* and the voting terminal; *that is, the messages input by the voter*. In particular, the voter should not need to trust the election hardware or software. However, achieving absolute verifiability in this context is difficult and protocols often need to trust some parts of the voting software or some administrators. Such trust assumptions are motivated by the fact that parts of a protocol can be audited, or can be executed in a distributed manner amongst several different election officials. For instance, in Helios 2.0 [5], the ballot construction can be audited using a cast-or-audit mechanism. Whether trust assumptions are reasonable depends on the context of the given election, but our work makes them explicit.

Tests Φ^{IV}, Φ^{UV} and Φ^{EV} are assumed to be verified in a trusted environment (if a test is checked by malicious software that always evaluates the test to hold, it is useless). However, the verification of these tests, unlike the election, can be repeated on different machines, using different software, provided by different stakeholders of the election. Another possibility to avoid this issue would be to have tests which are human-verifiable as discussed in [4, Chapter 5].

4.1 Formalising voting protocols for verifiability properties

To model verifiability properties we add a *record* construct to the applied pi calculus. We assume an infinite set of distinguished *record variables* $r, r_1, \ldots$. The syntax of plain processes is extended by the construct $\mathsf{rec}(r, M).P$. We write $\mathrm{rv}(A)$ and $\mathrm{rv}(M)$ for the set of record variables in a process and a term. Intuitively, the record message construct $\mathsf{rec}(r, M).P$ introduces the possibility to enter special entries in frames. We suppose that the sort system ensures that r is a variable of record sort, which may only be used as a first argument of the rec construct or in the domain of the frame. Moreover, we make the global assumption that a record variable has a unique occurrence in each process. Intuitively, this construct will be used to allow a voter to privately record some information which she may later use to verify the election.

As discussed in the introduction we want to explicitly specify the parts of the election protocol which need to be trusted. Formally the trusted parts of the voting protocol can be captured using a voting process specification.

Definition 7 (Voting process specification). *A* voting process specification *is a tuple $\langle V, A\rangle$ where V is a plain process without replication and A is a closed evaluation context such that $fv(V) = \{v\}$ and $rv(V) = \emptyset$.*

For the purposes of individual verifiability the voter may rely on some data derived during the protocol execution. We must therefore keep track of all such values, which is achieved using the record construct (Definition 8).

Definition 8. *Let* rv *be an infinite list of distinct* record variables*. We define the function* R *on a finite process* P *without replication as* $\mathsf{R}(P) = \mathsf{R}_{\mathsf{rv}}(P)$ *and, for all lists* rv*:*

$$
\begin{array}{ll}
\mathsf{R}_{rv}(0) & \widehat{=}\ 0 \\
\mathsf{R}_{rv}(P \mid Q) & \widehat{=}\ \mathsf{R}_{odd(rv)}(P) \mid \mathsf{R}_{even(rv)}(Q) \\
\mathsf{R}_{rv}(\nu n.P) & \widehat{=}\ \nu n.\mathsf{rec}(head(rv), n).\mathsf{R}_{tail(rv)}(P) \\
\mathsf{R}_{rv}(\mathsf{in}(u, x).P) & \widehat{=}\ \mathsf{in}(u, x).\mathsf{rec}(head(rv), x).\mathsf{R}_{tail(rv)}(P) \\
\mathsf{R}_{rv}(\mathsf{out}(u, M).P) & \widehat{=}\ \mathsf{out}(u, M).\mathsf{R}_{rv}(P) \\
\mathsf{R}_{rv}(\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q) & \widehat{=}\ \mathit{if}\ M = N\ \mathit{then}\ \mathsf{R}_{rv}(P)\ \mathit{else}\ \mathsf{R}_{rv}(Q)
\end{array}
$$

where the functions head and tail are the usual ones for lists, and odd (resp. even) returns the list of elements in odd (resp. even) position.

In the above definition odd and even are used as a convenient way to split an infinite list into two infinite lists. A voting process can now be constructed such that the voter V records the values constructed and input during execution.

Definition 9. *Given a voting process specification* $\langle V, A\rangle$*, integer* $n \in \mathbb{N}$*, and names* $s_1, \ldots, s_n$ *, we build the augmented voting process* $\mathsf{VP}_n^+(s_1, \ldots, s_n) = A[V_1^+ \mid \cdots \mid V_n^+]$ *where* $V_i^+ = \mathsf{R}(V)\{^{s_i}/_v\}\{^{r_i}/_r \mid r \in rv(\mathsf{R}(V))\}$*.*

Given a sequence of record variables $\tilde{r}$, we denote by $\tilde{r}_i$ the sequence of variables obtained by indexing each variable in $\tilde{r}$ with i. The process $\mathsf{VP}_n^+(s_1, \ldots, s_n)$ models the voting protocol for n voters casting votes $s_1, \ldots, s_n$, who privately record the data that may be needed for verification using record variables $\tilde{r}_i$.

4.2 Election verifiability

We formalize election verifiability using three tests Φ^{IV}, Φ^{UV}, Φ^{EV}. Formally, a test is built from conjunctions and disjunctions of *atomic tests* of the form $(M =_E N)$ where M, N are terms. Tests may contain variables and will need to hold on frames arising from arbitrary protocol executions. We now recall the purpose of each test and assume some naming conventions about variables.
Individual verifiability: The test Φ^{IV} allows a voter to identify her ballot in the bulletin board. The test has:

- a variable v referring to a voter's vote.
- a variable w referring to a voter's public credential.
- some variables $x, \bar{x}, \hat{x}, \ldots$ expected to refer to global public values pertaining to the election, *e.g.*, public keys belonging to election administrators.
- a variable y expected to refer to the voter's ballot on the bulletin board.
- some record variables $r_1, \ldots, r_k$ referring to the voter's private data.

Universal verifiability: The test Φ^{UV} allows an observer to check that the election outcome corresponds to the ballots in the bulletin board. The test has:

- a tuple of variables $\tilde{v} = (v_1, \ldots, v_n)$ referring to the declared outcome.

– some variables $x, \bar{x}, \hat{x}, \ldots$ as above.
– a tuple $\tilde{y} = (y_1, \ldots, y_n)$ expected to refer to all the voters' ballots on the bulletin board.
– some variables $z, \bar{z}, \hat{z}, \ldots$ expected to refer to outputs generated during the protocol used for the purposes of universal and eligibility verification.

Eligibility verifiability: The test Φ^{EV} allows an observer to check that each ballot in the bulletin board was cast by a unique registered voter. The test has:

– a tuple $\tilde{w} = (w_1, \ldots, w_n)$ referring to public credentials of eligible voters.
– a tuple $\tilde{y}$, variables $x, \bar{x}, \hat{x}, \ldots$ and variables $z, \bar{z}, \hat{z}, \ldots$ as above.

Individual and universal verifiability. The tests suitable for the purposes of election verifiability have to satisfy certain conditions: if the tests succeed, then the data output by the election is indeed valid (*soundness*); and there is a behaviour of the election authority which produces election data satisfying the tests (*effectiveness*). Formally these requirements are captured by the definition below. We write $\tilde{T} \simeq \tilde{T}'$ to denote that the tuples $\tilde{T}$ and $\tilde{T}'$ are a permutation of each others modulo the equational theory, that is, we have $\tilde{T} = T_1, \ldots T_n$, $\tilde{T}' = T'_1, \ldots T'_n$ and there exists a permutation χ on $\{1, \ldots, n\}$ such that for all $1 \leq i \leq n$ we have $T_i =_E T'_{\chi(i)}$.

Definition 10 (Individual and universal verifiability). *A voting specification $\langle V, A\rangle$ satisfies* individual and universal verifiability *if for all $n \in \mathbb{N}$ there exist tests Φ^{IV}, Φ^{UV} such that $fn(\Phi^{IV}) = fn(\Phi^{UV}) = rv(\Phi^{UV}) = \emptyset$, $rv(\Phi^{IV}) \subseteq rv(\mathsf{R}(V))$, and for all names $\tilde{s} = (s_1, \ldots, s_n)$ the conditions below hold. Let $\tilde{r} = rv(\Phi^{IV})$ and $\Phi^{IV}_i = \Phi^{IV}\{^{s_i}/_{v}, ^{\tilde{r}_i}/_{\tilde{r}}\}$.*

Soundness. For all contexts C and processes B such that $C[\mathsf{VP}^+_n(s_1, \ldots, s_n)] \Longrightarrow B$ and $\phi(B) \equiv \nu\tilde{n}.\sigma$, we have:

$$\forall i, j. \quad \Phi^{IV}_i\sigma \wedge \Phi^{IV}_j\sigma \Rightarrow i = j \tag{1}$$

$$\Phi^{UV}\sigma \wedge \Phi^{UV}\{^{\tilde{v}'}/_{\tilde{v}}\}\sigma \Rightarrow \tilde{v}\sigma \simeq \tilde{v}'\sigma \tag{2}$$

$$\bigwedge_{1 \leq i \leq n} \Phi^{IV}_i\{^{y_i}/_{y}\}\sigma \wedge \Phi^{UV}\sigma \;\Rightarrow\; \tilde{s} \simeq \tilde{v}\sigma \tag{3}$$

Effectiveness. There exists a context C and a process B, such that $C[\mathsf{VP}^+_n(s_1, \ldots, s_n)] \Longrightarrow B$, $\phi(B) \equiv \nu\tilde{n}.\sigma$ and

$$\bigwedge_{1 \leq i \leq n} \Phi^{IV}_i\{^{y_i}/_{y}\}\sigma \wedge \Phi^{UV}\sigma \tag{4}$$

An individual voter should verify that the test Φ^{IV} holds when instantiated with her vote s_i, the information $\tilde{r}_i$ recorded during the execution of the protocol and some bulletin board entry. Indeed, Condition (1) ensures that the test will

hold for at most one bulletin board entry. (Note that Φ_i^{IV} and Φ_j^{IV} are evaluated with the same ballot $y\sigma$ provided by $C[\_].$) The fact that her ballot is counted will be ensured by Φ^{UV} which should also be tested by the voter. An observer will instantiate the test Φ^{UV} with the bulletin board entries $\tilde{y}$ and the declared outcome $\tilde{v}$. Condition (2) ensures the observer that Φ^{UV} only holds for a single outcome. Condition (3) ensures that if a bulletin board contains the ballots of voters who voted $s_1, \ldots, s_n$ then Φ^{UV} only holds if the declared outcome is (a permutation of) these votes. Finally, Condition (4) ensures that there exists an execution where the tests hold. In particular this allows us to verify whether the protocol can satisfy the tests when executed as expected. This also avoids tests which are always false and would make Conditions (1)-(3) vacuously hold.

Eligibility verifiability. To fully capture *election verifiability*, the tests Φ^{IV} and Φ^{UV} must be supplemented by a test Φ^{EV} that checks eligibility of the voters whose votes have been counted. We suppose that the public credentials of eligible voters appear on the bulletin board. Φ^{EV} allows an observer to check that only these individuals (that is, those in possession of credentials) cast votes, and at most one vote each.

Definition 11 (Election verifiability). *A voting specification $\langle V, A\rangle$ satisfies* election verifiability *if for all $n \in \mathbb{N}$ there exist tests $\Phi^{IV}, \Phi^{UV}, \Phi^{EV}$ such that $fn(\Phi^{IV}) = fn(\Phi^{UV}) = fn(\Phi^{EV}) = rv(\Phi^{UV}) = rv(\Phi^{EV}) = \emptyset$, $rv(\Phi^{IV}) \subseteq rv(\mathsf{R}(V))$, and for all names $\tilde{s} = (s_1, \ldots, s_n)$ we have:*

1. *The tests Φ^{IV} and Φ^{UV} satisfy each of the conditions of Definition 10;*
2. *The additional conditions 5, 6, 7 and 8 below hold.*

Let $\tilde{r} = rv(\Phi^{IV})$, $\Phi_i^{IV} = \Phi^{IV}\{{}^{s_i}/_{v}, {}^{\tilde{r}_i}/_{\tilde{r}}, {}^{y_i}/_{y}\}$, $X = fv(\Phi^{EV}) \setminus \operatorname{dom} \mathsf{VP}_{\mathsf{n}}^{+}(s_1, \ldots, s_n)$

Soundness. For all contexts C and processes B such that $C[\mathsf{VP}_{\mathsf{n}}^{+}(s_1, \ldots, s_n)] \Longrightarrow B$ and $\phi(B) \equiv \nu\tilde{n}.\sigma$, we have:

$$\Phi^{EV}\sigma \wedge \Phi^{EV}\{{}^{x'}/_{x} \mid x \in X \setminus \tilde{y}\}\sigma \Rightarrow \tilde{w}\sigma \simeq \tilde{w}'\sigma \tag{5}$$

$$\bigwedge_{1 \le i \le n} \Phi_i^{IV}\sigma \wedge \Phi^{EV}\{{}^{\tilde{w}'}/_{\tilde{w}}\}\sigma \quad \Rightarrow \quad \tilde{w}\sigma \simeq \tilde{w}'\sigma \tag{6}$$

$$\Phi^{EV}\sigma \wedge \Phi^{EV}\{{}^{x'}/_{x} \mid x \in X \setminus \tilde{w}\}\sigma \Rightarrow \tilde{y}\sigma \simeq \tilde{y}'\sigma \tag{7}$$

Effectiveness. There exists a context C and a process B such that $C[\mathsf{VP}_{\mathsf{n}}^{+}(s_1, \ldots, s_n)] \Longrightarrow B$, $\phi(B) \equiv \nu\tilde{n}.\sigma$ and

$$\bigwedge_{1 \le i \le n} \Phi_i^{IV}\sigma \wedge \Phi^{UV}\sigma \wedge \Phi^{EV}\sigma \tag{8}$$

The test Φ^{EV} is instantiated by an observer with the bulletin board. Condition (5) ensures that, given a set of ballots $\tilde{y}\sigma$, provided by the environment, Φ^{EV} succeeds only for one list of voter public credentials. Condition (6) ensures that if a bulletin board contains the ballots of voters with public credentials $\tilde{w}\sigma$ then Φ^{EV} only holds on a permutation of these credentials. Condition (7) ensures that, given a set of credentials $\tilde{w}$, only one set of bulletin board entries $\tilde{y}$ are accepted by Φ^{EV} (observe that for such a strong requirement to hold we expect the voting specification's frame to contain a public key, to root trust). Finally, the effectiveness condition is similar to Condition (4) of Definition 10.

5 Conclusion

We have defined a framework for modelling cryptographic voting protocols in the applied pi calculus, and shown how to express in it privacy-type properties and verifiability properties. Within the framework, we can stipulate which parties are assumed to be trustworthy in order to obtain the desired property. In [9, 13] we investigated several protocols from the literature.

Regarding privacy-type properties we have stated the intuitive relationships between the three privacy-type properties: for a fixed set of trusted authorities, coercion-resistance implies receipt-freeness which implies vote-privacy. This is proved in our full version [9].

Regarding verifiability, we present a symbolic definition of election verifiability which allows us to precisely identify which parts of a voting system need to be trusted for verifiability. We also consider eligibility verifiability, an aspect of verifiability that is often neglected and satisfied by only a few protocols, but nonetheless an essential mechanism to detect ballot stuffing.

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. Ben Adida. *Advances in Cryptographic Voting Systems.* PhD thesis, MIT, 2006.
5. Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the Seventeenth Usenix Security Symposium*, pages 335–348. USENIX Association, 2008.
6. Ross Anderson and Roger Needham. Programming Satan's Computer. In Jan van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *LNCS*, pages 426–440. Springer, 1995.
7. 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.

8. 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.
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. 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.
11. Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-Resistant Electronic Elections. Cryptology ePrint Archive, Report 2002/165, 2002.
12. Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 61–70, New York, NY, USA, 2005. ACM. See also http://www.rsa.com/rsalabs/node.asp?id=2860.
13. Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. Technical Report CSR-10-06, University of Birmingham, 2010.
14. Participants of the Dagstuhl Conference on Frontiers of E-Voting. Dagstuhl accord. http://www.dagstuhlaccord.org/, 2007.
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. Ben Smyth, Mark D. Ryan, Steve Kremer, and Mounira Kourjieh. Towards automatic analysis of election verifiability properties. In *Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'10)*, Lecture Notes in Computer Science. Springer, 2010. To appear.