Toward Formal Analysis of the IVXV Voting Protocol using Computationally Complete Symbolic Attacker*

Hideki Sakurada†

ZEN University, Zushi, Kanagawa, Japan
me@hidekisakurada.com

Abstract

The IVXV voting system is an electronic voting protocol currently used in Estonia. Due to its societal importance, IVXV has been the subject of extensive analysis, and several security and privacy issues have been identified. However, only a limited number of studies have applied formal verification to IVXV, despite the strong need for rigorous and systematic assurance of its security. Formal methods, particularly symbolic model approaches, represent exchanged messages as symbolic terms and define possible operations according to the assumed security of cryptographic primitives. This allows exhaustive verification of whether security and privacy requirements are satisfied under the presence of an active adversary. While symbolic approaches enable automated and comprehensive reasoning, they encounter limitations when dealing with protocols like IVXV that employ algebraic cryptographic primitives. Considering all algebraic operations available to an adversary often leads to non-termination, whereas restricting adversarial capabilities risks overlooking potential vulnerabilities. To address this challenge, the Computationally Complete Symbolic Attacker (CCSA) has been proposed, enabling semi-automatic verification without constraining adversarial power. In this work, we present an analysis of a fragment of the IVXV protocol within the CCSA framework and propose a verification methodology that advances the rigorous security analysis of IVXV.

1 Introduction

Elections are one of the most important processes of democracy. Given their importance, considerable effort is required to ensure secrecy and eligibility of votes in an election and the integrity of the election. To expand accessibility of elections, a number of online voting protocols have been proposed. The Helios voting protocol [16, 3] and the IVXV voting protocol, which was proposed in [15], are such voting protocols that are used in practice. In particular, IVXV is used in political elections in Estonia. Although such protocols have been extensively studied due to their importance, some attacks are discovered after the protocols are deployed in practice (e.g., [12, 17]). Since such protocols use cryptographic schemes, such as homomorphic public-key encryption schemes, analysis of such protocols requires knowledge of cryptographic theory, and even experts in this area of research may make mistakes in analysis. For more rigorous analysis of cryptographic protocols, the application of formal methods has been proposed and tools to automate the formal analysis have been developed (e.g., ProVerif [18] and Tamarin Prover [19]).

Recently, Baloglu et al. [6] analyzed various security properties required for IVXV using ProVerif and Tamarin, found several attacks on the properties, and proposed some improvements. However, in their analysis, attacks found by Müller [17] are out of scope. The reason is

*Proceedings of the 9th International Conference on Mobile Internet Security (MobiSec'25), Article No. 56, December 16-18, 2025, Sapporo , Japan.

†Corresponding author

that ProVerif and Tamarin are tools based on symbolic security models, in which the attacker is assumed to have limited computational ability for analyzing messages, because otherwise the automatic analysis tends not to terminate. To remove this limitation, the computationally complete symbolic attacker [9, 10, 7, 8] (CCSA) has been proposed. CCSA is a logical methodology in which the attacker is as strong as in the computational model used in cryptographic theoretical analysis. While ProVerif and Tamarin can analyze protocols fully automatically, analyses in CCSA are not fully automated. For (partial) automation, we may implement analyses in CCSA using general-purpose proof assistant software such as Coq [14], now known as Rocq [1], or the Squirrel prover [2, 5, 4], which has been developed for the CCSA logic.

In this paper, toward formal analysis of the IVXV protocol, we analyze a small fragment of IVXV in the computationally complete symbolic attacker for equivalence properties [10, 8], which was used for the analysis [7] of the FOO voting protocol. We also aim at analyzing the improvement of IVXV proposed by Müller for protecting against the shifting attacks he found. To this end, we introduce some function symbols for describing cryptographic primitives used in the fragment and introduce some axioms for the cryptographic primitives. Since we analyze only a small fragment, we still need a significant amount of effort. However, the function symbols and the axioms introduced in this paper are essential for the vote secrecy of the improvement Müller proposed. In addition to the analysis of the full version of IVXV, we still need to prove the computational soundness of the axioms, to show that the security proofs in our analysis imply the security in the computational model.

2 Computationally Complete Symbolic Attacker

We summarize the syntax of the computationally complete symbolic attacker (CCSA) for equivalence properties and informally explain the semantics, focusing only on what's necessary for this paper. For detailed and formal definitions, we refer the reader to [10] and [8].

The computationally complete symbolic attacker for equivalence properties is a logic for analyzing computational indistinguishability between two executions of cryptographic protocols. It enables us to analyze computational indistinguishability through symbolic (syntactical) manipulation of terms when possible. Terms are interpreted as bitstrings in the semantics but also express how the bitstrings are computed. It has the indistinguishability predicate on two sequences of terms, for expressing computational indistinguishability between two sequences of messages an attacker can see in two executions of a protocol.

2.1 Terms and Formulas

Let $\mathcal{X}$ be an infinite set of *variables*. Let $\mathcal{F}$ be an infinite set of *function symbols*. Any function symbol $f \in \mathcal{F}$ is associated with a natural number n, which we call the *arity* of f. We often indicate the arity by formal arguments; for example, $f(\_, \_)$ denotes a function symbol of arity 2. The set $\mathcal{T}$ of *terms* is defined by the following grammar:

$$t ::= f(t_1, \ldots, t_n) \mid \mathsf{if}\, t_0\, \mathsf{then}\, t_1\, \mathsf{else}\, t_2$$

where $f \in \mathcal{F}$ is any function symbol and n is the arity of f.

We assume that there is an infinite subset $\mathcal{N} \subseteq \mathcal{F}$ of function symbols with arity 0, which we call *names*. We also assume that there are at least the following function symbols:

- **True**, **False**, and **0** of arity 0
- **EQ**$(\_, \_)$, of arity 2

We will later further assume other function symbols.

Among other formulas, we only introduce the atomic formula for the indistinguishability relation because we focus on the indistinguishability relation in this paper. The set of *formulas* is the set of first-order formulas built from the atomic predicates on terms

$$t_1, \ldots, t_n \sim_n u_1, \ldots, u_n$$

where $t_1, \ldots, t_n$ and $u_1, \ldots, u_n$ are terms and n is a natural number. We omit the subscript n in $\sim_n$ since it is clear from the context.

We define the equality predicate $u = v$ by

$$u = v \quad := \quad \mathbf{EQ}(u, v) \sim \mathbf{True}.$$

2.2 Semantics

The semantics is defined as usual in the semantics of the first-order languages once we define the semantics of terms and the atomic formulas. A function symbol $f \in \mathcal{F}$ with arity n is interpreted as a polynomial-time algorithm, which outputs a bitstring on inputs n bitstrings. Such an algorithm does not share memory between executions or with other algorithms and may be randomly chosen in each execution of a protocol.

The basic function symbols introduced above are interpreted as follows:

- **True** and **False** are bitstrings for Boolean constants, **0** is an empty bitstring.
- **EQ** is the function that returns **True** if the two bitstrings are equal and **False** otherwise.

The conditional expression $\mathsf{if}\, u \,\mathsf{then}\, v \,\mathsf{else}\, w$ evaluates to the bitstring for v or w depending on whether the bitstring for u is the bitstring for **True** or not.

2.3 Axioms

We show some of the core axioms of CCSA. To analyze a protocol, we usually assume some more function symbols for cryptographic primitives used in the protocol and add some axioms for the function symbols. Some axioms are given by axiom schemata, in which x, y, u, v, t, n and etc. are meta variables for terms, and $\vec{u}$ and $\vec{v}$ are lists of terms.

The following axioms are the minimal set of core axioms used in this paper:

- The indistinguishability relation $\sim$ is an equivalence relation.
- The equality relation $=$ is a congruence relation.
- IfTrue: $\mathsf{if}\, \mathbf{True} \,\mathsf{then}\, x \,\mathsf{else}\, y = x$
- IfFalse: $\mathsf{if}\, \mathbf{False} \,\mathsf{then}\, x \,\mathsf{else}\, y = y$

These axioms are computationally sound, i.e., they are valid in the computational semantics.

Since the IVXV protocol uses a CPA-secure public-key encryption scheme, we introduce some function symbols for the scheme and add some axioms. We first assume function symbols $\mathbf{enc}(\_, \_, \_)$, $\mathbf{dec}(\_, \_)$, $\mathbf{ek}(\_)$, $\mathbf{dk}(\_)$, and $\mathbf{r}(\_)$ that are respectively for encryption, decryption, encryption key, decryption key, and randomness of encryption. We also assume function symbol $\mathbf{L}(\_)$ for the length of plaintexts. We then assume the following axioms. Here $\vec{t}[x]$ is a list of terms where a variable x occurs and $\vec{t}[w]$ is the list of terms obtained by replacing all occurrences of x in $\vec{t}[x]$ with a term w.

- EncDec: $\mathbf{dec}(\mathbf{dk}(x), \mathbf{enc}(\mathbf{ek}(x), t, \mathbf{r}(y))) = t$
- EncCPA:

$$\vec{t}[\mathsf{if}\ \mathbf{EQ}(\mathbf{L}(u), \mathbf{L}(u'))\ \mathsf{then}\ \mathbf{enc}(\mathbf{ek}(n_1), u, \mathbf{r}(n_2))\ \mathsf{else}\ v] \sim \vec{t}[\mathsf{if}\ \mathbf{EQ}(\mathbf{L}(u), \mathbf{L}(u'))\ \mathsf{then}\ \mathbf{enc}(\mathbf{ek}(n_1), u', \mathbf{r}(n_2))\ \mathsf{else}\ v]$$

  where $n_1, n_2 \in \mathcal{N}$ do not occur anywhere else.

Note that the side condition of EncCPA requires that no term of the form $\mathbf{dec}(\mathbf{dk}(n_1), \_)$ occurs in both sides of $\sim$.

For example, by assuming $\mathbf{L}(u) = \mathbf{L}(u')$ and using EncCPA we can prove the indistinguishability between two ciphertexts:

$$\begin{aligned}
&\mathbf{enc}(\mathbf{ek}(x), u, \mathbf{r}(y)) \\
&= \mathsf{if}\ \mathbf{True}\ \mathsf{then}\ \mathbf{enc}(\mathbf{ek}(x), u, \mathbf{r}(y))\ \mathsf{else}\ v && (\mathsf{IfTrue}) \\
&= \mathsf{if}\ \mathbf{EQ}(\mathbf{L}(u), \mathbf{L}(u'))\ \mathsf{then}\ \mathbf{enc}(\mathbf{ek}(x), u, \mathbf{r}(y))\ \mathsf{else}\ v && (\text{assumption}) \\
&\sim \mathsf{if}\ \mathbf{EQ}(\mathbf{L}(u), \mathbf{L}(u'))\ \mathsf{then}\ \mathbf{enc}(\mathbf{ek}(x), u', \mathbf{r}(y))\ \mathsf{else}\ v && (\mathsf{EncCPA}) \\
&= \mathsf{if}\ \mathbf{True}\ \mathsf{then}\ \mathbf{enc}(\mathbf{ek}(x), u', \mathbf{r}(y))\ \mathsf{else}\ v && (\text{assumption}) \\
&= \mathbf{enc}(\mathbf{ek}(x), u', \mathbf{r}(y)) && (\mathsf{IfTrue})
\end{aligned}$$

3 The IVXV Voting Protocol and the Shifting Attack

In this section, we briefly review the IVXV voting protocol and the shifting attacks discovered by Müller [17], following the description provided in [17]. The shifting attacks exploit the algebraic properties of the cryptographic primitives used in IVXV, allowing an adversary to manipulate encrypted votes in a way that shifts their meaning without detection and are considered to be a variant of the attacks [12, 13] on the Helios voting protocol. Specifically, Müller demonstrated that under certain conditions, an attacker can alter ciphertexts so that the decrypted result corresponds to a different candidate, thereby violating vote privacy. This highlights the necessity for formal verification techniques that can account for such algebraic manipulations in the analysis of voting protocols.

3.1 The IVXV Voting Protocol

The IVXV protocol assumes a homomorphic public-key encryption scheme, a digital signature scheme, and a zero-knowledge proof system. A homomorphic public-key encryption scheme is required because it allows reencryption of ciphertexts; someone who does not know the decryption key may produce a seemingly unrelated ciphertext of the same plaintext.

Following the description provided in [17], an execution of the IVXV protocol is divided into the following disjoint phases.

Setup Phase The *Election Organizer* (EO), the administrator of the election, determines the list of candidates and the list of voters. A public-key infrastructure (PKI) is assumed for identifying the voters by their verification keys. EO runs a key-generation algorithm of the public-key encryption scheme. The lists of the candidates and the eligible voters and the encryption keys are made public.

Submission phase Each voter who chooses some candidate ch encrypts their choice, then signs the ciphertext c with their secret signing key, and submits the ballot consisting of the ciphertext c and the signature σ to the *Vote Collector* (VC). For each ballot $b \leftarrow (c, \sigma)$, VC verifies whether σ is a valid signature on c signed by an eligible voter. If the verification succeeds, VC stores the ballot b with the time at which it had been submitted. Voters can re-vote multiple times.

Tabulation phase From VC, the *I-Ballot Box Processor* (IBBP) receives the list of ballots. IBBP first verifies that all ballots in the list are signed by eligible voters. IBBP then extracts the last submitted ballot of each voter, removes the respective signatures, and forwards the list B_1 of the resulting ciphertexts to the *Mixing Service* (MS). MS re-encrypts all ciphertexts in B_1, shuffles the resulting re-encrypted ciphertexts uniformly at random, and computes a proof of correct shuffling π_{Shuffle}. MS then sends the resulting ciphertexts B_2 and π_{Shuffle} to EO. EO decrypts the ciphertexts by using its secret key to obtain the final result *Res*, and produces a proof of correct decryption π_{Dec}. The tuple $(B_1, B_2, \pi_{\mathsf{Shuffle}}, \pi_{\mathsf{Dec}}, Res)$ is the outcome of the tabulation phase. Eventually EO publishes the result *Res* after removing the choices that are not in the list of candidates.

Auditing phase For the output $(B_1, B_2, \pi_{\mathsf{Shuffle}}, \pi_{\mathsf{Dec}}, Res)$, the *Data Auditor* (DA) verifies whether π_{Shuffle} is a valid proof of shuffle for B_1 and B_2 and whether π_{Dec} is a valid proof of correct decryption for B_2 and *Res*.

3.2 The Shifting Attacks

Müller found the *shifting attacks* and *encoding attacks* against the secrecy of the votes cast by the voters. Both attacks utilize the homomorphism of the encryption scheme. An encryption scheme is homomorphic if there is a binary operation $\odot$ on the space of ciphertexts and a binary operation $\cdot$ on the finite field on which plaintexts are encoded that satisfy $c_1 \odot c_2$ is a ciphertext of $m_1 \cdot m_2$ where c_1 and c_2 are respectively ciphertexts of m_1 and m_2.

In the shifting attack, we assume

- The attacker is a legitimate voter.
- The attacker can see a voter V's encrypted ballot c.
- There is a candidate ch' for whom no other voter votes.

In a shifting attack, the attacker guesses the candidate ch for whom the voter V votes and can determine whether V actually votes for ch. The attack proceeds as follows. The attacker computes a ciphertext c' by encrypting $ch^{-1} \cdot ch'$ where $(\_)^{-1}$ is the inverse operator of the finite field on which plaintexts are encoded. It then submits $c \odot c'$ with its signature to VC. Due to the homomorphism, $c \odot c'$ is a ciphertext of $ch \cdot (ch^{-1} \cdot ch') = ch'$, which is the candidate for whom no other voter cast a vote. When the result of the vote is published, if the attacker finds a vote for ch', it knows that V voted for ch.

To protect IVXV from such attacks, Müller suggested that a zero-knowledge proof of knowledge of the plaintext and the randomness used in the encryption should be sent with the encrypted ballots. In the attack above, the attacker cannot prove the knowledge of the randomness for the ciphertext $c \odot c'$ because it does not know the randomness for the ciphertext c.

4 Analysis of a fragment of IVXV

To show the ideas for the analysis of the IVXV protocol in CCSA, we consider a simplified version of IVXV, in which the roles EO, VC, IBBP, and MS are merged and the shuffling of ciphertexts and the auditing phase are omitted. Accordingly, we omit the signatures of the ballots and the proofs π_{Shuffle} and π_{Dec} verified in the auditing phase. More concretely, we assume

- Two honest voters A and B respectively vote for candidates $\mathbf{ch}_A$ and $\mathbf{ch}_B$.
- An attacker that can see the ballot sent by A and controls a voter C.
- A vote collector VC that decrypts the ballots and publishes the results after removing votes for invalid candidates.

In this setting, the vote secrecy is formalized as indistinguishability between the above execution and another execution in which the candidates that A and B vote for are swapped: A and B respectively vote for $\mathbf{ch}_B$ and $\mathbf{ch}_A$.

In the analysis below, we analyze the following cases:

- The voter C does not vote. In this case the adversary is simply an eavesdropper.
- The adversary controls C. In this case the shifting attack is possible.
- The adversary controls C but voters are required to send the proofs of knowledge as Müller suggested.

4.1 Case I: Passive adversary

We first consider the case where the adversary is passive, i.e., does not send messages. In this case, our goal is to prove

$$\phi_{init}, \phi_{vote}, \phi_{open} \sim \phi_{init}, \phi'_{vote}, \phi'_{open}$$

where

$$\begin{aligned}
\phi_{init} &:= \mathbf{ek}(T) \\
\\
vote_A &:= \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_A) \\
vote_B &:= \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_B, r_B) \\
\phi_{vote} &:= vote_A, vote_B \\
\\
vote'_A &:= \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_B, r_A) \\
vote'_B &:= \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_B) \\
\phi'_{vote} &:= vote'_A, vote'_B
\end{aligned}$$

and ϕ_{open} and ϕ'_{open} are shown later. In the above, $\mathbf{ek}(T)$ is the public key of EO, and ϕ_{vote} consists of the ballots of the two voters A and B who respectively vote for the candidates $\mathbf{ch}_A$ and $\mathbf{ch}_B$. Similarly, ϕ'_{vote} consists of the ballots of the two voters who respectively vote for the candidates $\mathbf{ch}_B$ and $\mathbf{ch}_A$.

ϕ_{open} and ϕ'_{open} are respectively the results of the opening of the ballots voted by the two voters in ϕ_{vote} and ϕ'_{vote}

$$\begin{aligned}\phi_{open} := \mathbf{count}_2(&\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote_A))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote_A)\ \mathsf{else}\ \mathbf{0},\\ &\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote_B))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote_B)\ \mathsf{else}\ \mathbf{0})\\ \phi'_{open} := \mathbf{count}_2(&\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote'_A))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote'_A)\ \mathsf{else}\ \mathbf{0},\\ &\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote'_B))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote'_B)\ \mathsf{else}\ \mathbf{0})\end{aligned}$$

where the function symbols **valid** and $\mathbf{count}_2$ are (semantically) for checking whether the argument is a legitimate candidate and for tabulating the votes while hiding the order of the arguments, respectively. For these function symbols we assume the axioms

$$\begin{aligned}\mathbf{valid}(\mathbf{ch}_A) &= \mathbf{True}\\ \mathbf{valid}(\mathbf{ch}_B) &= \mathbf{True},\end{aligned}$$

and

$$\mathbf{count}_n(c_1, \ldots, c_n) = \mathbf{count}_n(c'_1, \ldots, c'_n)$$

where n is a natural number and $c'_1, \ldots, c'_n$ are a permutation of $c_1, \ldots, c_n$. We hereafter omit the subscript n in $\mathbf{count}_n$.

In general, the indistinguishability between two lists of terms ψ_0 and ψ_n is usually proved by first proving the indistinguishability $\psi_i \sim \psi_{i+1}$ for $i = 0, \ldots, n-1$ and then obtaining $\psi_0 \sim \psi_n$ by transitivity.

Our proof of vote secrecy also proceeds similarly. We first replace the decryption of ciphertexts in ϕ_{vote} with the decrypted plaintexts. This is done by repeatedly applying the EncDec axiom to obtain

$$\phi_{init}, \phi_{vote}, \phi_{open} \sim \phi_{init}, \phi_{vote}, \phi^1_{open}$$

where

$$\begin{aligned}\phi^1_{open} := \mathbf{count}(&\mathsf{if}\ \mathbf{valid}(\mathbf{ch}_A)\ \mathsf{then}\ \mathbf{ch}_A\ \mathsf{else}\ 0,\\ &\mathsf{if}\ \mathbf{valid}(\mathbf{ch}_B)\ \mathsf{then}\ \mathbf{ch}_B\ \mathsf{else}\ 0)\end{aligned}$$

We then apply the axioms for the function symbol **valid** shown above together with IfTrue to obtain

$$\phi_{init}, \phi_{vote}, \phi^1_{open} \sim \phi_{init}, \phi_{vote}, \phi^2_{open}$$

where

$$\phi^2_{open} := \mathbf{count}(\mathbf{ch}_A, \mathbf{ch}_B)$$

We then apply the axiom for the function symbol **count** mentioned above to obtain

$$\phi_{init}, \phi_{vote}, \phi^2_{open} \sim \phi_{init}, \phi_{vote}, \phi^3_{open}$$

where

$$\phi^3_{open} := \mathbf{count}(\mathbf{ch}_B, \mathbf{ch}_A)$$

Now we will swap plaintexts in ϕ_{vote} to obtain ϕ'_{vote}. This is done by replacing the first plaintext $\mathbf{ch}_A$ with $\mathbf{ch}_B$ and then the second plaintext $\mathbf{ch}_B$ with $\mathbf{ch}_A$. This requires us to consider an intermediate list of messages

$$\phi''_{vote} := \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_A), \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_B).$$

Similarly to the example in Section 2.3, we have

$$\phi_{init}, \phi_{vote}, \phi^3_{open} \sim \phi_{init}, \phi''_{vote}, \phi^3_{open}$$

and

$$\phi_{init}, \phi''_{vote}, \phi^3_{open} \sim \phi_{init}, \phi'_{vote}, \phi^3_{open}.$$

Note that here we can apply EncCPA since $\mathbf{dec}(\mathbf{dk}(T), \_)$ does not occur elsewhere.

From the steps above, by transitivity, now we have

$$\phi_{init}, \phi_{vote}, \phi_{open} \sim \phi_{init}, \phi'_{vote}, \phi^3_{open}.$$

We can also perform similar steps for $\phi_{init}, \phi'_{vote}, \phi'_{open}$ but in the reverse order to obtain

$$\phi_{init}, \phi'_{vote}, \phi^3_{open} \sim \phi'_{vote}, \phi'_{open},$$

Finally, we have

$$\phi_{init}, \phi_{vote}, \phi_{open} \sim \phi_{init}, \phi'_{vote}, \phi'_{open}.$$

4.2 Case II: A voter is controlled by the adversary

We now consider the case where there is a third voter C who is controlled by the adversary, and the adversary can observe voter A's ballot. In this case, it should not be possible to prove vote secrecy because the shifting attack described in Section 3 is feasible in this fragment of the IVXV protocol. We shall demonstrate why the proof from the previous case fails here.

The vote secrecy property is specified as

$$\phi_{init}, \phi_{vote}, \phi_{open} \sim \phi_{init}, \phi'_{vote}, \phi'_{open}$$

where ϕ_{vote}, ϕ'_{vote}, ϕ_{open}, and ϕ'_{open} are extended as follows. Both ϕ_{vote} and ϕ'_{vote} are extended by adding voter C's ballot:

$$\phi_{vote} := vote_A, vote_B, f(\mathbf{ek}(T), vote_A)$$
$$\phi'_{vote} := vote'_A, vote'_B, f(\mathbf{ek}(T), vote'_A)$$

where f is a function symbol representing the attacker's computation.

Similarly, ϕ_{open} and ϕ'_{open} are extended by adding the opening of voter C's ballot:

$$\begin{aligned}
\phi_{open} := \mathbf{count}(&\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote_A))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote_A)\ \mathsf{else}\ \mathbf{0},\\
&\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote_B))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote_B)\ \mathsf{else}\ \mathbf{0}\\
&\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), f(\mathbf{ek}(T), vote_A)))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), f(\mathbf{ek}(T), vote_A))\ \mathsf{else}\ \mathbf{0})\\
\phi'_{open} := \mathbf{count}(&\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote'_A))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote'_A)\ \mathsf{else}\ \mathbf{0},\\
&\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote'_B))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote'_B)\ \mathsf{else}\ \mathbf{0}\\
&\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), f(\mathbf{ek}(T), vote'_A)))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), f(\mathbf{ek}(T), vote'_A))\ \mathsf{else}\ \mathbf{0})
\end{aligned}$$

Following a similar approach to the proof in Section 4.1, the vote privacy reduces to

$$\phi_{init}, \phi_{vote}, \phi^2_{open} \sim \phi_{init}, \phi'_{vote}, \phi'^2_{open}$$

where

$$\begin{aligned}
\phi^2_{open} := \mathbf{count}(&\mathbf{ch}_A, \mathbf{ch}_B, \\
&\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), f(\mathbf{ek}(T), vote_A))) \\
&\mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), f(\mathbf{ek}(T), vote_A)) \\
&\mathsf{else}\ \mathbf{0}) \\
\phi'^2_{open} := \mathbf{count}(&\mathbf{ch}_B, \mathbf{ch}_A \\
&\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), f(\mathbf{ek}(T), vote'_A))) \\
&\mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), f(\mathbf{ek}(T), vote'_A)) \\
&\mathsf{else}\ \mathbf{0})
\end{aligned}$$

We cannot rewrite this further by applying EncDec because there is no ciphertext directly under **dec**. Since **dec** remains in both ϕ^2_{open} and ϕ'^2_{open}, and due to the side condition of EncCPA, we cannot apply EncCPA to replace the plaintexts occurring in $vote_A$ and $vote'_A$. This makes it impossible to prove vote secrecy, which is consistent with the fact that the shifting attack is indeed feasible for this fragment of the IVXV protocol.

One might consider whether assuming a stronger axiom for the encryption scheme, such as EncCCA from [8], would allow us to prove the above indistinguishability. However, since the (full) IVXV protocol requires the encryption scheme to be rerandomizable, we cannot assume CCA security, which is necessary for the computational soundness of EncCCA.

4.3 Case III: Extension with proof of possession of plaintexts

To protect vote secrecy against the shifting attack as well as some other attacks, Müller [17] suggested that each voter computes non-interactive zero-knowledge proof of knowledge (NIZK PoK) which proves that the voter knows the plaintext of the encrypted ballot. To incorporate this extension, we extend our fragment as follows.

We first let the voters send the proofs of knowledge $\mathbf{proof}(vote_i, \mathbf{ek}(T), \mathbf{ch}_i, r_i)$ as follows:

$$\begin{aligned}
vote_A &:= \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_A), \mathbf{proof}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_A), \mathbf{ek}(T), \mathbf{ch}_A, r_A) \\
vote_B &:= \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_B, r_B), \mathbf{proof}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_B, r_B), \mathbf{ek}(T), \mathbf{ch}_B, r_B) \\
vote_C &:= f(\mathbf{ek}(T), vote_A), g(\mathbf{ek}(T), vote_A)
\end{aligned}$$

The voter C, which is controlled by the adversary, also sends a message $g(\mathbf{ek}(T), vote_A)$ for the proof. When EO opens the ballots, it verifies the proofs of knowledge, and then counts valid

votes:

$$\begin{aligned}
open_A &:= \mathsf{if}\ \mathbf{verify}(\mathbf{ek}(T), vote_A) \\
&\quad \mathsf{then}\ \mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote_A))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote_A)\ \mathsf{else}\ 0 \\
&\quad \mathsf{else}\ 0 \\
open_B &:= \mathsf{if}\ \mathbf{verify}(\mathbf{ek}(T), vote_B) \\
&\quad \mathsf{then}\ \mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote_B))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote_B)\ \mathsf{else}\ 0 \\
&\quad \mathsf{else}\ 0 \\
open_C &:= \mathsf{if}\ \mathbf{verify}(\mathbf{ek}(T), vote_C) \\
&\quad \mathsf{then}\ \mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote_C))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote_C)\ \mathsf{else}\ 0 \\
&\quad \mathsf{else}\ 0 \\
\phi_{open} &:= \mathbf{count}(open_A, open_B, open_C)
\end{aligned}$$

For the function symbols **proof** and **verify**, we assume the following axioms, which correspond to the security properties required for NIZK PoK.

- Zero-knowledge (ZK): We assume that there is a function symbol $\mathbf{proof_{sim}}$ and the axiom

  $$\mathbf{proof}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_i, r_i), \mathbf{ek}(T), \mathbf{ch}_i, r_i) = \mathbf{proof_{sim}}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_i, r_i.))$$

  This axiom corresponds to the zero-knowledge property of the NIZK PoK: There is a simulator that, without having the knowledge, can generate a simulated proof that is statistically indistinguishable from the honestly generated proof.

- Simulation soundness (ZKsound): We assume that there is a function symbol **ext** and the axiom

  $$\begin{aligned}
  &\mathsf{if}\ \mathbf{verify}(\mathbf{ek}(T), c, p)\ \mathsf{then}\ u[\mathbf{dec}(\mathbf{dk}(T), c)]\ \mathsf{else}\ v \\
  &= \mathsf{if}\ \mathbf{verify}(\mathbf{ek}(T), c, p)\ \mathsf{then}\ u[\mathbf{ext}(\mathbf{ek}(T), p)]\ \mathsf{else}\ v
  \end{aligned}$$

  where the term $u[\mathbf{dec}(\mathbf{dk}(T), c)]$ is a term containing the subterm $\mathbf{dec}(\mathbf{dk}(T), c)$ and $u[\mathbf{ext}(\mathbf{ek}(T), p)]$ is the term obtained by replacing all occurrences of $\mathbf{dec}(\mathbf{dk}(T), c)$ with $\mathbf{ext}(\mathbf{ek}(T), p)$ in $u[\mathbf{dec}(\mathbf{dk}(T), c)]$. This axiom corresponds to the simulation soundness of the NIZK PoK of plaintext: There is a knowledge extractor that extracts the knowledge from any valid proof.

- Completeness (ZKcomplete)

  $$\mathbf{verify}(\mathbf{ek}(T), \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_i, r_i), \mathbf{proof}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_i, r_i), \mathbf{ek}(T), \mathbf{ch}_i, r_i)) = \mathbf{True}$$

  This axiom corresponds to the completeness of the NIZK PoK: An honestly generated proof is verified with probability 1.

Here the function symbols $\mathbf{proof_{sim}}$ and **ext** represent the simulator and the knowledge extractor of the underlying zero-knowledge proof of knowledge scheme, respectively.

We also extend ϕ'_{vote} and ϕ'_{open} similarly. Then the vote secrecy is defined as:

$$\phi_{init}, \phi_{vote}, \phi_{open} \sim \phi_{init}, \phi'_{vote}, \phi'_{open}$$

Its proof proceeds as follows.

Using the axioms ZKcomplete and IfTrue, we first have

$$\phi_{init}, \phi_{vote}, \phi_{open} \sim \phi_{init}, \phi_{vote}, \phi^1_{open}$$

where

$$\begin{aligned}\phi^1_{open} := \mathbf{count}(&\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote_A))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote_A)\ \mathsf{else}\ \mathbf{0},\\ &\mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote_B))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote_B)\ \mathsf{else}\ \mathbf{0},\\ &open_C.)\end{aligned}$$

Similarly to the proof in Section 4.1, we also have

$$\phi_{init}, \phi_{vote}, \phi^1_{open} \sim \phi_{init}, \phi_{vote}, \phi^2_{open}$$

where

$$\phi^2_{open} := \mathbf{count}(\mathbf{ch}_A, \mathbf{ch}_B, open_C.)$$

By applying the axiom ZK, we respectively replace the proofs

$$\begin{aligned}&\mathbf{proof}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_A), \mathbf{ek}(T), \mathbf{ch}_A, r_A),\ \text{and}\\ &\mathbf{proof}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_B, r_B), \mathbf{ek}(T), \mathbf{ch}_B, r_B)\end{aligned}$$

with the simulated proofs

$$\begin{aligned}&\mathbf{proof_{sim}}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_A)),\ \text{and}\\ &\mathbf{proof_{sim}}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_B, r_B))\end{aligned}$$

in ϕ_{vote} and ϕ^2_{open}, and then have

$$\phi_{init}, \phi_{vote}, \phi^2_{open} \sim \phi_{init}, \phi^3_{vote}, \phi^3_{open}$$

where

$$\begin{aligned}vote^3_A &:= \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_A), \mathbf{proof_{sim}}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_A))\\ vote^3_B &:= \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_B, r_B), \mathbf{proof_{sim}}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_B, r_B))\\ vote^3_C &:= f(\mathbf{ek}(T), vote^3_A), g(\mathbf{ek}(T), vote^3_A)\\ \phi^3_{vote} &:= vote^3_A, vote^3_B, vote^3_C\end{aligned}$$

$$\begin{aligned}open^3_C :=&\\ &\mathsf{if}\ \mathbf{verify}(\mathbf{ek}(T), vote^3_C)\\ &\mathsf{then}\ \mathsf{if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote^3_C))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote^3_C)\ \mathsf{else}\ 0\\ &\mathsf{else}\ 0\\ \phi^3_{open} &:= \mathbf{count}(\mathbf{ch}_A, \mathbf{ch}_B, open^3_C.)\end{aligned}$$

We then apply the axiom ZKsoundness and have

$$\phi_{init}, \phi^3_{vote}, \phi^3_{open} \sim \phi_{init}, \phi^3_{vote}, \phi^4_{open}$$

where

$$\begin{aligned}\phi^4_{open} := \mathbf{count}(&\mathbf{ch}_A, \mathbf{ch}_B,\\ &\mathsf{if}\ \mathbf{verify}(\mathbf{ek}(T), vote^3_C)\\ &\mathsf{then\ if}\ \mathbf{valid}(\mathbf{ext}(g(\mathbf{ek}(T), vote^3_A)))\ \mathsf{then}\ \mathbf{ext}(g(\mathbf{ek}(T), vote^3_A))\ \mathsf{else}\ 0\\ &\mathsf{else}\ 0)\end{aligned}$$

Since no decryption occurs in ϕ_{init}, ϕ^3_{vote}, and ϕ^4_{open}, we can apply axiom EncCPA, as in Section 4.1, to replace the two ciphertexts $\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_A)$ and $\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_B, r_B)$ with one another in ϕ^3_{vote} and ϕ^4_{open}, and have

$$\phi_{init}, \phi^3_{vote}, \phi^4_{open} \sim \phi_{init}, \phi^5_{vote}, \phi^5_{open}.$$

where

$$\begin{aligned}vote^5_A &:= \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_B, r_A), \mathbf{proof}_{\mathbf{sim}}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_A))\\ vote^5_B &:= \mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_A, r_B), \mathbf{proof}_{\mathbf{sim}}(\mathbf{enc}(\mathbf{ek}(T), \mathbf{ch}_B, r_B))\\ vote^5_C &:= f(\mathbf{ek}(T), vote^5_A), g(\mathbf{ek}(T), vote^5_A)\\ \phi^3_{vote} &:= vote^5_A, vote^5_B, vote^5_C\end{aligned}$$

$$\begin{aligned}open^5_C :=&\\ &\mathsf{if}\ \mathbf{verify}(\mathbf{ek}(T), vote^5_C)\\ &\mathsf{then\ if}\ \mathbf{valid}(\mathbf{dec}(\mathbf{dk}(T), vote^5_C))\ \mathsf{then}\ \mathbf{dec}(\mathbf{dk}(T), vote^5_C)\ \mathsf{else}\ 0\\ &\mathsf{else}\ 0\\ \phi^5_{open} &:= \mathbf{count}(\mathbf{ch}_A, \mathbf{ch}_B, open^5_C.)\end{aligned}$$

Now, by transitivity, it remains to show

$$\phi_{init}, \phi^5_{vote}, \phi^5_{open} \sim \phi_{init}, \phi'_{vote}, \phi'_{open}$$

which is proved first by applying the axiom for **count** and then by similarly performing the derivation above.

5 Conclusion

In this paper, toward the formal analysis of the IVXV voting protocol, we have analyzed a fragment of IVXV and an improved version using the computationally complete symbolic attacker. To this end, we have introduced function symbols for representing the cryptographic primitives used in IVXV and axioms for these primitives. Since we have analyzed only a small fragment, a significant amount of effort is still needed for analyzing the full version. The proof in this paper is hand-written, and since such proofs may contain errors, it is desirable to automate the construction and verification of the proof for the full version. We also need to prove the computational soundness of the axioms to guarantee that our security proof implies security in the computational model. It is also interesting to analyze other voting protocols such as Helios and Belenios [11] using CCSA because these protocols use zero-knowledge proof systems and the function symbols and axioms for NIZK PoK may be potentially adapted to these analyses.

References

[1] The Rocq prover. https://rocq-prover.org/.

[2] Squirrel prover. https://squirrel-prover.github.io/.

[3] Ben Adida. Helios: web-based open-audit voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, page 335–348, USA, 2008. USENIX Association.

[4] David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, and Joseph Lallemand. The Squirrel Prover and its Logic. *ACM SIGLOG News*, 11(2), April 2024.

[5] David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, and Solène Moreau. An Interactive Prover for Protocol Verification in the Computational Model. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'21)*, Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'21), San Fransisco / Virtual, United States, May 2021.

[6] Sevdenur Baloglu, Sergiu Bursuc, Sjouke Mauw, and Jun Pang. Formal verification and solutions for Estonian E-voting. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, New York, NY, USA, July 2024. ACM.

[7] Gergei Bana, Rohit Chadha, and Ajay Kumar Eeralla. Formal analysis of vote privacy using computationally complete symbolic attacker. In *Computer Security: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*, page 350–372, Berlin, Heidelberg, 2018. Springer-Verlag.

[8] Gergei Bana, Rohit Chadha, Ajay Kumar Eeralla, and Mitsuhiro Okada. Verification methods for the computationally complete symbolic attacker based on indistinguishability. *ACM Trans. Comput. Logic*, 21(1), October 2019.

[9] Gergei Bana and Hubert Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In Pierpaolo Degano and Joshua D. Guttman, editors, *Principles of Security and Trust*, pages 189–208, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[10] Gergei Bana and Hubert Comon-Lundh. A computationally complete symbolic attacker for equivalence properties. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 609–620, New York, NY, USA, 2014. Association for Computing Machinery.

[11] Véronique Cortier, Pierrick Gaudry, and Stephane Glondu. Belenios: a simple private and verifiable electronic voting system. In Joshua D. Guttman, Carl E. Landwehr, José Meseguer, and Dusko Pavlovic, editors, *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows*, volume 11565 of *LNCS*, pages 214–238. Springer, 2019.

[12] Veronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. In *Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium*, CSF '11, page 297–311, USA, 2011. IEEE Computer Society.

[13] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. *J. Comput. Secur.*, 21(1):89–148, January 2013.

[14] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Parent C. Murthy, C. Paulin-Mohring, and B. Werner. The Coq proof assistant user guide. Rapport INRIA 154, INRIA, 1993.

[15] Sven Heiberg, Tarvi Martens, Priit Vinkel, and Jan Willemson. Improving the verifiability of the estonian internet voting scheme. In *International Joint Conference on Electronic Voting*, 2016.

[16] Helios Voting. https://vote.heliosvoting.org/.

[17] Johannes Müller. Breaking and fixing vote privacy of the estonian e-voting protocol ivxv. In Shin'ichiro Matsuo, Lewis Gudgeon, Ariah Klages-Mundt, Daniel Perez Hernandez, Sam Werner, Thomas Haines, Aleksander Essex, Andrea Bracciali, and Massimiliano Sala, editors, *Financial Cryptography and Data Security. FC 2022 International Workshops*, pages 325–334, Cham, 2023. Springer International Publishing.

[18] ProVerif: Cryptographic protocol verifier in the formal model. https://bblanche.gitlabpages.inria.fr/proverif/.

[19] Tamarin prover. https://tamarin-prover.com/.