Verifying Properties of Electronic Voting Protocols *

Stéphanie Delaune
LSV, France Télécom R&D
ENS Cachan, CNRS, France
delaune@lsv.ens-cachan.fr

Steve Kremer
LSV, INRIA
ENS Cachan, CNRS, France
kremer@lsv.ens-cachan.fr

Mark Ryan
School of Computer Science
Univ. of Birmingham, UK
M.D.Ryan@cs.bham.ac.uk

Abstract

In this paper we report on some recent work to formally specify and verify electronic voting protocols. In particular, we use the formalism of the applied pi calculus: the applied pi calculus is a formal language similar to the pi calculus but with useful extensions for modelling cryptographic protocols. We model several important properties, namely fairness, eligibility, privacy, receipt-freeness and coercion-resistance. Verification of these properties is illustrated on two cases studies and has been partially automated using the Blanchet's ProVerif tool.

1 Introduction

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. But this convenience comes with the possibility of large-scale abuse and fraud. The procedures for detecting and avoiding fraud in paper-based systems, such as public counting of votes and monitored transport of ballot boxes, do not work when everything is done electronically. The electronic voting machines used in recent US elections have been fraught with problems [15].

Verification of electronic voting systems is therefore paramount, in order that voters can have the same or better confidence in electronic systems as they have in paper-based systems. Researchers have produced a plethora of formal protocols for electronic voting [8, 12, 3, 13, 9, 14, 17]. They offer the possibility of abstract analysis of the protocol against formally-stated properties.

*This work has been partly supported by the RNTL project PROUVÉ 03V360 and the ACI-SI Rossignol.

Among the properties which electronic voting protocols may satisfy are the following:

Fairness: no early results can be obtained which could influence the remaining voters.

Eligibility: only legitimate voters can vote, and only once.

Privacy: the system cannot reveal how a particular voter voted.

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 verify that her vote was really counted.

Universal verifiability: the published outcome really is the sum of all the votes.

Such security protocols are notoriously difficult to design and are known to be extremely error-prone. Formal analysis is crucial to assess their security. For instance, in other domains, security protocols which were thought to be correct for several years have, by means of formal verification techniques, been discovered to have major flaws [18, 6].

In order to perform formal analysis, the security properties, usually stated in natural language, need to be formalised. In this paper, we describe our work of two previously published papers [16, 10] in which we formalise some of these properties in a rigorous language, and verify whether they hold on particular protocols. We model them in the applied pi calculus [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, is supported by the ProVerif tool [4], and has been used to analyse a variety of security protocols in other domains [1, 11].

Outline of the paper. In Section 2, we describe two electonic voting protocols. In Section 3 we recall some notions of the applied pi calculus and in Section 4 we briefly discuss how such protocol can be modelled in this framework. Finally, in Section 5, we formalise some of the security properties given in the introduction and we discuss whether they hold on the two electronic voting schemes introduced in Section 2.

2 Electronic Voting Protocols

There are several kinds of protocols proposed for electroning voting [20]. For example, in protocols based on blind signature schemes [8, 12], the voter first obtains a token, which has been blindly signed by the administrator and which is only known to the voter herself. She later sends her vote anonymously, with this token as proof of eligibility. In schemes using homomorphic encryption [3, 13], the voter cooperates with the administrator in order to construct an encryption of her vote. The administrator then exploits homomorphic properties of the encryption algorithm to compute the encrypted tally directly from the encrypted votes. In yet other schemes [9], elaborate systems of MIX-nets are employed to guarantee voter privacy.

In this section, we describe two of these protocols: a protocol due to Fujioka *et al.* [12] which employs blind signatures and a protocol due to Lee *et al.* [17] which uses designated verifier proofs of re-encryption.

2.1 Fujioka et al., 1992

We give an informal description of the Fujioka *et al.* voting protocol [12]. 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.

In a first phase, the voter gets a signature on a commitment to his vote from the administrator. To ensure privacy, blind signatures [7] are used, *i.e.* the administrator does not learn the commitment of the vote.

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

The second phase is the actual voting phase.

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

The last phase 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 his 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.

2.2 Lee et al., 2003

We present a simplified version of the Lee *et al.* protocol [17]. One of the main advantages of this protocol is that it is *vote and go*: voters need participate in the election only once, in contrast with [12], where all voters have to finish a first phase before any of them can participate in the second phase.

We simplified the protocol in order to concentrate on the aspects that are important with respect to properties such as receipt-freeness and coercion-resistance. In particular we do not consider distributed authorities. The protocol relies on two less usual cryptographic primitives: re-encryption and designated verifier proofs (DVP) of re-encryption. We start by explaining these primitives.

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

A DVP of the re-encryption proves that the two ciphertexts contain indeed the same plaintext. However, a designated verifier proof only convinces one intended person, *e.g.*, the voter, that the re-encrypted ciphertext contains the original plaintext. In particular this proof cannot be used to convince the coercer.

Our simplified protocol can be described in three steps.

1. Firstly, the voter encrypts his vote with the collector's public key, signs the encrypted vote and sends it to an administrator on a private channel. The administrator checks whether the voter is a legitimate voter and has not voted yet. Then the administrator *re-encrypts* the given ciphertext, signs it and sends it back to the voter. The administrator also provides a DVP that the two ciphertexts contain indeed the same plaintext.

2. Then, the voter sends (via an anonymous channel) the re-encrypted vote, which has been signed by the administrator to the public board.

3. Finally, the collector checks the administrator's signature on each of the votes and, if valid, decrypts the votes and publishes the final results.

3 The applied pi calculus

The applied pi calculus [2] is a language for describing concurrent processes and their interactions. It is based on the pi calculus, but is intended to be less pure and therefore more convenient to use. Properties of processes described in the applied pi calculus can be proved by employing manual techniques [2], or by automated tools such as ProVerif [4]. As well as reachability properties which are typical of model checking tools, ProVerif can in some cases prove that processes are observationally equivalent [5]. This capability is important for privacy-type properties such as those we study here. The applied pi calculus has been used to study a variety of security protocols, such as those for private authentication [11] and for fast key establishment [1].

To describe processes in the applied pi calculus, one starts with a set of *names* (which are used to name communication channels or other constants), a set of *variables*, and a *signature* Σ which consists of the function symbols which will be used to define terms. In the case of security protocols, typical function symbols will include enc for encryption, which takes plaintext and a key and returns the corresponding cipher text, and dec for decryption, taking cipher text and a key and returning the plaintext. One can also describe the equations which hold on terms constructed from the signature, such as

$$\mathsf{dec}(\mathsf{enc}(x, k), k) = x.$$

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. Modelling bit commitment and blind signatures may also be done by choosing function symbols and defining appropriate equations.

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

$P, Q, R :=$

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

In the applied pi calculus, there is also a notion of *extended processes* A which generalises plain processes P. Details are not given here (but may be found in [2] and also in our papers [16, 10]).

Example 1 *Consider the following process P:*

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

The first component publishes the message $enc(s, k)$ by sending it on c_1. The second receives a message on channel c_1, uses the secret key k to decrypt it, and forwards the resulting plaintext on c_2.

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 $\to$. Structural equivalence, noted $\equiv$, is the smallest equivalence relation on extended processes that is closed under α-conversion on names and variables, by application of evaluation contexts—an evaluation context is an extended process with a hole instead of some extended process—and satisfying some further basic structural rules such as $A \mid 0 \equiv A$, associativity and commutativity of $|$, etc. Internal reduction $\to$ is the smallest relation on extended processes closed under structural equivalence and application of evaluation contexts such that $\mathsf{out}(a,x).P \mid \mathsf{in}(a,x).Q \to P \mid Q$ and for any ground terms M and N, whenever $M \neq_E N$, we have

$$\begin{array}{rcl} \text{if } M = M \text{ then } P \text{ else } Q & \to & P \\ \text{if } M = N \text{ then } P \text{ else } Q & \to & Q. \end{array}$$

Applied pi calculus processes evolve by executing the actions mentioned above. We write $A \to A'$ to mean that the process A evolves to A' by one step, and $A \to^* A'$ for finitely many steps.

Example 2 *Consider the process P described in example 1. We have $P \to \nu s, k.\mathsf{out}(c_2, s)$. This internal reduction expresses a communication on the channel c_1 between the two components of the process P. In the remainder of the process, y is replaced by $enc(s,k)$. Note that we have assumed that $dec(enc(x,y),y) = x$.*

Many properties of security protocols (including some of the properties we study in this paper) are formalised in terms of *observational equivalence* ($\approx$) between processes. Intuitively, processes which are *observationally equivalent* cannot be distinguished by an outside observer, no matter what sort of test he makes. This is formalised by saying that the processes are indistinguishable under any context, *i*.e., no matter in what environment they are executed.

Advantages and limitations of the applied pi calculus. An advantage of the applied pi calculus is that we can combine powerful (hand) proof techniques from the applied pi calculus with automated proofs provided by Blanchet's ProVerif tool. Moreover, the verification is not restricted to a bounded number of sessions and we do not need to explicitly define the adversary. We only give the equational theory describing the intruder. Generally, the intruder has access to any message sent on public, *i.e.* unrestricted, channels. These public channels model the network. Note that all channels are anonymous in the applied pi calculus. Unless the identity or something like the IP address is specified explicitly in the conveyed message, the origin of a message is unknown. This abstraction of a real network is very appealing, as it avoids having us to model explicitly an anonymiser service. However, we stress that a real implementation needs to treat anonymous channels with care. Another advantage of the applied pi calculus is its ability to model sophisticated cryptographic primitives (such as those used in the two example protocols) by means of the equational theory. One limitation concerns modelling non-determinism or probabilities, *e.g.* in MIX-nets [9]. In the applied pi calculus, all non-determinism is controlled by the attacker. If MIX-nets are modelled non-deterministically, this gives the attacker unreasonably strong powers.

```
let processV=
    (* his private key *)
    in(skvCh,skv).
    (* public keys of administrators *)
    in(pkaCh,pubka).
    in(pkcCh,pubkc).
    ν r.
    let e=pencrypt(v,pubkc,r) in
    out(chA,(pk(skv),e,sign(e,skv))).
    in(chA,m2).
    let (re,sa,dvpV)=m2 in
    if checkdvp(dvpV,e,re,pk(skv))=ok
    then if checksign(re,sa,pubka)=ok
    then out(ch,(re,sa))
```

Process 1. Voter process

4 Modelling Voting Protocols

Before defining the 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 1 *A voting process is a closed plain process*

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

The $V\sigma_i$ are the voter processes, the A_js the different election authorities 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 and at some moment the outcome of the vote is made public.

We also define an evaluation context S which is as VP, but has a hole instead of two of the $V\sigma_i$.

As an example, the voter process of the Lee *et al.* protocol is described in Process 1. First, each voter obtains his secret key from the PKI as well as the public keys of the election authorities. Then, a fresh random number is generated to encrypt his vote with the public key of the collector. Next, he signs the result and sends it on a private channel to the administrator. If the voter has been correctly registered, he obtains from the administrator, a re-encryption of his vote signed by the administrator together with a designated verifier proof of the fact that this re-encryption has been done correctly. If this proof is correct, then the voter sends his re-encrypted vote signed by the administrator to the collector.

5 Formalising Properties

We have analysed five major properties of electronic voting protocols: fairness, eligibility, privacy, receipt-freeness, and coercion resistance. The first two of these can be directly verified using ProVerif. The tool allows us to verify standard secrecy properties as well as resistance against guessing attacks, defined in terms of equivalences. But for privacy, receipt-freeness and coercion-resistance, we need to rely on the hand-proof techniques introduced in [2]. In the case of the last of our properties, we had to extend the applied pi calculus with a new notion which we call adaptive equivalence. We believe that the way we formalise and verify the properties increases the understanding of the properties themselves and also the way to model them.

5.1 Fairness

Fairness is the property that ensures that the protocol does not leak any votes before the opening phase. We discuss fairness using the Fujioka *et al.* protocol as an example. We model fairness as a secrecy property: it should be impossible for an attacker to learn a vote before the opening phase, *i.e.*, before the beginning of phase 2.

Standard secrecy. Checking *standard* secrecy, *i.e.* secrecy based on reachability, is the most basic property ProVerif can check. We request ProVerif to check that the variable `v` representing the vote cannot be deduced by the attacker. ProVerif directly succeeds to prove this result.

Strong secrecy. We also verified *strong secrecy* in the sense of [5]. Intuitively, strong secrecy is verified if the intruder cannot distinguish between two processes where the secret changes. ProVerif directly succeeds to prove strong secrecy.

Corrupt administrator. We have also verified standard secrecy and strong secrecy for the Fujioka *et al.* protocol in the presence of a corrupt administrator. A corrupt administrator is modeled by outputting the administrator's secret key on a public channel. Hence, the intruder can perform any actions the administrator could have done. Again, the result is positive: the administrator cannot learn the votes of a honest voter, before the committed votes are opened. Note that we do not need to model a corrupt collector, as the collector never uses his secret key, *i.e.* the collector could anyway be replaced by the attacker.

5.2 Eligibility

Eligibility is the property verifying that only legitimate voters can vote, and only once. Again, we discuss this property for the Fujioka *et al.* protocol. The way we verify the first part of this property is by giving the attacker a *challenge vote*. We modify the processes in two ways: (i) the attacker is not registered as a legitimate voter; (ii) the collector tests whether the received vote is the challenge vote and outputs the restricted name `attack` if the test succeeds. Verifying eligibility is now reduced to secrecy of the name `attack`. ProVerif succeeds in proving that `attack` cannot be deduced by the attacker.

If we register the attacker as a legitimate voter, the tool finds the trivial attack, where the intruder votes *challenge vote*. Similarly, if a corrupt administrator is modeled then the intruder can generate a signed commitment to the challenge vote and insert it.

5.3 Privacy

The privacy property aims to guarantee that the link between a given voter V and his vote v remains hidden. Anonymity and privacy properties have been successfully studied using equivalences. However, the definition of privacy in the context of voting protocols is rather subtle. While generally most security properties should hold against an arbitrary number of dishonest participants, arbitrary coalitions do not make sense here. Consider for instance the case where all but one voter are dishonest: as the results of the vote are published at the end, the dishonest voter can collude and determine the vote of the honest voter. A classical trick for modeling anonymity is to ask whether two processes, one in which V_1 votes and one in which V_2 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.

Definition 2 *A voting protocol respects* privacy *if*

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

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.

Both the Fujioka *et al.* and Lee *et al.* protocols may be shown to satisfy privacy, but this proof requires to be done by hand. Although ProVerif is capable of showing observational equivalence in some simple cases, it is not able to show it in this case.

5.4 Receipt-freeness

We also formalize receipt-freeness using observational equivalence. However, we need to model the fact that V_A is willing to provide secret information, *i.e.*, the receipt, to the coercer. We assume that the coercer is in fact the intruder who, as usual in the Dolev-Yao model, controls the public channels. To model V_A's communication with the coercer, we consider that V_A executes a voting process which has been modified: any input and any freshly generated names are forwarded to the coercer. Given a process A and a a channel name ch, we define P^{ch} to be the process like P, but which sends all of its secrets on the channel ch.

We also need a definition which removes communication from a processes. Given a process A and a channel name ch, we define the extended process $A^{\backslash out(ch,\cdot)}$ to be like the process A, but hiding the ouputs on the channel ch.

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 3 *A voting protocol is* receipt-free *if there exists a closed plain process V', satisfying the two conditions below:*

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

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 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. In accordance with intuition, we have formally shown in [10] that whenever VP is receipt-free, it also respects privacy.

The Fujioka *et al.* protocol is known not to satisfy receipt-freeness [19]. Indeed, anyone who gets to know the voter's token can easily find out her vote in the list published by the collector at the end of the election.

Lee *et al.* does satisfy that property. To show receipt-freeness for Lee *et al.*, one needs to construct a process V' which successfully can fake all secrets to a coercer. The idea is for V' to vote a, but when outputting secrets to the coercer V' prepares all outputs as if he was voting c. The crucial part is that, using his private key, he provides a fake dvp stating that the actual re-encryption of the encryption of vote a is a re-encryption of the encryption of vote c.

5.5 Coercion-resistance

Coercion-resistance is a stronger property as we give the coercer the ability to communicate *interactively* with the voter and not only receive information. In this model, the coercer can for instance prepare the messages he wants the voter to send. As for receipt-freeness, we are going to modify the voter process. However, we give the coercer the possibility to provide the messages the voter should send.

To define coercion resistance, we need to specify a stronger mode of communication with the attacker. To this end, given a process P and channel names c_1, c_2, we define the process P^{c_1,c_2} to be the process which:

- outputs all its secrets on c_1;
- accepts inputs on c_2 and uses them in its communication with the voting system.

This means that the attacker gets to see all P's secrets (via c_1), and gets to prepare messages which P should use in the voting process, and send them to P via c_2.

As a first approximation, we could try to define coercion-resistance in the following way:

$$S[V_A\{^c/_v\}^{c_1,c_2} \mid V_B\{^a/_v\}] \approx S[V' \mid V_B\{^c/_v\}].$$

This definition has an obvious problem as the coercer could oblige $V_A\{^c/_v\}^{c_1,c_2}$ to vote $c' \neq c$. In that case, the process $V_B\{^c/_v\}$ would not counterbalance the outcome to avoid a trivial way of distinguishing.

To properly define coercion-resistance, we define a new simulation relation which allows the second voting process on the right-hand side to dynamically adapt its vote to correspond to the coercer's choice. We do not give the full details here (they may be found in [10]). The intuition is that adaptive simulation holds between A and B, written $A \preceq_a B$, if no matter how the process A is closed and no matter how the environment reacts, B can be closed, such that the processes are indistinguishable.

We are now ready to define coercion-resistance.

Definition 4 *A voting protocol is* coercion-resistant *if there exists a closed extended process V' and a (strict) evaluation context C such that*

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

where x is a fresh free variable.

The intuition of this definition is that whenever the coercer requests a given vote on the left-hand side, then V_B can adapt his vote on the right-hand side and counter-balance the outcome. However, we need to avoid the case where $V' = V_A\{^c/_v\}^{c_1,c_2}$ letting V_B vote a. Therefore we require that when we apply a context C, intuitively the coercer, requesting $V_A\{^c/_v\}^{c_1,c_2}$ to vote c, V' in the same context votes a. There may be circumstances where V' may need not to cast a vote that is not a (fault attacks).

In accordance with intuition, we have formally shown that whenever a protocol is coercion-resistant, it is also receipt-free [10].

Since the Fujioka *et al.* protocol doesn't satisfy receipt-freeness, it follows that it doesn't satisfy coercion-resistance either. In the case of Lee *et al.*, the construction of V' is similar to the one for receipt-freeness. However, for coercion-resistance the coercer also provides the inputs for the messages to send out. If the coercer prepares messages corresponding to a given vote, we fake the outputs as previously and know that the non-coerced voter will counter-balance the outcome, by adaptively choosing the same vote. We have also to provide the context C and to show that the three equivalences hold.

The first one some subtleties about details of how the protocol is implemented. The attacker can observe a difference between both sides if he schedules the processes so that $V_A\{^c/_v\}^{c_1,c_2}$ is delayed on the left while $V_B\{^a/_v\}$ is allowed to proceed. If this happens, we cannot find the substitution for x required for the adaptive simulation so that $V_B\{^x/_v\}$ can proceed. The attacker will observe a difference because $V_B\{^a/_v\}$ will send his vote on the left, while $V_B\{^x/_v\}$ will not. To prevent this attack, we can make the voters report their votes along a private channel instead of a public one This means that the protocol could not be used over the internet if one wants to guarantee coercion resistance.

Another way to attack coercion resistance is to use the fault attacks. Here, the coercer provides a badly formatted input. The voter should detect this and just follow the instructions to avoid the fault attack. Invalid signatures are easy to detect. However, the case where the first encrypted vote sent to the administrator is an invalid encryption is more difficult to handle as V' cannot detect it. Here we can consider several cases depending on the details of the implementation, namely whether decryption is possible on every bitstring. If so (as in our equational theory), then the other voter could counterbalance by choosing x to be the decryption of the given garbage message. At the tallying stage this would indeed result in an invalid vote on both sides. If one considers encryption with integrity checking, which one could model in applied pi calculus by adding an explicit checking equation, then this protocol would not be coercion-resistant. This is because the non-coerced is only allowed to choose the value of its vote, but in other respects it follows the protocol and in particular encrypts the chosen vote correctly. Therefore it cannot mimick the coerced voter who sends an invalid vote. Thus the collector blocks when trying to decrypt the vote for the coerced voter, but not for V_B, resulting in an observable difference.

6 Conclusion

The paper describes our recent efforts to formally specify and verify electronic voting protocols in the applied pi calculus. Properties such as fairness and eligibility benefit from automated proofs. For more sophisticated anonymity properties, even specifying the properties is challenging, in particular receipt-freeness and coercion-resistance. In these cases we rely on hand proofs and reuse existant proof techniques from the applied pi calculus. The definition of these properties and their verification on two examples provided interesting insights.

References

[1] M. Abadi, B. Blanchet, and C. Fournet. Just fast keying in the pi calculus. In D. Schmidt, editor, *13th European Symposium on Programming (ESOP'04)*, volume 2986 of *LNCS*, pages 340–354, Barcelona, Spain, 2004. Springer.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In H. R. Nielson, editor, *Proceedings of the 28th ACM Symposium on Principles of Programming Languages*, pages 104–115, London, UK, Jan. 2001. ACM.

[3] J. G. Beneloh. *Verifiable Secret Ballot Elections*. PhD thesis, Yale University, 1987.

[4] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In S. Schneider, editor, *14th IEEE Computer Security Foundations Workshop*, pages 82–96, Cape Breton, Nova Scotia, Canada, 2001. IEEE Comp. Soc. Press.

[5] B. Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. In *IEEE Symposium on Security and Privacy*, pages 86–100, Oakland, California, May 2004.

[6] R. Chadha, S. Kremer, and A. Scedrov. Formal analysis of multi-party contract signing. In R. Focardi, editor, *17th IEEE Computer Security Foundations Workshop*, pages 266–279, Asilomar, CA, USA, June 2004. IEEE Computer Society Press.

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

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

[9] D. Chaum, P. Y. A. Ryan, and S. 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, Milan, Italy, 2005. Springer.

[10] S. Delaune, S. Kremer, and M. D. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *19th Computer Security Foundations Workshop (CSFW 2006)*. IEEE Comp. Soc. Press, 2006.

[11] C. Fournet and M. Abadi. Hiding names: Private authentication in the applied pi calculus. In *International Symposium on Software Security (ISSS'02)*, pages 317–338. Springer, 2003.

[12] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In J. Seberry and Y. Zheng, editors, *Advances in Cryptology — AUSCRYPT '92*, volume 718 of *LNCS*, pages 244–251. Springer, 1992.

[13] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In B. Preneel, editor, *Advances in Cryptography – Eurocrypt'00*, volume 1807 of *LNCS*, pages 539–556, Bruges, Belgium, may 2000. Springer.

[14] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In *Proc. of Workshop on Privacy in the Electronic Society (WPES'05)*, Alexandria, USA, 2005. ACM Press.

[15] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2004.

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

[17] B. Lee, C. Boyd, E. Dawson, K. Kim, J. Yang, and S. Yoo. Providing receipt-freeness in mixnet-based voting protocols. In *Proc. of Information Security and Cryptology (ICISC'03)*, volume 2971 of *LNCS*, pages 245–258, Seoul, Korea, 2004. Springer.

[18] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[19] T. Okamoto. An electronic voting scheme. In *IFIP World Conference on IT Tools*, pages 21–30, Canberra, Australia, 1996.

[20] Z. Rjaskova. Electronic voting schemes. Master's thesis, Comenius University, 2002. `www.tcs.hut.fi/~helger/crypto/link/protocols/voting.html`.