ISSN No. 0976-5697

Volume 3, No. 3, May-June 2012

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

Formalization &Analysis of Electronics Voting Protocols by using Applied PI Calculus

*Ms. Swati A. Khodke and 2Prof. Jayant S. Deshpande
*II Yr. M.E.[I.T.], PRMIT&R, Badnera and Prof., PRMIT&R, Badnera,
Information Technology Department
PRMIT&R, Badnera
*swatikhodke@gmail.com

Abstract— In this paper we report recent work on analysis of protocols in remote electronics voting protocols. A potentially much more secure system could be implemented, based on formal protocols that specify the messages sent to electronics voting machines.Protocols which were thought to be correct for several years have, by means of formal verification techniques, been discovered to have major flaws [1, 2]. Our aim is to use verification techniques to analyze the protocol. We model it in the applied pi calculus [3], which has the advantages of being based on well-understood concepts.

I. INTRODUCTION

Electronics voting promises the possibility of a convenient, efficient, and secure facility for regarding and tallying votes in a election. 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:

a. ***Fairness:*** no early results can be obtained which could influence the remaining voters.
b. ***Eligibility:*** only legitimate voters can vote, and only once.
c. ***Privacy:*** the fact that a particular voted in a particular way is not revealed to anyone.
d. ***Individual verifiability:*** a voter can verify that her vote was really counted.
e. ***Universal verifiability:*** the published outcome really is the sum of all the votes.
f. ***Receipt-freeness:*** a voter cannot prove that she voted in a certain way

In this paper, we study a protocol commonly known as the FOO 92 scheme [4], which works with blind signatures. By informal analysis (e.g., [5]), it has been concluded that FOO 92 satisfies the first four properties in the list above.

II. PROTOCOL FOO 92

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 [1] 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 commitmentscheme $\xi$and a random key *r*;

– *V* computes the message $e = \chi(x,b)$ using a blinding function $\chi$and a randomblinding factor *b*;

– *V* digitally signs *e* and sends his signature $\sigma V(e)$ to the administrator *A* togetherwith his identity;

– *A* verifies that *V* has the right to vote, has not voted yet and that the signature isvalid; if all these tests hold, *A* digitally signs *e* and sends his signature $\sigma A(e)$ to *V* ;

– *V* now *unblinds*$\sigma A(e)$ and obtains $y = \sigma A(x)$, i.e. a signed commitment to *V* 's vote.The second phase of the protocol is the actual voting phase.

– *V* sends *y*, *A*'s signature on the commitment to *V* 's vote, to the collector *C* usingan anonymous channel;

– *C* checks correctness of the signature *y* and, if the test succeeds, enters (□, *x*, *y*)onto a list as an *l*-th item.

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

– *C* publishes the list (*_i*, *xi*, *yi*) of commitments he obtained;

– *V* verifies that his commitment is in the list and sends _, *r* to *C* via an anonymouschannel;

– *C* opens the _-th ballot using the random *r* and publishes the vote *v*.

III. FORMAL METHODOLOGY USE

The applied pi calculus [6] is a language for describing and analysing security protocols. The applied pi calculus is a language for describing concurrent processes and their interactions. It provides intuitive process syntax for detailing the actions of the participants in a protocol, emphasizing their communication. The syntax is coupled with a formal semantics to allow reasoning about protocols. The language is based on the pi calculus with the addition of rich term algebra to enable modelling of the cryptographic operations used by security protocols. A wide variety of cryptographic primitives can be abstractly modelled by means of an equational theory. The calculus allows one to express several types of security goal, and to analyses whether the protocol meets its goal or not.

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 defineterms.In the applied pi calculus, one has (plain) processes and extended processes. Plainprocesses are built up in a similar way to processes in the pi calculus, except that messagescan contain terms (rather than just names). Extended processes can also be *activesubstitutions*: *{M/x}* is the substitution that replaces the variable *x* with the term *M*.Active substitutions generalise “let”. The process *νx*.(*{M/x}* | *P*) corresponds exactlyto “let *x* = *M* in *P*”.

Active substitutions are useful because they allow us to map an extended process *A*to its *frame* *φ*(*A*) by replacing every plain processes in *A* with 0. A frame is an extendedprocess built up from 0 and active substitutions by parallel composition and restriction.The frame *φ*(*A*) can be viewed as an approximation of *A* that accounts for the staticknowledge *A* exposes to its environment, but not *A*’s dynamic behavior.The operational semantics of processes in the applied pi calculus is defined by structuralrules defining two relations: *structural equivalence*, noted ≡, and *internal reduction*,noted→. A context *C*[·] is a process with a hole; an evaluation context is a context whose hole is not under a replication, a conditional, an input, or an output. Structural equivalence is is the smallest equivalence relation on extended processes that is closed under *α*-conversion on names and variables, by application of evaluation contexts, and satisfying some further basic structural rules such as *A* | 0 ≡*A*, associativity and commutativity of |, binding-operator-like behaviour of *ν*, and when $\Sigma \vdash M = N$ the equivalences:

νx.*{M/x}* ≡0 *{M/x}* | *A* ≡ *{M/x}* | *A{M/x}* *{M/x}* ≡ *{N/x}*

Internal reduction →is the smallest relation on extended processes closed under structural equivalence such that ¯*a_x_*.*P*| *a*(*x*).*Q* →*P* | *Q* and whenever $\Sigma \vdash M = N$,

if*M* = *M* then *P* else *Q* →*P* if *M* = *N* then *P* el/se *Q* →*Q*.

Definition 1.*Observational equivalence (≈) is the largest symmetric relation R betweenclosed extended processes with the same domain such thatA R B implies:*

1. ifA ⇓a then B ⇓a.

2. ifA →∗A_ then B →∗B_ and A_ R B_ for some B_.

3. C[*A*] *R C*[*B*] *for closing evaluation contexts C.*

In cases in which the two processes differ only by the terms they contain, if they are also observationally equivalent then ProVerif may be able to prove it directly. However,ProVerif’s ability to prove observational equivalence is incomplete, and therefore sometimes one has to resort to manual methods, whose justifications are contained in [7]. The method we use in this paper relies on two further notions: *static equivalence*(≈*s*), and *labeled bisimilarity* (≈*l*).

IV. MODELING PROTOCOL IN THE APPLIED PI CALCULUS

A. *Model:*

We use the applied pi calculus to model the FOO 92 protocol. 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 theory. Generally, the intruder has access to any message sent on a public, i.e. unrestricted, channel. 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 anonymous service.

B. *Signature and equational theory:*

The signature and equational theory are represented in Process 1. The functions and equations that handle public keys and hostnames shouldbe clear. Digital signatures are modeled as being signatures with message recovery, i.e. the signature itself contains the signed message which can be extracted using the checksignfunction. To model blind signatures we add a pair of functions blind and 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 blinded signature, when the blinding factor is known. We also consider the functionsfstand sndto extract the first, respectively second element of a pair. Note that because of the propertyunblind(sign(blind(m,r),sk),r)=sign(unblind(blind(m,r),r),sk)= sign(m,sk),

Process 1.signature and equational theory*(* Signature *)*

fun*commit /2 (* bit commitment *)*

fun*open /2 (* open bit commitment *)*

fun*sign /2 (* digital signature *)*

fun*checksign /2 (* open digital signature *)*

fun*pk /1 (* get public key from private key *)*

fun*host /1 (* get host from public key *)*

fun*getpk /1 (* get public key from host *)*

fun*b l i n d /2 (* blinding *)*

fun*unblind /2 (* undo blinding *)*

(Equational theory *)*

equation open (commit (m, r) , r) = m

equationgetpk (host (pubkey))= pubkey

equationchecksign (sign (m, sk) , pk (sk)) = m

equationunblind (b l i n d (m, r) , r) = m

equation unblind (sign (b l i n d (m, r) , sk) , r) = sign (m, sk)

C. *The environment process:*

The main process is specified in Process 2. Here we model the environment and specify how the other processes are combined. First, fresh secret keys for the voters and the administrator are generated using the restriction operator. For simplicity, all legitimate voters share the same secret key in our model (and therefore the same public key). The public keys and hostnames corresponding to the secret keys are then sent on a public channels, i.e. they are made available to the intruder. The list of legitimate voters is modeled by sending the public key of the voters to the administrator on a private communication channel. We also register the intruder as being a legitimate voter by sending his public key pk(ski) where ski is a free variable: this enables the intruder to introduce votes of his choice and models that some voters may be corrupted. Then we combine an unbounded number of each of the processes (voter, administrator and collector). An unbounded number of administrators and collectors models that these processes are servers, creating a separate instance of the server process (e.g. by "forking") for each client.

Process 2.environment process

process
*v*ska . *v*skv . (* *private keys* *)
*v*privCh . (* *channel for registering legimitate voters* *)
l e t pka=pk (ska) ***in***
l e t hosta = host (pka) ***in***
l e t pkv=pk (skv) ***in***
l e t hostv=host (pkv) ***in***
(publish host names and public keys *)*
out(ch , pka) . ***out***(ch , hosta) .
out(ch , pkv) . ***out***(ch , hostv) .
(register legimitate voters *)*
((***out*** (privCh , pkv) . ***out*** (privCh , pk (s k i))) |
(!processV) | (!processA) | (! processC))

D. *The voter process:*

The voter process given in Process 3 models the role of a voter. At the beginning two fresh random numbers are generated for blinding, respectively bit commitment of thevote. Note that the vote is not modeled as a fresh nonce. This is because generally the domain of values of the votes is known. For instance this domain could be *{yes, no}*, a finite number of candidates, etc. Hence, vulnerability to guessing attacks is an importanttopic. We will discuss this issue in more detail in section 5. The remainder of the specification follows directly the informal description given in section 2. The command in(ch,(l,=s)) means the process inputs not any pair but a pair whose second argument is s. Note that we use phase separation commands, introduced by the ProVerif tool as global synchronization commands. The process first executes all instructions of a given phase before moving to the next phase. The separation of the protocol in phases is useful when analyzing fairness and the synchronization is even crucial for privacy to hold.

Process 3.voter process

let*processV =*
vbl i n d e r . vr .
let*blindedcommitedvote=blind (commit(v ,r) ,blinder)****in***
out *(ch , (hostv , sign (blindedcommitedvote , skv)))*
in*(ch ,m2) .*
let*blindedcommitedvote0=checksign (m2, pka)* ***in***
i f *blindedcommitedvote0=blindedcommitedvote****then***
l e t *signedcommitedvote=unblind (m2, b l i n d e r)* ***in***
phase 1 .
out*(ch , signedcommitedvote) .*
in*(ch , (l ,= signedcommitedvote)) .*
phase 2 .
out*(ch , (l , r))*

E. *The administrator process:*

The administrator is modeled by the process represented in Process 4. In order to verify that a voter is a legitimate voter, the administrator first receives a public key on a private channel. Legitimate voters have been registered on this private channel in the environment process described above. The received public key has to match the voter who is trying to get a signed ballot from the administrator. If the public key indeed matches, then the administrator signs the received message which he supposes to be a blinded ballot.

Process 4.administrator process

l e t processA =
in(privCh , pubkv).(**register legimitate voters* *)
in(ch ,m1) .
l e t (hv , sig)=m1 ***in***
l e t pubkeyv=getpk (hv) ***in***
i f pubkeyv = pubkv***then***
out (ch , sign (checksign (sig , pubkeyv) , ska))

F. *The collector process:*

In Process 5 we model the collector. When the collector receives a committed vote, he associates a fresh label 'l' with this vote. Publishing the list of votes and labels is modeled by sending those values on a public channel. Then the voter can send back the random number which served as a key in the commitment scheme together with the label. The collector receives the key matching the label and opens the vote which he then publishes. Note that in this model the collector immediately publishes the vote without waiting that all voters have committed to their vote. In order to verify in section 5 that no early votes can be revealed we simply omit the last steps in the voter and collector process corresponding to the opening and publishing of the results.

Process 5.collector process

l e t processC =
phase 1 .
in(ch ,m3) .
*v*l . ***out***(ch , (l ,m3)) .
phase 2 .
in(ch , (= l , rand)) .
l e t voteV=open (checksign (m3, pka) , rand) ***in***
out(ch , voteV)

V. ANALYSIS

We have analysed three 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 [8]. 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.

A. *Fairness:*

Fairness is the property that ensures that no early results can be obtained and influence the vote. people revealing their vote when asked. 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.

a. ***Standard secrecy****:* Checking *standard* secrecy, i.e. secrecy based on reach ability, is the most basic property ProVerif can check. We request ProVerif to check that the private free variable v representing the vote cannot be deduced by the attacker. ProVerif directly succeeds to prove this result.
b. ***Strong secrecy:*** We also verified *strong secrecy* in the sense of [9]. Intuitively, strong secrecy is verified if the intruder cannot distinguish between two processes where the secret changes. For the precise definition, we refer the reader to [9]. The main difference with guessing attacks is that strong secrecy relies on observational equivalence rather than static

equivalence. ProVerif directly succeeds to prove strong secrecy.

c. ***Corrupt administrator:*** We have also verified standard secrecy, resistance against guessing attacks and strong secrecy 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.

B. Eligibility:

Eligibility is the property verifying that only legitimate voters can vote, and only once. The way we verify the first part of this property is by giving the attacker a *challengevote*. 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. The modified collector process is given in Process 6. Verifying eligibility is now reduced to secrecy of the name attack. ProVerif succeeds in proving that attack cannot be deduced by the attacker.

C. Privacy:

As ProVerif takes as input processes in the applied pi calculus, we can rely on hand proof techniques to show privacy. The processes modeling the two voters are shown in Process 8. The main process is adapted accordingly to publish public keys and host names.

Proposition 1.*The FOO 92 protocol respects privacy, i.e.* $P[vote1/v1, vote2/v2] \approx$
$P[vote2/v1, vote1/v2]$, *where P is given in Process 9.*

The proof can be sketched as follows. First note that the only difference between
$P[vote1/v1, vote2/v2]$ and $P[vote2/v1, vote1/v2]$ lies in the two voter processes. We therefore first show that
$(processV1|processV2)[vote1/v1, vote2/v2] \approx$
$(processV1|processV2)[vote2/v1, vote1/v2]$.

Process 8.two voters for checking the privacy property

(Voter1 *)*
l e t *processV1 =*
νblinder1 . νr1 .
l e t *blindedcommitedvote1=b l i n d (commit (v1 , r1) , blinder1)* ***in***
out *(ch , (hostv1 , sign (blindedcommitedvote1 , skv1))) .*
in*(ch ,m21) .*
l e t *blindedcommitedvote01=checksign (m21, pka)* ***in***
i f *blindedcommitedvote01=blindedcommitedvote1* ***then***
l e t *signedcommitedvote1=unblind (m21, blinder1)* ***in***
phase 1 .
out*(ch , signedcommitedvote1) .*
in*(ch , (l1 ,= signedcommitedvote1)) .*
phase 2 .
out*(ch , (l1 , r1))*
(Voter2 *)*
l e t *processV2 =*
νblinder2 . νr2 .
l e t *blindedcommitedvote2=b l i n d (commit (v2 , r2) , blinder2)* ***in***
out *(ch , (hostv2 , sign (blindedcommitedvote2 , skv2))) .*
in*(ch ,m22) .*
l e t *blindedcommitedvote02=checksign (m22, pka)* ***in***
i f *blindedcommitedvote02=blindedcommitedvote2* ***then***
l e t *signedcommitedvote2=unblind (m22, blinder2)* ***in***
phase 1 .
out*(ch , signedcommitedvote2) .*
in*(ch , (l2 ,= signedcommitedvote2)) .*
phase 2 .
out*(ch , (l2 , r2))*

Process 9.main process with two voters

process
νska . νskv1 . νskv2 . *(* private keys *)*
νprivCh . *(* channel for registratinglegimitate voters *)*
l e t pka=pk (ska) ***in***
l e t hosta = host (pka) ***in***
l e t pkv1=pk (skv1) ***in***
l e t hostv1=host (pkv1) ***in***
l e t pkv2=pk (skv2) ***in***
l e t hostv2=host (pkv2) ***in***
(publish host names and public keys *)*
out(ch , pka) . ***out***(ch , hosta) .
out(ch , pkv1) . ***out***(ch , hostv1) .
out(ch , pkv2) . ***out***(ch , hostv2) .
l e t v1=choice [vote1 , vote2] ***in***
l e t v2=choice [vote2 , vote1] ***in***
((***out*** (privCh , pkv1) . ***out*** (privCh , pkv2) . ***out*** (privCh , pk (s k i))) |
(processV1) | (processV2) | (! processA) | (! processC))

After the synchronization at phase 1, the remaining of the voter processes are structurally equivalent: the remaining of the first voter's process of *P*1 is equivalent to the remaining of the second voter's process of *P*2 and vice-versa. Due to this structural equivalence, *P*2 can always simulate *P*1 (and vice-versa). Moreover static equivalence will be ensured: with respect to frames $\varphi 1$ and $\varphi 2$ no other difference will be introduced and the blinding factors are never divulged.

VI. 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 existent proof techniques from the applied pi calculus.

VII. REFERENCES

[1]. RohitChadha, Steve Kremer, and Andre Scedrov.Formal analysis of multi-party contract signing. In Riccardo Focardi, editor, 17th IEEE Computer Security Foundations Workshop, pages 266–279, Asilomar, CA, USA, June 2004. IEEE Computer Society Press.

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

[3]. Ricardo Corin, JeroenDoumen, and SandroEtalle. Analysing password protocol security against off-line dictionary attacks. In 2nd International Workshop on Security Issueswith Petri Nets and other Computational Models (WISP'04), Electronic Notes in Theoretical Computer Science.Elsevier, 2004.To appear.

[4]. Atsushi Fujioka, Tatsuaki Okamoto, and KazuiOhta.A practical secret voting scheme for large scale elections. In J. Seberry and Y. Zheng, editors, Advances in Cryptology— AUSCRYPT '92, volume 718 of Lecture Notes in Computer Science, pages 244–251. Springer, 1992.

[5]. ZuzanaRjaskova. Electronic voting schemes.Master's thesis, Comenius University, 2002. www.tcs.hut.fi/ helger/crypto/link/protocols/voting.html.

[6]. Martin Abadi and Cedric 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.

[7]. Mart´ınAbadi and C´edricFournet. Mobile values, new names, and secure communication. In Hanne Riis Nielson, editor, Proceedings of the 28th ACM Symposium on Principles ofProgramming Languages, pages 104–115, London, UK, January 2001. ACM.

[8]. MartnAbadi and Cedric 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.

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