Formal Analysis of Vote Privacy Using Computationally Complete Symbolic Attacker

Gergei Bana[1](✉), Rohit Chadha[2](✉), and Ajay Kumar Eeralla[2](✉)

[1] University of Luxembourg, Luxembourg City, Luxembourg
gergei.bana@uni.lu
[2] University of Missouri, Columbia, USA
chadhar@missouri.edu, ae266@mail.missouri.edu

Abstract. We analyze the FOO electronic voting protocol in the provable security model using the technique of Computationally Complete Symbolic Attacker (CCSA). The protocol uses commitments, blind signatures and anonymous channels to achieve vote privacy. Unlike the Dolev-Yao analyses of the protocol, we assume neither perfect cryptography nor existence of perfectly anonymous channels. Our analysis reveals new attacks on vote privacy, including an attack that arises due to the inadequacy of the blindness property of blind signatures and not due to a specific implementation of anonymous channels. With additional assumptions and modifications, we were able to show that the protocol satisfies vote privacy in the sense that switching votes of two honest voters is undetectable to the attacker. Our techniques demonstrate effectiveness of the CCSA technique for both attack detection and verification.

1 Introduction

The FOO protocol was introduced by Fujioka, Okamoto, and Ohta in [1]. It was one of the first protocols for large-scale secure electronic voting. The design was supposed to achieve fairness, eligibility, vote privacy and individual verifiability. Since it's publication, it has been the subject of several attempts to formalize and verify its security properties. The focus of this paper is the formal analysis of vote privacy of FOO, namely, the property that the votes of honest voters cannot be linked to the voters. There are more modern protocols nowadays such as [2,3]. We chose FOO because of its abstract formulation, which make it convenient for symbolic analysis and its numerous phases which makes it prone to attacks.

Gergei Bana was partially supported by the ERC Consolidator Grant CIRCUS (683032) and by the National Research Fund (FNR) of Luxembourg under the Pol-Lux project VoteVerif (POLLUX-IV/1/2016). Rohit Chadha was partially supported by NSF CNS 1314338 and NSF CNS 1553548. Ajay Kumar Eeralla was partially supported by NSF CNS 1314338.

J. Lopez et al. (Eds.): ESORICS 2018, LNCS 11099, pp. 350–372, 2018.
https://doi.org/10.1007/978-3-319-98989-1_18

As far as we are aware of, the FOO protocol was the first electronic voting protocol that was formally specified and analyzed [4]. In the seminal work of Kremer and Ryan [4], both the formalization and analysis of the FOO protocol was carried out in the Dolev-Yao (DY) attacker model using the applied pi-framework [5]. The vote privacy property was the most intricate property analyzed in [4], and Kremer and Ryan were only able to prove vote privacy by hand. Subsequent development in DY verification allowed the proof to carried out automatically [6,7]. As the DY model makes the assumption of perfect cryptography, the question of whether the proof carries over in the provable security model (computational model) remained unanswered.

In this paper, we formally verify vote privacy in the provable security framework, using symbolic verification techniques. In particular, we use the computationally complete symbolic attacker technique for indistinguishability properties introduced by Bana and Comon in [8] (a.k.a. CCSA framework). As far as we know, this is the first formal analysis of FOO protocol in the provable security model. Besides proving vote privacy for FOO, our other aim is to further develop the library of axioms of the CCSA framework and to demonstrate its effectiveness in attack detection and verification.

CCSA technique was first introduced by Bana and Comon in [9] for reachability properties and then for indistinguishability properties in [8]. Since then it has been used to find new attacks to the Needham-Schroeder-Lowe protocol [10,11]; to treat algebraic operations notoriously difficult to reason about in the DY model, such as exponentiation along with it the decisional Diffie-Hellmann property and versions of the Diffie-Hellman key-exchange protocol [12]; to verify unlinkability of RFID protocols [13]; and to analyze key wrapping API's [14]. Automated tool is not yet available for the indistinguishability technique, but work is in progress. In the meantime, we continue developing the library of axioms, and verifying relatively simple protocols by hand.

FOO protocol assumes two election authorities: administrator and collector. The protocol proceeds in three phases: In the first phase, voters prepare their ballots in the form of a trapdoor commitment of their votes and obtain a blind signature on the ballots from the administrator indicating that they are eligible to vote. In the second, voters send their ballots to the collector using an anonymous channel who publishes them on a public bulletin board (BB). In the final phase, each voter verifies the presence of their ballots on the BB and then sends their trapdoor key along with the entry number of their commitment on bulletin board to the collector, again via an anonymous channel. The trapdoor keys are then also added to the bulletin board next to the commitments. After the votes finish, votes are tallied from BB. The creators of the FOO protocol did not specify how the anonymous channels are implemented. In our modeling, we model anonymous channel as a mix-net server [15] which, upon receiving a list of encrypted messages outputs their plaintexts in lexicographic order after decrypting them.

FOO protocol is designed to provide vote privacy even when the administrator and collector are corrupt and our analysis of vote privacy assumes this to be the case. As in [4,16,17], vote privacy is modeled as indistinguishability of two protocol executions: in one, honest voter A votes for candidate v_1 and honest voter B votes for candidate v_2 and in the other, honest voter A votes for candidate v_2 and honest voter B votes for candidate v_1. Observe that, as stated above, FOO protocol does not satisfy vote privacy as the attacker may choose to forward only Alice's trapdoor key in the final phase of the protocol to the mix-net server. We argue that privacy of votes can never be guaranteed for any voting protocol if the attacker allows only one participant to complete the protocol. Hence, our formalization is carefully crafted to avoid these cases.

Our analysis revealed new attacks on the FOO protocol (See Sect. 3.2). The first attack occurs because of an inadequacy of blindness property of blind signatures. Intuitively, blindness [18] means that a dishonest signer who engages in two sessions (parallel or sequential) with an honest user on messages m_0 and m_1 cannot detect which session is for m_0 and which session is for m_1 if the user successfully outputs signatures in *both* sessions. The blindness property, however, does allow the possibility that the attacker can distinguish between the sessions if the user is successful in only one session. In order to prevent this attack, we have to assume that the identities of the candidates are of equal length. This attack does not depend on the implementation of the anonymous channel. The second attack exploits the fact that encryption scheme used by the mix-net server may be length-revealing and hence the length of the encrypted messages to the mix-net server may reveal their senders. In order to prevent this, we have to assume that the signatures obtained on *equal length* messages by executing the blind signature interactive protocol with the *same* signer must be of equal length. The above two attacks lie outside the DY model and hence were not detected in previous works on formal analysis of FOO protocol. A third attack is a DY style replay attack in which messages from the Voting phase can be replayed in the Opening phase. This attack can be prevented by introducing phase numbers in the FOO protocol.

With these additional assumptions, we establish vote privacy of the FOO protocol for one session with two honest voters and one dishonest voter. The proof carries over to n dishonest voters for any fixed n. The proof of vote privacy rests on the blindness property of the blind signature, the computational hiding property of the trapdoor commitments and of IND-CCA2 assumption on the encryption used in the anonymizing mechanism. The proof of vote privacy in the DY model, in contrast, relies only on the blindness property.[1] The commitment hiding property of trapdoor commitments does not play a role in establishing vote privacy in the DY analysis.

Related Work. There have been several attempts at formal analysis of FOO protocol in the DY model (see, for example, [4,6,7]). These analyses assume perfectly anonymous channels. In the computational model, there are several attempts at formalizing vote privacy in electronic voting such as in [16,17,19,20].

[1] As evidence, we ran AKiSs [7] on a two-phase variant of the FOO protocol without commitments. The variant satisfies the vote privacy property in the DY model.

Please see [20] for a comparison amongst these definitions. All these definitions apply to single phase voting protocols. Our definition is adapted to FOO-protocol which has three phases. The only other work at formally verifying vote privacy for electronic voting that we are aware of is the mechanized proof of vote privacy for the Helios family of single-phase protocols given in [21]. The inadequacy of the blindness axiom has also been pointed out in [22] who show how any blind signature scheme can be combined with trapdoor commitments to resolve this inadequacy.

2 FOO Voting Protocol and Its Computational Modeling

We briefly recall the electronic voting protocol FOO introduced by Fujioka, Okamoto, and Ohta in [1]. We assume that the reader is familiar with the cryptographic primitives of public key encryption, trapdoor commitment schemes and digital signatures schemes. The FOO protocol also uses blind signature schemes. Informally, a blind signature scheme is an interactive protocol that allows a party $\mathcal{U}$ to obtain the signature of a party $\mathcal{S}$ on a message that is obfuscated by $\mathcal{U}$ until $\mathcal{S}$ completes the protocol and $\mathcal{U}$ publishes the signed message. We assume as in [23,24] that the interactive protocol consists of three phases: blinding by $\mathcal{U}$, (blind) signing by $\mathcal{S}$ and unblinding by $\mathcal{U}$. The primitives are described in detail in Sect. 2.2.

The FOO protocol has three roles: voters (V_i), administrator (A), and collector (C). It assumes the existence of anonymous channels. Following Kremer et al. [4], we group the (original) protocol in three phases, Authentication, Voting, and Opening, as described in Fig. 1. In the Figure, → means send, ⇢ means send via anonymous channel.

2.1 Anonymous Channel

The creators of FOO did not specify how to implement anonymous channels. We model anonymous communication using a mix-net server which, upon receiving a list of messages encrypted with its public key, checks if they are all distinct, decrypts each message in the list, outputs them in lexicographic order. Henceforth, we refer to mix-net server as the mixer. In particular, in the Voting phase (resp. Opening phase), the voter encrypts $\langle c_i, \sigma_{\mathsf{A}}(c_i)\rangle$ (resp. $\langle l_i, k_i\rangle$) with the mixer M's public key pk_M. The mixer waits until a certain time during which receives the messages from the voters, checks if all messages received until that time are distinct. If the check succeeds, the mixer decrypts the messages, shuffles the decrypted messages into lexicographic order and outputs them.

2.2 Computational Modeling of the FOO Protocol

As usual, we assume that all agents in the protocol execution, the voters, the administrator, the collector are interactive probabilistic polynomial-time Turing (PPT) machines. Furthermore, the network is controlled by an attacker, which

Authentication :

1 : V_i : computes $c_i := \xi(v_i, k_i)$, a commitment on her vote v_i with trapdoor k_i, $b_i := \chi(c_i, r_i)$, blinding the ballot c_i with blinding key r_i, and $s_i := \mathsf{sign}_i(b_i)$, a signature on b_i

2 : $V_i \rightarrow \mathsf{A}$: $\langle ID_i, b_i, s_i \rangle$

3 : A : verifies the signature s_i and eligibility (not yet applied for his signature) of V_i

4 : $\mathsf{A} \rightarrow V_i$: $\mathsf{bsign}_{\mathsf{A}}(b_i)$, (blind) signature of A on the blinded ballot b_i

Voting :

5 : V_i : checks if the received message is A's blind signature on b_i; if the check passes, V_i unblinds the signature and it is denoted by $\sigma_{\mathsf{A}}(c_i)$

6 : $V_i \dashrightarrow \mathsf{C}$: $\langle c_i, \sigma_{\mathsf{A}}(c_i) \rangle$

7 : C : verifies the signature, and adds the received message to Bulletin Board (BB) with label ℓ_i

8 : C : after a certain time, C publishes the BB

Opening :

9 : V_i : finds her ballot on the BB and records its label ℓ_i

10 : $V_i \dashrightarrow \mathsf{C}$: $\langle \ell_i, k_i \rangle$

11 : C : opens the commitments at ℓ_i using the key k_i, and appends $\langle k_i, v_i \rangle$ to $\langle c_i, \sigma_{\mathsf{A}}(c_i) \rangle$ in the BB, and after a certain time, publishes the new BB

12 : C : checks the validity of the votes, counts them and publishes the result

Fig. 1. FOO Voting Protocol

is also an interactive PPT machine. Each message goes through the attacker, except those that are "published", in which case the message is written directly on each participant's work tape synchronously. The Bulletin Board BB is simply a list of tuples. We assume that the id's ID_i are the same as the voters' public keys used for verifying their digital signatures.

We assume that encryption used in the protocol is *indistinguishable against adaptive chosen cipher text attack*, i.e., satisfies the IND-CCA2 property. The interested reader is referred to [25]. Digital signatures are used in the protocol to ensure that only eligible voters vote and do not play a part in establishing vote privacy. We now present the computational modeling of the commitment and blind digital signature schemes, and their security properties we assume for the verification.

Trapdoor Commitments. A trapdoor commitment scheme allows one party, say V, to commit to a value obfuscating the value it committed to until it is revealed by V with the help of the trapdoor. In general, a commitment scheme satisfies two properties: *binding* and *hiding*. The former property states that the party V cannot change the message once V committed to it while the latter property says that it should be computationally infeasible for any other party to retrieve the message from the commitment unless the V reveals the trapdoor.

Formally, a commitment scheme [25] $\mathcal{C}$ is a triple of algorithms (KG, $Commit$, $Open$) such that KG is a PPT algorithm with input 1^η, while $Commit$

and *Open* are PT algorithms such that there is a polynomial $p(\cdot)$ such that for each security parameter η and bit-string m of length at most $p(\eta)$, $Commit(m, k)$ computes the commitment on message m for $k \leftarrow KG(1^\eta)$, and *Open* is a deterministic algorithm such that for all $k \leftarrow KG(1^\eta)$, $Open(Commit(m, k), m, k) = true$. We assume that the commitment schemes are length regular. That is, if m_0 and m_1 are messages of equal length then the commitments of m_0 and m_1 are of equal length [26,27].

The computational hiding property is formalized as a game between a (PPT) attacker and a (honest) challenger. Initially, the attacker generates two messages m_0 and m_1 of *equal length*, which it gives to the challenger. The challenger creates commitments, com_0 and com_1 for m_0 and m_1 with different (secret) commitment keys and sets a *secret* bit b with uniform probability. The challenger then gives the ordered pair (com_b, com_{1-b}) to the attacker who wins if it correctly guesses the value of b. The commitment scheme is said to satisfy the commitment hiding property [25] if the winning probability of the attacker cannot be non-negligibly different from $\frac{1}{2}$.

Blind Digital Signatures. Blind signature schemes were introduced by Chaum [23]. They are small interactive protocols between a user and a signer. Informally, a blind signature scheme is an interactive protocol that allows a party $\mathcal{U}$ to obtain the signature of a party $\mathcal{S}$ on a message that is obfuscated by $\mathcal{U}$ until $\mathcal{S}$ completes the protocol and $\mathcal{U}$ publishes the signed message.

Formally, a blind digital signature scheme [18,24] B is a tuple $(\mathsf{Gen}, \langle \mathcal{S}, \mathcal{U} \rangle, \mathsf{verif})$, a PPT key generation algorithm Gen, an interactive algorithm $\langle \mathcal{S}, \mathcal{U} \rangle$ of the PPT singer $\mathcal{S}$ and the PPT user $\mathcal{U}$, and a PT verification algorithm verif, such that for each security parameter η: $\mathsf{Gen}(1^\eta)$ outputs a public and private key pair (pk, sk), the joint execution of signer $\mathcal{S}(pk, sk)$ and the user $\mathcal{U}(pk, m)$ on the message $m \in \{0, 1\}^\eta$ produces an output σ for the user, and the deterministic algorithm $\mathsf{verif}(m, \sigma, pk)$ outputs a bit. The algorithm $\mathcal{U}(pk, m)$ may fail to produce an output; in which case $\mathcal{U}$ is said to abort and $\mathcal{U}(pk, m)$ said to be undefined. In that case, we write $\mathcal{U}(pk, m) = \bot$. A blind signature scheme is required to be *complete*, i.e., for all parameter η the following holds: for all messages $m \in \{0, 1\}^\eta$, if $(pk, sk) \leftarrow \mathsf{Gen}(1^\eta)$ then the joint execution of $\mathcal{S}(pk, sk)$ and $\mathcal{U}(pk, m)$ must be defined and the produced output σ should be such that $\mathsf{verif}(m, \sigma, pk) = true$.

Blind signatures should also satisfy two security properties, *blindness* and *unforgeability*. Intuitively, unforgeability says a malicious user who engages in ℓ sessions with an honest user should not be able to produce $\ell + 1$ valid message-signature pairs. In the FOO protocol, unforgeability is used to make sure only eligible voters vote. It does not play a role in voting privacy. For this reason, we omit the formalization of unforgeability. The interested reader is referred to [18].

The blindness property [18] on the other hand, is useful in establishing privacy of votes. Intuitively, blindness means that a dishonest signer cannot learn the contents of the message it signed. The blindness property is formalized as a game between a (PPT) attacker acting as the signer and a (honest) challenger.

Initially, the attacker generates two messages m_0 and m_1, which it gives to the challenger. The challenger sets a *secret* bit b with uniform probability. The challenger, acting as the user, then engages in two blind signature sessions with the attacker as the signer; the first is for obtaining signature on m_b and the second is for obtaining signature on m_{1-b}. If the user successfully completes both sessions and obtains signature σ_b, σ_{1-b} then it gives the ordered pair (σ_0, σ_1) to the attacker. Otherwise, it gives the pair $(\perp, \perp)$ to the attacker. The attacker wins if it correctly guesses the value of b. Note that it is important that the challenger gives the pair (σ_0, σ_1) and not (σ_b, σ_{1-b}) (otherwise the attacker will win with probability 1). The blind signature scheme is said to satisfy the blindness property [18] if the winning probability of the attacker cannot be non-negligibly different from $\frac{1}{2}$.

We model the blind signatures using an interactive protocol between the user and the singer in three steps, blinding, blind signing, and unblinding and involving four algorithms, *blind*, *bsign*, *accept*, and *unblind* [23,24]. Before the protocol commences, the signer generates a public key and secret-key pair (pk, sk) and publishes pk. **Blinding:** User computes $b := blind(m, pk, r_b)$ for a message m, using the signer's public key pk, and a random seed r_b, and sends b to the signer. **Blind signing:** The signer computes $\rho := bsign(b, sk, r_s)$ using his own secret key sk and a random seed r_s, and sends it to the user. **Unblinding:** User checks validity of ρ by running $accept(m, pk, r_b, \rho)$. If it outputs false, the user quits the protocol without any output. If it outputs true, the user computes $\sigma = unblind(m, pk, r_b, \rho)$.

3 Vote Privacy for the FOO Protocol

The fundamental idea of vote privacy in the literature is that permutation of votes cast by honest voters cannot be detected by the attacker. There are numerous attempts to formalize this idea such as [19,20]. In the FOO protocol however, unlike the *single-phase* protocols considered therein, the Bulletin Board (BB) is built incrementally in different phases. Hence, we have to adapt the definition of privacy of votes to FOO protocol. We describe our formalization below. We consider only one session.

3.1 Formalization of Vote Privacy

We assume that the attacker controls the network and we allow the administrator and the collector to be corrupt. We require that the flipping of votes of two honest voters should result in computationally indistinguishable runs to any PPT attacker even if he controls all other voters. Under these assumptions however, there is an obvious unavoidable attack. Suppose A votes v_0 and B votes v_1, while in the flipped voting scenario, A votes v_1 and B votes v_0. If the attacker allows the ballots of both A and B to be posted in the Voting phase, but subsequently blocks B's message in the opening phase, then the two situations will be distinguishable. This is because in the first scenario only v_0 vote will appear on the BB, while in the second, only v_1. Therefore, our formalization requires that the scenario with the original votes and with the flipped votes be indistinguishable only when either both voters' votes appear on the BB or neither does.

Accordingly, we require the following two games to be computationally indistinguishable to a PPT attacker. For bit b, let Π_b be the following game between the Attacker and the Challenger simulating two honest voters of the FOO protocol and the mixer but aborting when only one of the commitment keys reach the mixer. The attacker simulates the corrupt administrator, the corrupt collector, and at the end has to guess the value of b:

1. Challenger publishes public keys of A, B, and the mixer, and a list of candidates.
2. Attacker publishes possibly dishonestly generated public keys of other voters, the administrator, and the collector. It then gives Challenger two valid votes v_0 and v_1.
3. Challenger creates the ballots of v_b for A and v_{1-b} for B.
4. Attacker calls one of A or B to authorize vote with administrator.
5. Challenger prepares the corresponding blinded ballot. It sends the corresponding identity, the blinded ballot, and the corresponding signature to the attacker.
6. Attacker creates the possibly fake authorization, sends it back to the voter, and calls for the other voter to carry out the authorization.
7. Challenger prepares the appropriate blinded ballot and sends it to the attacker.
8. Attacker creates and sends again the authorization now for the other voter, and asks either A or B to proceed with sending the ballot.
9. If the corresponding blind signature was accepted, Challenger unblinds it, and sends the signed ballot encrypted with the mixer's public key. Otherwise skips.
10. Attacker makes some computations and sends a message back (which is possibly the unchanged ciphertext sent in the previous step).
11. If the other of A or B's blind signature was accepted, Challenger now unblinds that, sends the signed ballot encrypted with the mixer's public key. Otherwise skips.
12. Attacker makes some computations and sends a message back (which is possibly the unchanged ciphertext sent in the previous step).
13. Attacker also sends the Challenger further possible encrypted signed ballots of the other, possibly corrupted voters.
14. Challenger waits for all the signed encrypted ballots to arrive. It decrypts the signed ballots with the secret key of the mixer, and puts them through the shuffle. Gives the result of the shuffle to the attacker.
15. Attacker creates the bulletin board BB and gives it to the Challenger. Note that the ballots of A and B may or may not appear on the BB. Attacker also specifies which of A or B should move to the next step first.
16. If A (resp. B) was asked to move next, A (resp. B) accepted the blind signature in phase 2 and A's (resp. B's) ballot appears on the bulletin board,

then the Challenger sends A's (resp. B's) $\langle l_i, k_i \rangle$ to the attacker encrypted with the public key of the mixer. Otherwise, the challenger skips this step.

17. Attacker makes some computations and sends a message back (which is possibly the unchanged cipher sent in the previous step).
18. If the other agent accepted the blind signature in phase 2 and its ballot appeared on the bulletin board, then the Challenger sends the other agent's $\langle l_i, k_i \rangle$ to the attacker encrypted with the public key of the mixer. Otherwise, it skips this step.
19. Attacker makes some computations and sends a message back (which is possibly the unchanged cipher sent in the previous step).
20. Attacker may also send other messages of the form $\langle l_i, k_i \rangle$ encrypted.
21. Challenger puts the decrypted messages through the shuffle. If only one of A's or B's commitment key appears, the Challenger aborts. If none of them appear or both of them appear, the Challenger gives the result of the shuffle to the attacker.
22. The attacker outputs 0 or 1.

If in the situation above the probability that the attacker outputs 1 playing game Π_0 is non-negligibly different from outputting 1 playing game Π_1, then the attacker wins the game. Our aim is to show that no PPT attacker can win the above game.

3.2 Attacks on the FOO Protocol

We describe three attacks on the vote privacy for FOO protocol, which we caught with the CCSA technique. We will comment at the beginning of Sect. 4 on how attacks are found in CCSA, and then in Sect. 5.3 on how these specific attacks were found. The first two attacks cannot be captured in the DY model as they cannot be discovered under the assumption of perfect cryptography. Moreover, the first attack is an issue of the original FOO protocol and the standard definition of blindness property; it is not a feature of our implementation of anonymous channels. On the other hand, the second attack appears because our implementation uses CCA secure encryption. The third attack is DY in nature but does not appear in previous DY analyses of the protocol as those analyses assume perfectly anonymous channels.

The first attack exploits an insufficiency in the blindness property. This insufficiency has also been pointed out in [22]. Recall that in the blindness game, a (potentially dishonest) signer engages in two sessions (parallel or sequential) with an honest user on messages m_0 and m_1. If one session successfully completes and the other aborts then the information of which session aborted is not revealed to the attacker. Thus, the blindness property does *not rule out* the possibility that the signer is able to deduce which session corresponded to m_0 and which corresponded to m_1 if one session aborted and the signer knows which session aborted. For example, in the three-step blind signature scheme described above, replace *accept* by *accept'* where $\mathit{accept}'(m, pk, r_b, \rho)$ always returns false if m is of even length and returns the value of $\mathit{accept}(m, pk, r_b, \rho)$ otherwise. The resulting blind signature scheme continues to satisfy the blindness property if the original one does. Now, if m_0 is of even length and m_1 is of odd length, then the session corresponding to message m_0 always aborts and thus the dishonest signer be able to associate the aborted session to m_0.

The first attack on FOO protocol proceeds as follows. Assume that we use the modified blind signature scheme in the FOO protocol and that the candidate choice v_0 is of even length and the choice v_1 is of odd length. Since commitments may reveal the length, it is possible that the commitment of v_0 is also even while of v_1 is odd. Assume that A chooses v_b and Bob chooses v_{1-b}. Vote privacy requires that the attacker not be able to deduce the value of b. In authentication phase, the attacker, acting as the administrator, correctly follows the protocol. The blinding still hides the commitments even though their lengths are different, but according to our modified accept algorithm, the signature on v_0 will be rejected while the one on v_1 will be accepted. Then during the voting phase, only one voter will send its ballot to the administrator. If this voter is A then b must be 1, otherwise b must be 0. Note, this attack is not a feature of our specific implementation of the anonymous channel. Note also that even though the commitments are revealed on the BB, there they appear in the same order on the two sides, so the attack works not because the commitments reveal to the attacker the vote inside, but because one accept may pass, the other may not. Observe also that the attack does not manifest if v_0 and v_1 are of equal length. Indeed, if v_0 and v_1 are of equal length then the property of commitment hiding ensures that an abort by a voter does not reveal information about its intended vote.

The second attack relies on the fact that there is no guarantee on the lengths of the (blind) signatures and that an encryption scheme used may reveal the lengths of the underlying plaintexts. If a length revealing encryption scheme (such as IND-CCA2) is used then the encrypted ballots in the voting phase are different, which allows the attacker to deduce how A and B voted. To *rule out* this attack, we have to assume that the (blind) signatures on equal length messages are of equal length.

Finally, we point out the DY style attack. The attack replays a message in the voting phase of the protocol in the opening phase. Assume that there is a third voter C in addition to A and B. The attacker remembers A's encrypted ballot to the mixer in the voting phase. In the opening phase of the protocol, the attacker forwards the encrypted commitment keys of A and B to the mixer, but replaces C's commitment key by A's encrypted ballot from the voting phase. The mixer checks that the three encryptions are different, decrypts them and outputs them after shuffling them into lexicographic order. The only ballot that appears in the output is A's. The commitment keys of A and B are also part of the output. The attacker can then deduce how A and B voted. In order to prevent this attack, we modify the FOO protocol to *also include phase* numbers inside the encryptions in the voting and opening phases of the FOO protocol. The mixer *must* check that the phase numbers correspond to the current phase in the protocol.

4 Computationally Complete Symbolic Attacker (CCSA)

The CCSA model was introduced by Bana and Comon in [9] with the aim of establishing computational guarantees for symbolic analysis by making the symbolic attacker as powerful as the computational one. We limit our attention in this paper to their indistinguishability framework given in [8]. The CCSA framework is similar to the DY framework in the sense that explicit symbolic adversarial messages are created. The DY framework specifies *all* rules that the attacker can use to create new messages from what he has seen, and the protocol agents use pattern matching to check whether messages coming from the attacker have the correct form. On the other hand, in the CCSA framework, each message from the attacker is modeled by a function f_i applied to the sequence of messages that the attacker has seen thus far. Pattern matching is replaced by applying function symbols on the term coming from the attacker. Limitations on attacker capabilities originating from computational assumptions on the primitives are specified as a set of axioms A in first-order logic based on a single indistinguishability predicate (representing computational indistinguishability of sequences of messages). Computationally, a security property of a protocol Π is formulated as the computational indistinguishability of two protocols Π_1 and Π_2 constructed (depending on the security property) from the original Π.

Verification. In the CCSA framework, the security translates to the validity of the formula obtained by applying the indistinguishability predicate on the list of terms produced by the symbolic execution of Π_1 and Π_2. Hence the security formula obtained this way must be derived as a logical consequence of the axioms A using first-order inference. If the formula is derived and the axioms in A shown to be computationally sound then the protocol Π is computationally secure according to Theorem 1 in [8].

Attack finding. If the security formula cannot be derived then the negation of the security formula is consistent with A and a symbolic attacker model is then obtained. In practice, for the security proof, a proof tree is being built, and some branch cannot be reduced to axioms. For example, often what happens is that it cannot be shown that an attacker message $f_i(\cdots)$ cannot equal a term, that is, the attacker may produce a message that passes some check when it should not. Then from this branch, an Herbrand model can be built on the terms, resulting in a symbolic attack. This symbolic attack then has to be checked to see if it corresponds to a computational attacker or if it originates from having too few axioms in A, as the set of axioms is not complete.

4.1 Syntax

Terms: Let S be a finite set of *sorts* that includes at least the sorts bool and msg. $\mathcal{X}$ is an infinite set of *variable symbols*, each coming with a sort $s \in S$. Terms are built on a set of function symbols $\mathcal{F}$ representing honest computation of primitives, a set of function symbols $\mathcal{G}$ representing attacker's computation, a set of zero-arity function symbols (names) $\mathcal{N}$ representing honest generation of randomness, and a set of variables $\mathcal{X}$. The set $\mathcal{F}$ contains the basic symbols such as $\mathbf{0}$, $\boldsymbol{true}$, $\boldsymbol{false}$, L, $\langle\_,\_\rangle$, $\mathsf{EQ}(\_,\_)$, $\mathsf{if}\,\_\,\mathsf{then}\,\_\,\mathsf{else}\,\_$, π_1 and π_2 with the typing rules as follows:

- $\mathbf{0} : \mathsf{msg}$ representing the empty message.
- $\boldsymbol{true}, \boldsymbol{false} : \mathsf{bool}$.
- $\mathsf{L}(\_) : \mathsf{msg} \rightarrow \mathsf{msg}$. $\mathsf{L}(x)$ represents the length of x in unary.
- $\langle\_,\_\rangle : \mathsf{msg} \times \mathsf{msg} \rightarrow \mathsf{msg}$ representing pairs.
- Polymorphic equality test $\mathsf{EQ}(\_,\_) : \begin{array}{l}\mathsf{msg} \times \mathsf{msg} \rightarrow \mathsf{bool}\\ \mathsf{bool} \times \mathsf{bool} \rightarrow \mathsf{bool}.\end{array}$
- Polymorphic conditional branching $\mathsf{if}\,\_\,\mathsf{then}\,\_\,\mathsf{else}\,\_ : \begin{array}{l}\mathsf{bool} \times \mathsf{msg} \times \mathsf{msg} \rightarrow \mathsf{msg}\\ \mathsf{bool} \times \mathsf{bool} \times \mathsf{bool} \rightarrow \mathsf{bool}.\end{array}$
- $\pi_1(\_), \pi_2(\_) : \mathsf{msg} \rightarrow \mathsf{msg}$. π_i represents the i-th projection of a pair

Formulas: As presented in [12], we have for every sequence of sorts $s_1, \ldots, s_n$ a *predicate symbol* that takes $2 \times n$ arguments of sort $(s_1 \times \ldots \times s_n)^2$, which we write as $t_1, \ldots, t_n \sim u_1, \ldots, u_n$. The predicate $t_1, \ldots, t_n \sim u_1, \ldots, u_n$ represents *computational indistinguishability* of the two sequences of terms $t_1, \ldots, t_n$ and $u_1, \ldots, u_n$.

The first-order formulas are built from the above atomic formulas combining the Boolean connectives $\neg$, $\wedge$, $\vee$, and $\rightarrow$, and the quantifiers $\forall$ and $\exists$. The formulas are used to represent both axioms (assumptions) and the security properties of the protocols.

Equational Theory: We use binary relation symbol $=_E$ to indicate equations the function symbols have to satisfy. Note that $=_E$ is not part of the first-order signature. The equations specified by $=_E$ will result in axioms for $\sim$. We assume that the following hold: $\forall x_1, x_2.\ \pi_i(\langle x_1, x_2\rangle) =_E x_i$, for $i = 1, 2$; $\mathsf{L}(\mathsf{L}(x)) =_E \mathsf{L}(x)$.

Abbreviations: In order to reduce the size of the terms, we shall use the following abbreviations: $\mathsf{not}(b) \overset{\text{def}}{\equiv} \mathsf{if}\, b\, \mathsf{then}\ \boldsymbol{false}\ \mathsf{else}\ \boldsymbol{true}$, $b_1\ \&\ b_2 \overset{\text{def}}{\equiv} \mathsf{if}\, b_1\, \mathsf{then}\, b_2\, \mathsf{else}\, \boldsymbol{false}$, $b_1\, \mathsf{or}\, b_2 \overset{\text{def}}{\equiv} \mathsf{if}\, b_1\, \mathsf{then}\, \boldsymbol{true}\, \mathsf{else}\, b_2$ and $x = y \overset{\text{def}}{\equiv} \mathsf{EQ}(x, y) \sim \boldsymbol{true}$. The abbreviation $\mathsf{EQL}(x_1, x_2) \overset{\text{def}}{\equiv} \mathsf{EQ}(\mathsf{L}(x_1), (\mathsf{L}(x_2)))$ modeling length equality test. Finally, the abbreviation $\langle x_1, x_2, x_3\rangle \overset{\text{def}}{\equiv} \langle x_1, \langle x_2, x_3\rangle\rangle$ encodes triples.

4.2 Semantics

As Bana and Comon defined it in [8], the logic is interpreted over a *computational model.* A computational model $\mathcal{M}^c$ is a particular first-order model in which the domain consists of probabilistic polynomial-time algorithms taking the input 1^η

together with two infinitely long random tapes (ρ_1, ρ_2), where ρ_1 is for honestly generated names, and ρ_2 for adversarial use. We use $[\![\_]\!]$ to denote the semantics of syntactic objects. Once $[\![\_]\!]$ is given on function symbols, and a valuation σ of variables of term t in the domain, the interpretation of t in the domain is defined the usual way and denoted as $[\![t]\!]^\sigma$. Let $[\![t]\!]^\sigma_{\eta,\rho} := [\![t]\!]^\sigma(1^\eta; \rho_1, \rho_2)$. In this model, function symbols in $\mathcal{F}$ representing the cryptographic primitives are interpreted as polynomial-time algorithms and they act on the outputs of the PPT algorithms in their arguments. For example, $[\![\mathsf{EQ}(t_1, t_2)]\!]^\sigma_{\eta,\rho} := 1$ if $[\![t_1]\!]^\sigma_{\eta,\rho} = [\![t_2]\!]^\sigma_{\eta,\rho}$, otherwise it is 0, and $[\![\mathsf{if}\, t_1 \,\mathsf{then}\, t_2 \,\mathsf{else}\, t_3\,]\!]^\sigma_{\eta,\rho} := [\![t_2]\!]^\sigma_{\eta,\rho}$ if $[\![t_1]\!]^\sigma_{\eta,\rho} = 1$, and $[\![\mathsf{if}\, t_1 \,\mathsf{then}\, t_2 \,\mathsf{else}\, t_3\,]\!]^\sigma_{\eta,\rho} := [\![t_3]\!]^\sigma_{\eta,\rho}$ if $[\![t_1]\!]^\sigma_{\eta,\rho} = 0$. Function symbols in $\mathcal{G}$ representing adversarial computation are interpreted as probabilistic polynomial-time algorithms and they also act on the outputs of the PPT algorithms in their arguments, but they can also use randomness from ρ_2. Each name $n \in \mathcal{N}$ is interpreted as a machine $[\![n]\!]$ which extracts a random word of length $p(\eta)$ (where p is a polynomial globally fixed by $\mathcal{M}^c$) from ρ_1. Different names draw from disjoint parts of the tape ρ_1. The predicate symbol $\sim$ is interpreted as computational indistinguishability of the outputs of the PPT algorithms on the two sides of $\sim$. Interested reader can consult either [8] or [12] for complete definition of this semantics. A formula is computationally valid if it is satisfied in all computational interpretations.

4.3 Computationally Valid Axioms

A core set of valid axioms of the CCSA have been given in [12]. Essentially, these core axioms can be divided into four sets. The first set of axioms reflect the properties of the indistinguishability predicate $\sim$, namely, that it is an equivalence relation and is preserved under projection, permutation and function application. The second set of axioms says that the abbreviation $=$ is a congruence and preserves the equational theory $=_E$. The third set of axioms reflect the properties of the function symbol $\mathsf{if\_then\_else\_}$ such as $\mathsf{if}\, \boldsymbol{true} \,\mathsf{then}\, x \,\mathsf{else}\, y\ = x$. Finally, we have a couple of axioms that reflect the *truly* random nature of names such as replacing any name n_1 by a fresh name n_2 yields indistinguishable sequence of terms. The interested reader is referred to [12] for additional details, where several small examples of usage of the axioms are also given.

5 Vote Privacy for the FOO Protocol in CCSA Framework

We assume that the FOO protocol is modified to ensure that the length candidate identities are equal and that the phase identifiers are used in Voting and Opening phases. We will drop the adjective modified in the rest of the section and show how vote privacy can be modeled in CCSA framework. In order to make things simpler, we assume that other than voters A and B, there is only a single other (possibly dishonest) voter. The proof of vote privacy carries over to n dishonest voters for any fixed n.

5.1 Function Symbols and Axioms

In order to formalize the FOO voting protocol in the CCSA framework, we shall include the following (see Table 1) function symbols in $\mathcal{F}$. Please note that each argument in these symbols has sort msg. Each key-generation algorithm and each random seed generation will be represented by a corresponding unary function symbol. When the function symbol is applied to a name n (recall that names represent truly random objects), then the resulting term shall represent a correctly generated key or seed.

Table 1. FOO function symbols with their co-domain sorts (each argument sort is msg)

Commitments	**Encryptions**	**Blind Signatures**	**Signatures**	**Other Symbols**
		$\bot$: msg		$\mathsf{ph}_2, \mathsf{ph}_3$: msg
$\mathsf{k}_\mathrm{c}(\_)$: msg		$\mathsf{k}_\mathrm{b}(\_)$: msg		A, B, M: msg
$\mathsf{com}(\_,\_)$: msg	$\mathsf{k}_\mathrm{e}(\_)$: msg	$\mathtt{r}_\mathrm{b}(\_)$: msg	$\mathsf{k}_\mathrm{s}(\_)$: msg	C_1, C_2, C_3: msg
$\mathsf{open}(\_,\_,\_)$: msg	$\mathtt{r}_\mathrm{e}(\_)$: msg	$\mathsf{bsign}(\_,\_,\_)$: msg	$\mathtt{r}_\mathrm{s}(\_)$: msg	$\mathsf{to}(\_)$: msg
	$\{\_\}_{\_}^{\_}$: msg	$\mathsf{b}(\_,\_,\_)$: msg	$\mathsf{sign}(\_,\_,\_)$: msg	$\mathsf{V}_0(\_)$: msg
Shuffling	$\mathsf{dec}(\_,\_)$: msg	$\mathsf{ub}(\_,\_,\_,\_)$: msg	$\mathsf{ver}(\_,\_,\_)$: bool	$\mathsf{V}_1(\_)$: msg
$\mathsf{shufl}(\_,\_,\_)$: msg		$\mathsf{acc}(\_,\_,\_,\_)$: bool		$\mathsf{pubkey}(\_)$: msg
		$\mathsf{bver}(\_,\_,\_)$: bool		

Commitments. We include a symbol $\mathsf{k}_\mathrm{c}(\_)$ in $\mathcal{F}$ to represent the key generation algorithm for the commitment schemes defined in Sect. 2.2. $\mathsf{com}(x, y)$ is the commitment of x using the key y whereas $\mathsf{open}(u, z)$ is the opening of the commitment u using the key z. We assume that, for all x, y, $\mathsf{open}(\mathsf{com}(x, \mathsf{k}_\mathrm{c}(y)), \mathsf{k}_\mathrm{c}(y)) =_E x$.

Encryptions. The symbol $\mathsf{k}_\mathrm{e}(\_)$ denotes the public-key secret-key pair generation algorithm of encryptions. We define $\mathsf{pk}_\mathrm{e}(x) \overset{\text{def}}{\equiv} \pi_1(\mathsf{k}_\mathrm{e}(x))$ and $\mathsf{sk}_\mathrm{e}(x) \overset{\text{def}}{\equiv} \pi_2(\mathsf{k}_\mathrm{e}(x))$, the public encryption key and secret decryption key parts of $\mathsf{k}_\mathrm{e}(x)$ respectively. In order to formalize the random seed of encryptions, we introduce a symbol $\mathtt{r}_\mathrm{e}(\_)$ (see Table 1). Encryption of x with random seed z and public key y is denoted as $\{x\}_y^z$ and decryption of x with secret key z is denoted as $\mathsf{dec}(x, z)$. We assume that for all x, y, z, $\mathsf{dec}(\{x\}_{\mathsf{pk}_\mathrm{e}(y)}^z, \mathsf{sk}_\mathrm{e}(y)) =_E x$.

Blind Signatures. We introduce a symbol $\mathsf{k}_\mathrm{b}(\_)$ to represent a public-verification-key secret-signing-key pair generation algorithm of blind signatures. Once again, $\mathsf{pk}_\mathrm{b}(x) \overset{\text{def}}{\equiv} \pi_1(\mathsf{k}_\mathrm{b}(x))$ and $\mathsf{sk}_\mathrm{b}(x) \overset{\text{def}}{\equiv} \pi_2(\mathsf{k}_\mathrm{b}(x))$ represent the public verification key and secret signing key parts of the key $\mathsf{k}_\mathrm{b}(x)$. We also introduce a symbol $\mathtt{r}_{\mathrm{b}}(\_)$ to represent blinding key generation algorithm. The interpretations of the function symbols b, bsign, bver, ub and acc are *blind*, *blindsign*, *verify*, *unblind* and *accept* as defined in Sect. 2.2. $\mathsf{b}(x, y, z)$ is the blinding of the message x using the public verification key y and the blinding key z. $\mathsf{bsign}(x, y, r)$ is the (blinded) message x blind-signed with secret blind signing key y and random seed r. $\mathsf{acc}(x, y, z, w)$ is the acceptance check of the blindsign w created for the blinded message $\mathsf{b}(x, y, z)$ and $\mathsf{ub}(x, y, z, w)$ its unblinding. $\perp$ represents undefined that is used to model blindness game. Finally, $\mathsf{bver}(x, s, y)$ is the verification of unblinded signature s on the message x with the public verification key y. The co-domains of the symbol acc and bver are bool.

Digital Signatures. In addition to blind signatures, we also need plain digital signatures. As in [12], let $\mathsf{k}_{\mathrm{s}}(\_)$ represent generation of a key pair, the public verification key and the secret signing key. $\mathsf{vk}(x) \overset{\text{def}}{\equiv} \pi_1(\mathsf{k}_{\mathrm{s}}(x))$ and $\mathsf{ssk}(x) \overset{\text{def}}{\equiv} \pi_2(\mathsf{k}_{\mathrm{s}}(x))$ represent the public verification key and secret signing key parts of the key $\mathsf{k}_{\mathrm{s}}(x)$. In order to randomize signatures, we introduce a symbol $\mathtt{r}_{\mathrm{s}}(\_)$ for random seed generation. $\mathsf{sign}(x, y, r)$ is the message x singed with secret signing key y with a random seed r and $\mathsf{ver}(z, u, y)$ is the verification of signature u on the message z with the public verification key y.

Mixer. We introduce a symbol $\mathsf{shufl}(\_, \_, \_)$ to model the shuffling functionality of mix-nets. $\mathsf{shufl}(x_1, x_2, x_3)$ is the lexicographic ordering of the messages x_1, x_2, and x_3. As we model only three voters, shufl has only three arguments.

Other Function Symbols. We introduce constants ph_2 (resp. ph_3) to represent the Voting (resp. Opening) Phase of the FOO protocol. These symbols are necessary to avoid *replay attacks* in which an attacker can learn honest vote value by forwarding one of the honest voter's message from the Voting Phase to the Opening Phase (see Subsect. 3.2). Constants A and B model the identity of two honest voters and constant M models the identity of the mixer. C_1, C_2, C_3 model the identities of the candidates. (For simplicity, we assume only three candidates.) Finally, the function symbols $\mathsf{to}(\_), \mathsf{V}_0(\_), \mathsf{V}_1(\_), \mathsf{pubkey}(\_)$ are used to model the security game and explained in Sect. 5.2.

Axioms. We present axioms (see Table 2) that formalize the security assumptions on the primitives. The axioms use the notion of *freshness* [8]: For any list of names $\vec{n}$, and a (possibly empty) list of terms $\vec{v}$, $\mathsf{fresh}(\vec{n}; \vec{v})$ is the constraint that the names in $\vec{n}$ are pairwise distinct and that none of the names in $\vec{n}$ occur in $\vec{v}$.

The axiom CompHid models the computational hiding property of the commitments (see Sect. 2.2). Intuitively, the axiom says that if the attacker is presented with commitments on equal length messages but the order of these messages is hidden, then the attacker cannot determine the order if it does not have access to commitment keys. CompHid can be shown to be computationally valid if and only if $[\![\mathsf{k}_{\mathrm{c}}(\_)]\!]$, $[\![\mathsf{com}(\_,\_)]\!]$, $[\![\mathsf{open}(\_,\_)]\!]$ satisfy the computational hiding property.

The axiom Blindness models the blindness property of the blind signatures (see Sect. 2.2). Intuitively, in the blindness game, the dishonest signer attacker is asked to sign two blinded messages (could be of unequal length), but the order of these messages is hidden. Thus, the blinded messages on the two sides of $\sim$ are in reversed order. Moreover, the signer is also allowed to see the unblinded signatures if the user does not abort for either signature. However, in order to hide the order, the unblinded signatures are presented in the same order. Blindness can be shown to be computationally valid if and only if blind signature scheme $([\![\mathsf{k}_{\mathrm{b}}]\!].[\![\mathtt{r}_{\mathrm{b}}]\!], [\![\mathsf{b}]\!], [\![\mathsf{ub}]\!], [\![\mathsf{bsign}]\!], [\![\mathsf{acc}]\!], [\![\mathsf{bver}]\!])$ satisfies the blindness property. The axiom Shuffle models the shuffling capability of the mixer. We also require further assumptions on the primitives. In particular, CommEQL models the assumption that the length of commitments on equal length messages are equal. CommKeyEQL models the assumption that the length of honestly generated commitment keys are equal. UBNotUndefined models the assumption that if a user successfully completes the blind signature protocol then then the output is different from $\perp$. UBEQL models the assumption that blind signatures on equal length message are of equal length. PairEQL models the length regularity of pairing. CandEQL (resp. PhaseEQL) models the assumptions that candidate identities (resp. phase identities) are of equal length. AgentDist (resp. PhaseDist) models the assumption that identities of honest voters (resp. phase identifiers) are distinct.

Finally, we recall the axiom representing IND-CCA2 property of encryptions from [12]. Let $\vec{t}[x]$ be a list of terms with a single variable x. For a closed term v, let $\vec{t}[v]$ denote the term that we receive from $\vec{t}[x]$ by replacing all occurrences of x by v. Let u, u', u'' be closed terms. Consider the formula

$$\vec{t}[\mathsf{if}\ \mathsf{EQL}(u,u')\ \mathsf{then}\ \{u\}_{\mathsf{pk}(n_1)}^{\mathtt{r}_{\mathrm{e}}(n_2)}\ \mathsf{else}\ u''\,] \sim \vec{t}[\mathsf{if}\ \mathsf{EQL}(u,u')\ \mathsf{then}\ \{u'\}_{\mathsf{pk}(n_1)}^{\mathtt{r}_{\mathrm{e}}(n_3)}\ \mathsf{else}\ u''\,]$$

in which $n_1 \in \mathcal{N}$ occurs only as $\mathsf{k}(n_1)$, $\mathsf{sk}(n_1)$ only occurs in decryption position (that is, as in $\mathsf{dec}(\_,\mathsf{sk}(n_1))$), and n_2, n_3 do not occur anywhere else. We call the above formula $\textsc{Enc}_{\mathrm{CCA2}}$ if for any $t'[x]$ term with x explicitly occurring in $t'[x]$, $\mathsf{dec}(t'[x],\mathsf{sk}(n_1))$ occurs only as, $\mathsf{if}\ \mathsf{EQ}(t'[x],x)\ \mathsf{then}\ t''[x]\ \mathsf{else}\ \mathsf{dec}(t'[x],\mathsf{sk}(n_1))$, where $t''[x]$ is not of the form $\mathsf{dec}(t'''[x],\mathsf{sk}(n_1))$. The intuition is that since in the IND-CCA2 game, after encryption, the decryption oracle decrypts only those messages that are different from the encryption, we have to make sure that the decrypted message, $t'[x]$ is different from the encryption, for which x stands. The reader is referred to [12] for additional details.

5.2 Modeling the Vote Privacy Security Game

As defined in [8], the BC technique treats protocols as abstract transition systems. We do not recall the formal definitions, we just apply them to the vote privacy games Π_b for $b = 0,1$ between the Challenger and the Attacker as defined in Sect. 3.1. The transition system produces the terms of the execution,

Table 2. Axioms that formalize the computational assumptions of the primitives

Commitments

Let t_1 and t_2 be two ground terms. Let n_1 and n_2 be names such that $\mathsf{fresh}(n_1, n_2; t_1, t_2)$ holds.

CommEQL: $\text{if } \mathsf{EQL}(t_1, t_2) \text{ then } \mathsf{EQL}(\mathsf{com}(t_1, \mathsf{k_c}(n_1)), \mathsf{com}(t_2, \mathsf{k_c}(n_2))) \text{ else } \textbf{\textit{true}} = \textbf{\textit{true}}$

CommKeyEQL: $\mathsf{EQL}(\mathsf{k_c}(n_1), \mathsf{k_c}(n_2)) = \textbf{\textit{true}}$

CompHid: Let t, t_1, and t_2 be three ground terms and let $\vec{z}$ be a list of ground terms and n_1 and n_2 be names such that $\mathsf{fresh}(n_1, n_2; \vec{z}, t, t_1, t_2)$ holds.

$$\vec{z}, \begin{array}{l}\text{if } \mathsf{EQL}(t_1, t_2) \\ \text{then } \langle \mathsf{com}(t_1, \mathsf{k_c}(n_1)), \mathsf{com}(t_2, \mathsf{k_c}(n_2)) \rangle \\ \text{else } t \end{array} \sim \vec{z}, \begin{array}{l}\text{if } \mathsf{EQL}(t_1, t_2) \\ \text{then } \langle \mathsf{com}(t_2, \mathsf{k_c}(n_1)), \mathsf{com}(t_1, \mathsf{k_c}(n_2)) \rangle \\ \text{else } t \end{array}$$

Blind Digital Signatures

Blindness: Let m_0, m_1, t be ground terms and let $\vec{z}$ be a list of ground terms. Let t_0 and t_1 be terms containing two variables x and y. Let n_0, and n_1 be names such that $\mathsf{fresh}(n_0, n_1; \vec{z}, t, m_0, m_1, t_0, t_1)$ holds.

$$\vec{z}, \mathsf{b}(m_0, t, \mathtt{r_b}(n_0)), \mathsf{b}(m_1, t, \mathtt{r_b}(n_1)), u^0 \sim \vec{z}, \mathsf{b}(m_1, t, \mathtt{r_b}(n_0)), \mathsf{b}(m_0, t, \mathtt{r_b}(n_1)), u^1$$

where $u^j = \text{if } \mathsf{acc}_0^j \,\&\, \mathsf{acc}_1^j \text{ then } p^j \text{ else } \langle \bot, \bot \rangle$, $\mathsf{acc}_i^0 = \mathsf{acc}(m_i, t, \mathtt{r_b}(n_i), t_i^0)$

$\mathsf{acc}_i^1 = \mathsf{acc}(m_{1-i}, t, \mathtt{r_b}(n_i), t_i^1), p^{\mathrm{j}} = \left\langle \mathsf{ub}(m_0, t, \mathtt{r_b}(n_j), t_j^{\mathrm{j}}), \mathsf{ub}(m_1, t, \mathtt{r_b}(n_{1-j}), t_{1-j}^{\mathrm{j}}) \right\rangle$

$t_i^{\mathrm{j}} = t_i[x \leftarrow \mathsf{b}(m_j, t, \mathtt{r_b}(n_0)), y \leftarrow \mathsf{b}(m_{1-j}, t, \mathtt{r_b}(n_1))]$, $i,\ j \in \{0, 1\}$

For $i = 0, 1$, let t, t_1^i, and t_2^i be ground terms. Let n^i be a name with $\mathsf{fresh}(n^i; t, t_1^1, t_1^2, t_2^{1-i})$, such that it occurs in t_2^i, only as $\mathsf{b}(t_1^i, t, \mathtt{r_b}(n^i))$:

UBNotUndefined:

$\text{if } \mathsf{acc}(t_1^0, t, \mathtt{r_b}(n^0), t_2^0) \text{ then } \mathsf{EQ}(\mathsf{ub}(t_1^0, t, \mathtt{r_b}(n^0), t_2^0), \bot) \text{ else } \textbf{\textit{false}} = \textbf{\textit{false}}$

UBEQL:

$$\begin{array}{l}\text{if } \mathsf{EQL}(t_1^0, t_1^1) \,\&\, \mathsf{EQL}(t_2^0, t_2^1) \,\&\, \mathsf{acc}(t_1^0, t, \mathtt{r_b}(n^0), t_2^0) \,\&\, \mathsf{acc}(t_1^1, t, \mathtt{r_b}(n^1), t_2^1) \\ \text{then } \mathsf{EQL}(\mathsf{ub}(t_1^0, t, \mathtt{r_b}(n^0), t_2^0), \mathsf{ub}(t_1^1, t, \mathtt{r_b}(n^1), t_2^1)) \\ \text{else } \textbf{\textit{true}} \end{array} = \textbf{\textit{true}}$$

Mix-net Server

Shuffle: For any permutation p of $\{1, 2, 3\}$, $\mathsf{shufl}(x_1, x_2, x_3) = \mathsf{shufl}(x_{p(1)}, x_{p(2)}, x_{p(3)})$

Further length regularity

PairEQL: $\text{if } \mathsf{EQL}(x_1, y_1) \,\&\, \mathsf{EQL}(x_2, y_2) \text{ then } \mathsf{EQL}(\langle x_1, x_2 \rangle, \langle y_1, y_2 \rangle) \text{ else } \textbf{\textit{true}} = \textbf{\textit{true}}$

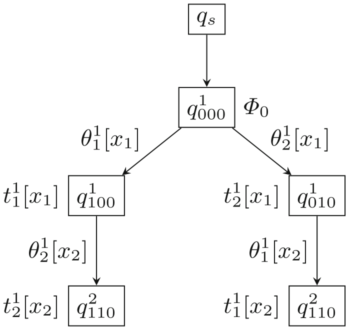
CandEQL: $\mathsf{EQL}(C_i, C_j) = \textbf{\textit{true}}$ for $i, j \in \{1, 2, 3\}$

PhaseEQL: $\mathsf{EQL}(\mathsf{ph}_2, \mathsf{ph}_3) = \textbf{\textit{true}}$

Distinctness of constants

AgentDist: $\mathsf{EQ}(A, B) = \mathsf{EQ}(A, M) = \mathsf{EQ}(B, M) = \textbf{\textit{false}}$

PhaseDist: $\mathsf{EQ}(\mathsf{ph}_2, \mathsf{ph}_3) = \textbf{\textit{false}}$



$\theta^1_1[x_i] \equiv (\mathsf{EQ}(\mathsf{to}(x_i), A)\ \&\ \mathsf{vcheck}[v_b]$
$t^1_1[x_i] \equiv \langle \mathsf{vk}(n_A), b_1, \mathsf{sign}(b_1, \mathsf{ssk}(n_A), \mathtt{r}_{\mathrm{s}}(n^1_{\mathrm{s}}))\rangle$
$\theta^1_2[x_i] \equiv (\mathsf{EQ}(\mathsf{to}(x_i), B)\ \&\ \mathsf{vcheck}[v_{1-b}]$
$t^1_2[x_i] \equiv \langle \mathsf{vk}(n_B), b_2, \mathsf{sign}(b_2, \mathsf{ssk}(n_B), \mathtt{r}_{\mathrm{s}}(n^2_{\mathrm{s}}))\rangle$
where i is in $\{1, 2\}$
$\Phi_0 \equiv A, B, M, \mathsf{vk}(n_A), \mathsf{vk}(n_B), \mathsf{pk}_{\mathrm{e}}(n_M), C_1, C_2, C_3$
$b_1 \equiv \mathsf{b}(c_1, \mathsf{PK}_{\mathrm{AD}}, \mathtt{r}_{\mathrm{b}}(n^1_{\mathrm{b}})); c_1 \equiv \mathsf{com}(v_b, \mathsf{k}_{\mathrm{c}}(n^1_{\mathrm{c}}))$
$b_2 \equiv \mathsf{b}(c_2, \mathsf{PK}_{\mathrm{AD}}, \mathtt{r}_{\mathrm{b}}(n^2_{\mathrm{b}})); c_2 \equiv \mathsf{com}(v_{1-b}, \mathsf{k}_{\mathrm{c}}(n^2_{\mathrm{c}}))$
$\mathsf{vcheck}[v] \equiv \mathsf{EQ}(v, C_1)$ or $\mathsf{EQ}(v, C_2)$ or $\mathsf{EQ}(v, C_3)$
$v_b \equiv \mathsf{V}_b(x_1); v_{1-b} \equiv \mathsf{V}_{1-b}(x_1)$
$\mathsf{PK}_{\mathrm{AD}} \equiv \mathsf{pubkey}(x_1)$

Fig. 2. Authentication phase of the security game Π_b

on which the indistinguishability predicate is applied. The verification task is to prove that the axioms together with first-order inference rules imply that the execution terms produced by the transition system for $b = 0$ and for $b = 1$ are indistinguishable.

Recall that in addition to honest voters A and B, there is an additional voter only a single other voter. We assume the mixer M is honest, and the administrator as well as the collector are corrupt. Recall that we assume that the names A, B, and M are distinct message constants. We also use distinct names n_A, n_B, n_M for (honest) signing key generations for the voters A, B, and encryption key generation for the Mixer M.

The function symbol to extracts from an incoming message the agent name. This allows the attacker to specify which of A, B, and M should receive the message sent by the attacker. The function symbols V_b, $b \in \{0, 1\}$ extract from an incoming attacker message the candidate choices for the honest voters A and B. The function symbol pubkey extracts from an incoming attacker message the (dishonest) public-key of the attacker.

We present the transition diagram that represents the Authentication Phase of the security game Π_b in Fig. 2. It illustrates all possible moves of the challenger in the authentication phase. At the start of the game, the knowledge of the attacker is initialized and is represented by ϕ_0. The left branch of q^1_{000} represents the situation where A moves first for authentication followed by B while the right branch simulates the situation that B moves first followed by A. $\theta^p_i[x_j]$s are the bool conditions that the challenger checks upon receiving the message x_j from the attacker before taking the respective transitions. Here p represents the Phase number, i represents the agent that has to move (we use 1 for A, 2 for B and 3 for M) and j represents the message number. Similarly the terms $t^p_i[x_j]$s represent the outputs of the challenger to the attacker which increases attacker's knowledge. When the checks in a transition fail, the transition moves to state q_{exit}, which we omit in transition diagram for clarity. The transition systems for other phases can be similarly defined (see full version of the paper [28]).

We obtain the execution terms from the transition system in the following way. For a given b bit of the Challenger, Φ_i^b lists of terms representing what the attacker has seen up to step i are created according to the rounds of the protocol. Notice that the initialization frame Φ_0 is defined in Fig. 2 and it is independent of the bit b. For the first step, the attacker's computation f_1 is applied to Φ_0, and $f_1(\Phi_0)$ is sent to the Challenger, who then carries out the checks θ_1^1 and θ_2^1, and sends back

$$\phi_1^b := \mathsf{if}\ \theta_1^1[f_1(\Phi_0)]\ \mathsf{then}\ t_1^1[f_1(\Phi_0)]\ \mathsf{else}\ (\mathsf{if}\ \theta_2^1[f_1(\Phi_0)]\ \mathsf{then}\ t_2^1[f_1(\Phi_0)]\ \mathsf{else}\ \mathbf{0}\ ),$$

and we set $\Phi_1^b \overset{\mathrm{def}}{\equiv} \Phi_0, \phi_1^b$. Similarly, in the second step, we have

$$\begin{aligned} &\mathsf{if}\ \theta_1^1[f_1(\Phi_0)] \\ \phi_2^b := &\mathsf{then}\ (\mathsf{if}\ \theta_2^1[f_2(\Phi_1^b)]\ \mathsf{then}\ t_2^1[f_2(\Phi_1^b)]\ \mathsf{else}\ \mathbf{0}\ ) \\ &\mathsf{else}\ (\mathsf{if}\ \theta_2^1[f_1(\Phi_0)]\ \mathsf{then}\ (\mathsf{if}\ \theta_1^1[f_2(\Phi_1^b)]\ \mathsf{then}\ t_1^1[f_2(\Phi_1^b)]\ \mathsf{else}\ \mathbf{0}\ )\ \mathsf{else}\ \mathbf{0}\ ), \end{aligned}$$

and $\Phi_2^b \overset{\mathrm{def}}{\equiv} \Phi_1^b, \phi_2^b$, which is the end of the authentication phase. The other phases are done similarly continuing from Φ_3^b.

Let Φ_m^b be the last frame. In order to establish vote-privacy for the FOO protocol we have to show that the axioms and first-order inference rules imply that $\Phi_m^0 \sim \Phi_m^1$. Then from the soundness theorem of [8], it follows that there is no successful PPT attacker that breaks the security game. We have the following.

Theorem 1. *The modified FOO protocol respects vote privacy for one session with two honest voters and one dishonest voter.*

Proof Sketch: We recall the core axioms FuncApp and IfBranch from [12]:

$$\textsc{FuncApp}:\ \ \vec{f} \in \mathcal{F} \cup \mathcal{G},\ \vec{x} \sim \vec{y} \longrightarrow \vec{x}, \vec{f}(\vec{x}) \sim \vec{y}, \vec{f}(\vec{y}).$$

$$\begin{aligned} \textsc{IfBranch}:\ &\vec{z}, b, x_1, ..., x_n \sim \vec{z}', b', x_1', ..., x_n' \ \wedge\ \vec{z}, b, y_1, ..., y_n \sim \vec{z}', b', y_1', ..., y_n' \\ &\longrightarrow \\ &\quad \vec{z}, b, \begin{matrix}\mathsf{if}\ b & \mathsf{then}\ x_1 \\ & \mathsf{else}\ y_1\end{matrix}, ..., \begin{matrix}\mathsf{if}\ b & \mathsf{then}\ x_n \\ & \mathsf{else}\ y_n\end{matrix} \sim \vec{z}', b', \begin{matrix}\mathsf{if}\ b' & \mathsf{then}\ x_1' \\ & \mathsf{else}\ y_1'\end{matrix}, ..., \begin{matrix}\mathsf{if}\ b' & \mathsf{then}\ x_n' \\ & \mathsf{else}\ y_n'\end{matrix}. \end{aligned}$$

Instead of showing the full security proof, we show how the combination of CompHid, CommEQL, UBNotUndefined and Blindness allows us to fix the inadequacy of the blindness property. The key idea is that as the length of candidate identities are equal, the commitments hide the underlying vote. Thus, the probability that the attacker can cause the blind signature for candidate v_b is accepted but rejected for v_{1-b} is negligibly small. This is formalized in the following Proposition.

Proposition 1. Let v_0, v_1, t be ground terms, let $\vec{z}$ be a list of ground terms. Assume $\mathsf{EQL}(v_0, v_1) = \boldsymbol{true}$. Let t_0 and t_1 be terms containing two variables x and y. Let $n_{\mathrm{b}0}$, $n_{\mathrm{b}1}$, $n_{\mathrm{c}0}$, $n_{\mathrm{c}1}$ be names such that $\mathsf{fresh}(n_{\mathrm{b}0}, n_{\mathrm{b}1}, n_{\mathrm{c}0}, n_{\mathrm{c}1}; \vec{z}, t, v_0, v_1, t_0, t_1)$ holds. Suppose that the blind signatures satisfy Blindness and UBNotUndefined, and commitments satisfy CompHid and CommEQL properties. Then

$$\vec{z}, \mathsf{b}(c_0^0[v_0], t, \mathtt{r}_{\mathrm{b}}(n_{\mathrm{b}0})), \mathsf{b}(c_1^0[v_1], t, \mathtt{r}_{\mathrm{b}}(n_{\mathrm{b}1})), t^0 \sim \vec{z}, \mathsf{b}(c_0^1[v_1], t, \mathtt{r}_{\mathrm{b}}(n_{\mathrm{b}0})), \mathsf{b}(c_1^1[v_0], t, \mathtt{r}_{\mathrm{b}}(n_{\mathrm{b}1})), t^1 \tag{1}$$

$$
\begin{aligned}
\text{where } t^0 \equiv{}& \mathsf{if}\ \mathsf{acc}_0^0\ \&\ \mathsf{acc}_1^0\ \mathsf{then}\ \big\langle\big\langle \mathsf{ub}_0^0[v_0], c_0^0[v_0], \mathsf{k}_\mathrm{c}(n_{\mathrm{c}0})\big\rangle, \big\langle \mathsf{ub}_1^0[v_1], c_1^0[v_1], \mathsf{k}_\mathrm{c}(n_{\mathrm{c}1})\big\rangle\big\rangle \\
&\quad \mathsf{else\,if}\ \mathsf{acc}_0^0\ \mathsf{then}\ \big\langle\big\langle \mathsf{ub}_0^0[v_0], c_0^0[v_0], \mathsf{k}_\mathrm{c}(n_{\mathrm{c}0})\big\rangle, \bot\big\rangle \\
&\quad \mathsf{else\ if}\ \mathsf{acc}_1^0\ \mathsf{then}\ \big\langle \bot, \big\langle \mathsf{ub}_1^0[v_1], c_1^0[v_1], \mathsf{k}_\mathrm{c}(n_{\mathrm{c}1})\big\rangle\big\rangle \\
&\quad \mathsf{else}\ \langle \bot, \bot\rangle \\
t^1 \equiv{}& \mathsf{if}\ \mathsf{acc}_0^1\ \&\ \mathsf{acc}_1^1\ \mathsf{then}\ \big\langle\big\langle \mathsf{ub}_1^1[v_0], c_1^1[v_0], \mathsf{k}_\mathrm{c}(n_{\mathrm{c}1})\big\rangle, \big\langle \mathsf{ub}_0^1[v_1], c_0^1[v_1], \mathsf{k}_\mathrm{c}(n_{\mathrm{c}0})\big\rangle\big\rangle \\
&\quad \mathsf{else\,if}\ \mathsf{acc}_1^1\ \mathsf{then}\ \big\langle\big\langle \mathsf{ub}_1^1[v_0], c_1^1[v_0], \mathsf{k}_\mathrm{c}(n_{\mathrm{c}1})\big\rangle, \bot\big\rangle \\
&\quad \mathsf{else\ if}\ \mathsf{acc}_0^1\ \mathsf{then}\ \big\langle \bot, \big\langle \mathsf{ub}_0^1[v_1], c_0^1[v_1], \mathsf{k}_\mathrm{c}(n_{\mathrm{c}0})\big\rangle\big\rangle \\
&\quad \mathsf{else}\ \langle \bot, \bot\rangle
\end{aligned}
$$

$$
\begin{aligned}
&t_i^{\mathrm{j}} = t_i[x \leftarrow \mathsf{b}(c_0^j[v_j], t, \mathtt{r}_\mathrm{b}(_{\mathrm{b}0}))), y \leftarrow \mathsf{b}(c_1^j[v_{1-j}], t, \mathtt{r}_\mathrm{b}(_{\mathrm{b}1})))],\ i,\ j \in \{0,1\}, \\
&c_i^0[v_i] \equiv \mathsf{com}(v_i, \mathsf{k}_\mathrm{c}(n_{\mathrm{ci}})) \qquad\qquad c_i^1[v_{1-i}] \equiv \mathsf{com}(v_{1-i}, \mathsf{k}_\mathrm{c}(n_{\mathrm{ci}})) \\
&\mathsf{acc}_i^0 \equiv \mathsf{acc}(c_i^0[v_i], t, \mathtt{r}_\mathrm{b}(n_{\mathrm{bi}}), t_i^0) \quad \mathsf{acc}_i^1 \equiv \mathsf{acc}(c_i^1[v_{1-i}], t, \mathtt{r}_\mathrm{b}(n_{\mathrm{bi}}), t_i^1) \\
&\mathsf{ub}_i^0[v_i] \equiv \mathsf{ub}(c_i^0[v_i], t, \mathtt{r}_\mathrm{b}(n_{\mathrm{bi}}), t_i^0)\ \mathsf{ub}_i^1[v_{1-i}] \equiv \mathsf{ub}(c_i^1[v_{1-i}], t, \mathtt{r}_\mathrm{b}(n_{\mathrm{bi}}), t_i^1)
\end{aligned}
$$

The proof idea of the Proposition is the following. Let us further introduce the notation
$u_i^j[v_{j\oplus i}] \equiv \Big\langle \mathsf{ub}_i^j[v_{j\oplus i}], c_i^j[v_{j\oplus i}], \mathsf{k}_\mathrm{c}(n_{\mathrm{ci}})\Big\rangle$ and $b_i^j[v_{j\oplus i}] \equiv \mathsf{b}(c_i^j[v_{j\oplus i}], t, \mathtt{r}_\mathrm{b}(n_{\mathrm{bi}}))$ where $\oplus$ is XOR. Then using the axiom IfBranch, it is sufficient to show the following equivalences:

$$
\begin{aligned}
&\vec{z}, b_0^0[v_0], b_1^0[v_1], \mathsf{acc}_0^0\ \&\ \mathsf{acc}_1^0,\ \ \mathsf{if}\ \mathsf{acc}_0^0\ \&\ \mathsf{acc}_1^0\ \mathsf{then}\ \big\langle u_0^0[v_0], u_1^0[v_1]\big\rangle\ \mathsf{else}\ \langle\bot,\bot\rangle \\
\sim\ &\vec{z}, b_0^1[v_1], b_1^1[v_0], \mathsf{acc}_0^1\ \&\ \mathsf{acc}_1^1,\ \ \mathsf{if}\ \mathsf{acc}_0^1\ \&\ \mathsf{acc}_1^1\ \mathsf{then}\ \big\langle u_1^1[v_0], u_0^1[v_1]\big\rangle\ \mathsf{else}\ \langle\bot,\bot\rangle
\end{aligned}
$$

which follows from the Blindness, UBNotUndefined and FuncApp; and the equivalences

$$
\begin{aligned}
&\vec{z}, b_0^0[v_0], b_1^0[v_1], \mathsf{acc}_0^0\ \&\ \mathsf{acc}_1^0,\ \ \mathsf{if}\ \mathsf{acc}_i^0\ \mathsf{then}\ u_i^0[v_i]\ \mathsf{else}\ \bot \\
\sim\ &\vec{z}, b_0^1[v_1], b_1^1[v_0], \mathsf{acc}_0^1\ \&\ \mathsf{acc}_1^1,\ \ \mathsf{if}\ \mathsf{acc}_{1-i}^1\ \mathsf{then}\ u_{1-i}^1[v_i]\ \mathsf{else}\ \bot
\end{aligned}
$$

which follow from CompHid and FuncApp. However, the axiom itself is not sufficient, we need the condition that the votes have equal length, otherwise these cannot be proven and an attack is constructed, which is the first attack described in Sect. 3.2.

Of course, in the actual security game, encryptions are sent too where the votes are also in reversed order, and decryptions are done by the mixer. This is dealt with the Enc$_{\mathrm{CCA2}}$ axiom. For the applicability of the axiom, the terms sent have to be rewritten using the equality to introduce cases when the adversarial function symbols on which decryptions are applied are either of the previous encryptions (namely the ones sent by voter A and B), or when they are neither. More precisely, a term of the form $\mathsf{dec}(f(\varPhi), \mathsf{sk}(n_1))$, assuming that $\varPhi$ has say two honest encryptions in them, $\{t\}_{\mathsf{pk}(n_1)}^{r(n)}$ and $\{t'\}_{\mathsf{pk}(n_1)}^{r(n')}$, has to be rewritten as

$$
\mathsf{dec}(f(\varPhi), \mathsf{sk}(n_1)) = \begin{array}{l}
\mathsf{if}\ \mathsf{EQ}(f(\varPhi), \{t\}_{\mathsf{pk}(n_1)}^{r(n)}) \\
\mathsf{then}\ t \\
\mathsf{else\ if}\ \mathsf{EQ}(f(\varPhi), \{t'\}_{\mathsf{pk}(n_1)}^{r(n')})\ \mathsf{then}\ t'\ \mathsf{else}\,\mathsf{dec}(f(\varPhi), \mathsf{sk}(n_1))
\end{array}.
$$

Thus, when the attacker messages are equal to some encrypted messages sent, the decryptions can be replaced by the plaintext. When they are not equal, then the decryption on the function symbol is kept. Once this is done, the Enc$_{\mathrm{CCA2}}$

axiom can be applied, and the plaintexts inside the encryptions can now be switched as long as they are of equal length. This is why we needed to assume that the unblinding results equal length. Without that assumption we obtain the second attack in Sect. 3.2.

Finally, when the decryptions are applied on the correct previous encryptions, there are still several possibilities for each decryption because a priori the adversary could redirect the same encryption twice. A priori, each decryption will succeed on any of the previous (four) encryptions sent by A and B. To ensure that there is no clash, we made sure that the mixer checks both the phase and that all the decryptions are different. Otherwise we obtain the third attack of Sect. 3.2. □

5.3 Attack Finding

Finally we comment on how we found the attacks presented in Sect. 3.2, which was actually rather simple for this protocol. We illustrate this on the first two attacks. Suppose that we do not require that the lengths of the names of the candidates are the same. At a certain point we arrive at having to verify that the messages sent by the challenger up to point 9 in Sect. 3.1 are equivalent in the two executions with flipped votes. This is the point when one of A or B according to the attacker's choice is supposed to have sent her ballot after having accepted the blind signature. As all the messages in the frames, this on both sides has the form of $\mathsf{if}\, c\, \mathsf{then}\, t_1\, \mathsf{else}\, t_2$, where c has all the conditions to get to this point, $t_2 = \mathbf{0}$, and t_1 has the encrypted ballot of either A or B depending on which of them was called first by the adversary, which in turn contains corresponding unflipped vote on one side and the flipped vote on the other side. To show that the two sides are equivalent, we would need to use the IND-CCA2 axiom for encryption, but that is only applicable if the equality of the lengths of the plaintexts are verified. However, without the formula $L(v_1) = L(v_2)$ this cannot be derived, and this gives our second attack through an Herbrand model. The first attack is obtained when we try to verify that the condition c on the two sides are equivalent. This condition contains the $\mathtt{acc}$ function symbol applied on the blind signature of v_1 or v_2, flipped on the two sides. The only axiom that is suitable to verify such difference (in the absence of encryption in c) the hiding of the commitment. Note, the blindness axiom can only be applied once both blind signatures were accepted, but at this point we only have one on each branch. The hiding of the commitment on the other hand can only be applied if the lengths are equal. As a result, we can set again $L(v_1) \neq L(v_2)$ and one of the accepts to ***true*** and the other to ***false***, and again an Herbrand model delivers a symbolic attack, which turns out to be a real computational one.

6 Conclusions

We analyzed the FOO electronic voting protocol for vote privacy in the provable security model using the computationally complete symbolic attacker (CCSA) framework. As part of the analysis, we showed that security properties of trapdoor commitments and blind signatures can be faithfully translated into axioms in the CCSA framework. We demonstrated that the framework is effective in that it revealed new attacks on the FOO protocol and could be used to prove the modified FOO protocol secure. As part of future work, we plan to investigate expressing and verifying stronger privacy properties of receipt-freeness and coercion-resistance for electronic voting protocols in the CCSA framework. We also plan to investigate automation of the verification tasks.

References

1. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: Seberry, J., Zheng, Y. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 244–251. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57220-1_66
2. Adida, B.: Helios: web-based open-audit voting. In: Proceedings of the 17th Conference on Security Symposium. SS 2008, pp. 335–348. USENIX Association (2008)
3. Ryan, P.Y.A., Rønne, P.B., Iovino, V.: Selene: voting with transparent verifiability and coercion-mitigation. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D., Brenner, M., Rohloff, K. (eds.) FC 2016. LNCS, vol. 9604, pp. 176–192. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53357-4_12
4. Kremer, S., Ryan, M.: Analysis of an electronic voting protocol in the applied pi calculus. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 186–200. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31987-0_14
5. Abadi, M., Blanchet, B., Fournet, C.: The applied pi calculus: mobile values, new names, and secure communication. J. ACM **65**, 1–41 (2017)
6. Delaune, S., Ryan, M., Smyth, B.: Automatic verification of privacy properties in the applied pi calculus. In: Karabulut, Y., Mitchell, J., Herrmann, P., Jensen, C.D. (eds.) IFIPTM 2008. ITIFIP, vol. 263, pp. 263–278. Springer, Boston, MA (2008). https://doi.org/10.1007/978-0-387-09428-1_17
7. Chadha, R., Cheval, V., Ciobâcă, Ş., Kremer, S.: Automated verification of equivalence properties of cryptographic protocols. ACM Trans. Comput. Log. **17**, 1–32 (2016)
8. Bana, G., Comon-Lundh, H.: A computationally complete symbolic attacker for equivalence properties. In: ACM Conference on Computer and Communications Security, pp. 609–620 (2014)
9. Bana, G., Comon-Lundh, H.: Towards unconditional soundness: computationally complete symbolic attacker. In: Degano, P., Guttman, J.D. (eds.) POST 2012. LNCS, vol. 7215, pp. 189–208. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28641-4_11
10. Bana, G., Adão, P., Sakurada, H.: Computationally complete symbolic attacker in action. In: Foundations of Software Technology and Theoretical Computer Science, pp. 546–560 (2012)
11. Bana, G., Adão, P., Sakurada, H.: Symbolic Verification of the Needham-Schroeder-Lowe Protocol (2012). http://web.ist.utl.pt/pedro.adao/pubs/drafts/nsl-long.pdf
12. Bana, G., Chadha, R.: Verification methods for the computationally complete symbolic attacker based on indistinguishability. Cryptology ePrint Archive, Report 2016/069 (2016). http://eprint.iacr.org/2016/069

13. Comon, H., Koutsos, A.: Formal computational unlinkability proofs of RFID protocols. In: IEEE Computer Security Foundations Symposium, pp. 100–114 (2017)
14. Scerri, G., Stanley-Oakes, R.: Analysis of key wrapping APIs: generic policies, computational security. In: IEEE Computer Security Foundations Symposium, pp. 281–295 (2016)
15. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**, 84–90 (1981)
16. Benaloh, J., Tuinstra, D.: Receipt-free secret-ballot elections (extended abstract). In: ACM Symposium on Theory of Computing, pp. 544–553 (1994)
17. Benaloh, J.D.C.: Verifiable Secret-ballot Elections. Ph.D. thesis, Yale University (1987)
18. Juels, A., Luby, M., Ostrovsky, R.: Security of blind digital signatures. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 150–164. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052233
19. Smyth, B., Bernhard, D.: Ballot secrecy and ballot independence coincide. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 463–480. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40203-6_26
20. Bernhard, D., Cortier, V., Galindo, D., Pereira, O., Warinschi, B.: Sok: a comprehensive analysis of game-based ballot privacy definitions. In: IEEE Symposium on Security and Privacy, pp. 499–516 (2015)
21. Cortier, V., Dragan, C.C., Dupressoir, F., Schmidt, B., Strub, P., Warinschi, B.: Machine-checked proofs of privacy for electronic voting protocols. In: IEEE Symposium on Security and Privacy, pp. 993–1008 (2017)
22. Fischlin, M., Schröder, D.: Security of blind signatures under aborts. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 297–316. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00468-1_17
23. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) Advances in Cryptology, pp. 199–203. Springer, Boston, MA (1983). https://doi.org/10.1007/978-1-4757-0602-4_18
24. Abdalla, M., Namprempre, C., Neven, G.: On the (im)possibility of blind message authentication codes. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 262–279. Springer, Heidelberg (2006). https://doi.org/10.1007/11605805_17
25. Smart, N.P.: Cryptography Made Simple. Information Security and Cryptography. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-21936-3
26. Damgård, I.B., Pedersen, T.P., Pfitzmann, B.: On the existence of statistically hiding bit commitment schemes and fail-stop signatures. J. Cryptol. **10**, 163–194 (1997)
27. Naor, M.: Bit commitment using pseudorandomness. J. Cryptol. **4**, 151–158 (1991)
28. Bana, G., Chadha, R., Eeralla, A.K.: Formal analysis of vote privacy using computationally complete symbolic attacker. Cryptology ePrint Archive, Report 2018/624 (2018). https://eprint.iacr.org/2018/624